http://www.trakya.edu.tr/Enstituler/FenBilimleri/Dergi/net/index.htm
Trakya Üniversitesi Bilimsel Araştırmalar Dergisi
B Serisi Fen Bilimleri, 4(1): 37-45, 2003
ISSN 1302 647X
DIC: 77AMACET*/4106030703*

Derleme/Review

DAĞITIK NESNE YÖNETİMİ MİMARİLERİNDEN CORBA VE DCOM MİMARİLERİNİN KARŞILAŞTIRMASI

Altan MESUT, Aydın CARUS
Trakya Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, EDİRNE, e-mail: altanmesut@trakya.edu.tr

Alınış : 08.11.2002
Kabul ediliş : 14.03.2003

Özet: Son on yılda çıkan nesneye yönelik programlama ve dağıtık sistem teknolojileri modern yazılımları büyük ölçüde etkiledi. RPC (Remote Procedure Call – Uzak Prosedür Çağrısı) gibi eski nesil istemci/sunucu mimarileri nesneye yönelik bir modele sahip değildirler. Bu mimaride istemci, sunucuya nasıl ulaşması gerektiğini ve sunucunun yerini bilmek, ve kodu eklenecek her yeni servis için değiştirilmek zorundadır. İstemci/sunucu sistem teknolojisinin gelişiminin bir sonraki basamağı, dağıtık sistem ile nesneye yönelik programlama teknolojilerinin birleşimi olan dağıtık nesne yönetimi sistemleridir. Dağıtık sistemlerin gerçek faydası, ancak karmaşık uygulamaların yeniden kullanılabilir yazılım bileşenleri kullanılarak meydana getirilmesine izin veren, dağıtık nesne yönetimi sistemlerinin kullanılması ile mümkün olacaktır. Bu çalışmada dağıtık nesne yönetimi mimarilerinden COM/DCOM ve CORBA incelenmiştir.

Anahtar Kelimeler: RPC, COM, OMG, Microsoft

Comparison of distributed object management architectures CORBA and DCOM

Abstract: The object-oriented programming and distributed computing techniques made significant impact on modern software development over the past ten years. Older generation client/server architectures, such as RPC (Remote Procedure Call), do not have an object-oriented model. In these architectures, the client must be aware of where the server is located and how to access the server, and its code must be modified to make use of new services that become available. The next step in the evolution of the client/server architecture is the distributed object management systems, which is the union of object-oriented programming and distributed computing. The significant promise of this technology is that it enables construction of complex applications from reusable software components.

Key Words: RPC, COM, OMG, Microsoft

Giriş

Bu çalışmanın amacı, değişik zamanlarda, farklı platformlar (işletim sistemleri & donanımlar) ve farklı programlama dilleri ile, birbirinden bağımsız olarak tasarlanmış yazılım bileşenlerinin, bir bütün olarak çalışabilmesi için kullanılan dağıtık nesne yönetimi mimarilerinden CORBA (Common Object Request Broker Architecture – Genel Nesne İstek Aracı Mimarisi) ve DCOM (Distributed Component Object Model – Dağıtık Bileşen Nesne Modeli) mimarilerinin incelenmesi, aralarındaki benzerlikler ve farklılıkların belirlenmesidir.

Bu benzerlik ve farklılıklar göz önüne alınarak, geliştirilecek olan uygulamaların daha etkin olarak tasarlanması sağanabilecektir.

DAĞITIK UYGULAMA TASARLAMAK

Dağıtık uygulamaları tasarlayıp yürütürken, tasarımcıya değişik uygulama geliştirme ortamları sunulur. Bu seçenekler uygulamanın geliştirildiği ortamdan, yürütme için kullanılan dilinin seçimine kadar farklılıklar gösterebilir. Şimdi bu farklı ortamları kısaca inceleyelim:

Soket Programlama

Modern sistemlerin çoğunda, makineler arasındaki ve bazen aynı makinedeki işlemler arasındaki iletişim, soketler kullanılarak yapılır. Bir soket uygulamaların kendi aralarında bağlanabilmeleri ve haberleşebilmeleri için bir kanaldır (ROSENBERGER, 1998). Uygulama bileşenleri arasında haberleşmeyi sağlamak için en uygun yol soketleri direkt olarak kullanmaktır (bu soket programlama olarak bilinir). Geliştirici veriyi sokete yazar ve/veya veriyi soketten okur.

Soket Programlama'da kullanılan Uygulama Programlama Arayüzü (API – Application Programming Interface), oldukça düşük seviyelidir. Bu yüzden soket programlama karmaşık uygulamalar için uygun değildir. Örneğin uygulama bileşenleri farklı tipten makinelerde bulunuyorlarsa veya farklı programlama dillerinde

yürütülmüşlerse, karmaşık veri tiplerini elde etmek için soket programlama uygun değildir. Direkt soket programlama çok ufak ve etkili uygulamalarda sonuç verebilir, ama genel kanı karmaşık uygulama geliştirmesi için uygun olmadığıdır.

RPC - Remote Procedure Call

Soket programlamanın bir üst basamağı RPC'dir. RPC, soket seviyesindeki iletişimlere fonksiyon-tabanlı bir arayüz sağlar. RPC kullanarak, verinin akarak bir soket oluşturmasını direkt olarak sağlamak yerine, geliştirici, C gibi fonksiyonel dillerdekine benzer şekilde bir fonksiyon tanımlayıp, bu fonksiyonun çağırana normal bir fonksiyon gibi görünmesini sağlayan bir kod oluşturur. Fonksiyon, uzaktaki bir sunucuyla (remote server) iletişim kurmak için, arka planda soketleri kullanır (ROSENBERGER, 1998).

Fonksiyon-tabanlı bir arayüz sağladığı için, RPC'yi kullanmak çoğu kez ham soket programlamayı kullanmaktan daha kolaydır. RPC aynı zamanda, birçok istemci/sunucu uygulamaları için temel oluşturabilecek kadar güçlüdür. RPC protokolünün uyumsuz birçok yürütmesi olmasına rağmen, birçok platforma uyumlu olan standart bir RPC protokolü vardır.

DCE – Distributed Computing Environment

OSF (Open Software Foundation – Açık Yazılım Kuruluşu) tarafından geliştirilmiş bir yöntem olan DCE (Dağıtık Hesaplama Ortamı), daha çok dağıtık ortamlar için çeşitli standartlar tanımlamak amacıyla üretilmiştir. Bu standartları tanımlarken de RPC'yi kullanmıştır. DCE standardı bir süre kullanılmış, fakat tam sonuç alınamadan yerine geçecek değişik yöntemlerin çıkması (CORBA, DCOM) sonucu, bugün çok az uygulamada kullanılmaktadır. Microsoft COM'u geliştirirken, DCE RPC'yi temel almıştır.

CORBA – Common Object Request Broker Architecture

OMG (Object Management Group - Nesne Yönetim Grubu) tarafından geliştirilen CORBA, bileşenler arasındaki arayüzleri tanımlamak için standart bir mekanizmaya sahiptir, ve bu arayüzlerin gerçeklenmesini kolaylaştırmak için geliştiricinin seçeceği programlama dilini kullanan bazı araçlar içerir. CORBA, bir uygulamanın veya farklı uygulamaların çeşitli bileşenlerinin, kendi aralarında haberleşebilmeleri için gerekli olan tüm altyapıyı sağlar. OSF'den farklı olarak, OMG gerçek yazılım üretmez, sadece ayrıntılı tanımlar üretir (VINOSKI, 1993).

CORBA, bilgisayar yazılım dünyasında nadir olarak sağlanabilen "platform bağımsızlığı" ve "dil bağımsızlığı" özelliklerini içerir. Platform bağımsızlığı, üzerinde CORBA ORB yürütmesi olan her türlü platform (modern işletim sistemleri) üzerinde CORBA nesnelerinin kullanılabilmesi demektir. Dil bağımsızlığı ise, CORBA nesnelerinin hemen hemen istenilen tüm programlama dillerinde gerçeklenebileceği, haberleştikleri diğer CORBA nesnelerinin hangi dili kullandıklarını bilmek zorunda olmadıkları anlamına gelir.

DCOM – Microsoft Distributed Component Object Model

1993 yılında duyurduğu COM (Component Object Model – Bileşen Nesne Modeli) ile farklı sistemler üzerindeki bileşenlere erişim sağlayan Microsoft, 1996 yılında duyurduğu DCOM modeli ile de bileşenlerden oluşan ağ uygulamaları yaratmayı mümkün hale getirerek, dağıtık sistemler piyasasına giriş yapmıştır. DCOM, CORBA ile benzer imkanlara sahiptir. DCOM Microsoft işletim sistemlerine (Windows 9x, NT, 2000) entegre edilmiştir. Fakat Windows işletim sistemleri dışında seyrek kullanılır.

CORBA-DCOM köprüleri sayesinde CORBA nesneleri DCOM nesneleri ile, ve DCOM nesneleri de CORBA nesneleri ile haberleşebilirler. Mevcut uyumsuzluklar nedeni ile iki sistemi uzlaştırmak zordur. Bu köprüler her ne kadar mükemmel çözüm olmasalar da, DCOM ve CORBA nesnelerinin bir arada kullanılması gerektiği durumlar için elverişlidirler.

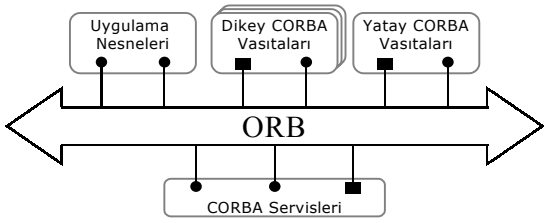
RMI – Java Remote Method Invocation

RMI, bir kaç özelliği dışında CORBA'ya çok benzer. RMI'nin avantajlarından biri nesneleri CORBA'nın aksine referans olarak değil de değer olarak geçirmesidir. (CORBA'nın 2.0 sürümünden itibaren nesnelerin değer olarak geçirilmesine destek verilmiştir.) Ancak RMI'nin en büyük dezavantajı sadece Java destekli olmasıdır. Yani hem istemci hem sunucu tarafı programları tamamen Java ile yazılmış olmalıdır (ROSENBERGER, 1998).

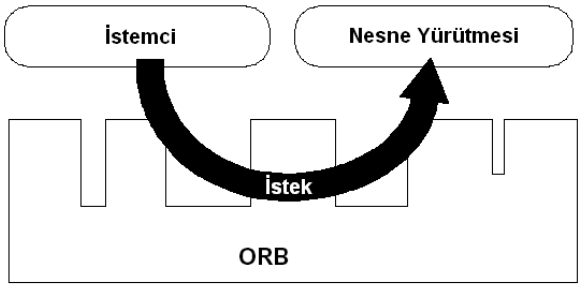
CORBA – DCOM KARŞILAŞTIRMASI

CORBA Hakkında Genel Bilgi

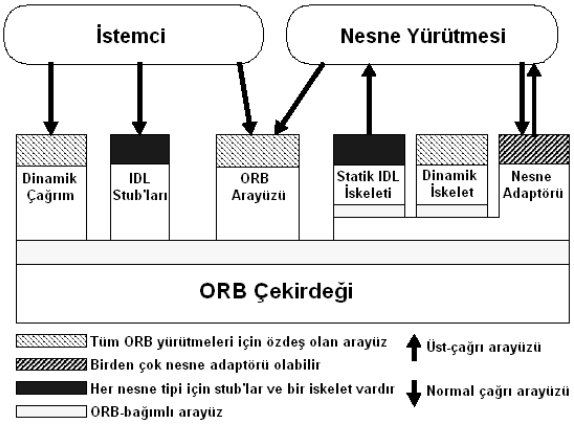
CORBA, OMG (The Object Management Group) tarafından geliştirilmiştir. OMG, Nisan 1989'da, içlerinde 3Com Corporation, American Airlines, Canon, Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems ve Unisys Corporation'ın da bulunduğu 11 şirket ortaklığında, dağıtık nesneye yönelik uygulamaların beraber çalışabilirliğini ve taşınabilirliğini sağlayan standartları oluşturmak amacıyla kurulmuştur. Şu anda 800 kadar üyesi olan konsorsiyum, yazılım endüstrisi için firma bağımsız ayrıntılı tanımlar geliştirmeye devam



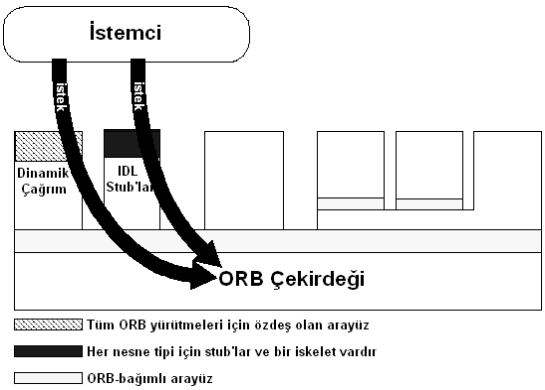
Şekil 1. OMA Referans Modeli



Şekil 2. ORB Üzerinden Gönderilen Bir İstek



Şekil 3. Nesne İstek Arayüzlerinin Yapısı



Şekil 4. Stub'ı veya Dinamik Çağrı Arayüzünü Kullanan Bir İstemci.

etmektedir. Günümüzde birçok işletim sisteminde bu ayrıntılı tanımların olduğunu görebilirsiniz. Bu ayrıntılı tanımlar, Dağıtık Nesne Hesaplamaları için gerekli standart arayüzleri, detayı ile anlatır.

OMG'nin 1989 yılındaki kuruluşunu takiben, Aralık 1990 yılında CORBA 1.0 piyasaya sürülmüştür. Bunu, 1991 yılının başlarında, IDL (Interface Definition Language – Arayüz Tanımlama Dili) ve uygulamaların bir ORB ile haberleşebilmeleri için API'lerin tanımlandığı CORBA 1.1 takip etmiştir. CORBA 1.x sürümleri ile birlikte farklı mimarilerde, farklı makinelerde ve farklı dillerde yazılmış nesnelerin birbirleriyle haberleşmelerini sağlayarak nesneler arası ilişkilere ayrı bir boyut kazandırmıştır.

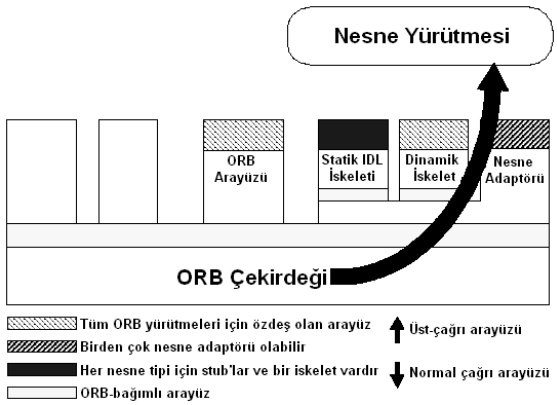
CORBA 1.x sürümleri dağıtık nesnelerin ilişkilendirilmesi açısından atılmış önemli adımlardı, fakat bazı eksiklikleri vardı. IDL için ve bir uygulama üzerinden bir ORB'ye erişmek için standartlar tanımlanmış olsa da, en büyük eksiklik ORB'ların birbirleriyle haberleşebilmeleri için standart bir protokol tanımlanmamış olmasıydı. Bu yüzden bir üreticinin ürettiği bir ORB ile farklı bir üreticinin ORB'u haberleşemeyecekti. Bu da dağıtık sistemlerin amacına sınırlama getirmekteydi.

Aralık 1994'de tamamlanan CORBA 2.0 sürümünün en büyük başarısı, farklı ORB'ler arası haberleşmenin sağlanabilmesi için standart bir protokolün tanımlanmış olmasıydı. IIOP (Internet InterORB Protocol) olarak adlandırılan bu protokol sayesinde CORBA uygulamaları daha fazla üreticiden bağımsız hale geldiler. IIOP protokolü sadece, Internet ve birçok intraneti bünyesinde barındıran TCP/IP tabanlı ağlarda kullanılabilir.

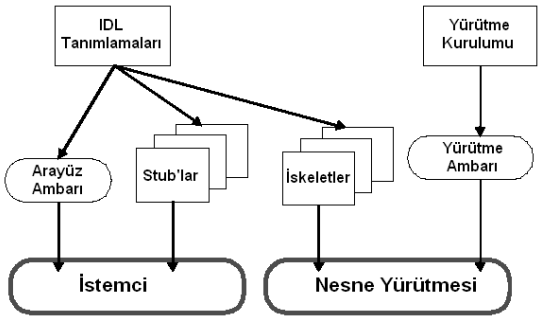
CORBA, OMA'nın (Object Management Architecture - Nesne Yönetim Mimarisi) bir parçasıdır (Şekil 1). OMA, heterojen bir ortamda nesnelerin nasıl dağıtıldığını tanımlayan bir Nesne Modeli ve bu nesneler arasındaki etkileşimleri tanımlayan bir Referans Modeli'nden oluşur (VINOSKI, 1997). CORBA'nın OMA içerisindeki görevi ORB (Object Request Broker – Nesne İstek Aracı) fonksiyonlarını yerine getirmektir.

ORB

ORB, istek için nesne yürütmesini bulmada, nesne yürütmesini isteği almak için hazırlamada, ve isteği oluşturan veri ile iletişim kurmada gerekli tüm mekanizmalardan sorumludur. İstemci arayüzünün görünüşü, nesnenin nerde konumlandığından, hangi programlama dilinde yürütüldüğünden, veya nesnenin arayüzüne yansımayan herhangi başka bir durumdan tamamen bağımsızdır (OMG, 2001).



Şekil 5. İstek Alan Bir Nesne Yürütmesi



Şekil 6. Arayüz ve Yürütme Ambarları

Şekil 2, bir istemci tarafından bir nesne yürütmesine gönderilmiş bir isteği gösterir. İstemci, nesne üzerinde bir işlem yapmak isteyen varlıktır, ve nesne yürütmesi gerçekte nesneyi yürüten kod ve veridir.

Şekil 3, başlı başına bir Object Request Broker (ORB)'ın yapısını gösterir. ORB'ye doğru olan arayüzler kutular ile gösterilmiştir, ve oklar ya ORB'nin çağrıldığını yada arayüz üzerinden bir üst-çağrı yapıldığını gösterir.

Bir çağrı yapmak için İstemci, Dinamik Çağrı Arayüzü'nü (DII – Dynamic Invocation Interface – hedef nesnenin arayüzünden bağımsız olan aynı arayüz) veya bir OMG IDL stub'ını (hedef nesnenin arayüzüne bağlı olan özel stub) kullanabilir. İstemci bazı fonksiyonlar için ORB ile direkt olarak temas ta kurabilir (OMG, 2001).

Nesne Yürütmesi bir çağrıyı, OMG IDL tarafından üretilen iskelet üzerinden veya bir dinamik iskelet üzerinden, bir üst-çağrı olarak alır. Nesne Yürütmesi bir isteği işlerken veya başka durumlarda, Nesne Adaptörünü ve ORB'yi çağırabilir.

Arayüzlerin nesnelere tanımlanması iki yolla yapılabilir. Arayüzler, arayüz tanımlama dilinde (IDL) statik olarak tarif edilebilir. Bu dil, nesnelerin tiplerini, üzerlerinde yapılabilecek işlemlere, ve bu işlemlerin alabileceği parametrelere göre tanımlar. Alternatif olarak, arayüzler aynı zamanda bir Arayüz Ambarı (Interface Repository) servisine eklenebilir; bu servis bir arayüzün bileşenlerini, bu bileşenlere çalışma zamanı (run-time) erişime izin vererek, nesneler olarak gösterir. Her ORB yürütmesinde, Arayüz Tanımlama Dili ve Arayüz Ambarı eşit güçtedirler.

İstemci, nesnenin tipini ve yapılması gereken işlemi bilerek, ve nesne için bir Nesne Referansına erişerek, bir çağrı yapar. İstemci çağrıyı, nesneye özel stub rutinlerini çağırarak, veya çağrıyı dinamik olarak yapılandırarak, başlatır (Şekil 4).

Bir çağrıyı başlatmak için dinamik ve stub arayüzü aynı çağrı semantiklerini yerine getirirler, ve mesajın alıcısı çağrının nasıl başlatıldığını bilemez.

Bir CORBA nesnesi ORB ile, bir ORB arayüzü veya bir nesne adaptörü vasıtasıyla etkileşime girer (RAJ, 1998). ORB uygun yürütme kodunu yerleştirir, parametreleri aktarır, ve IDL iskeleti veya dinamik iskelet üzerinden Nesne Yürütmesine kontrolü geçirir (Şekil 5). İskeletler arayüze ve nesne adaptörüne özeldir. Çağrıyı gerçekleştirirken, nesne yürütmesi, Nesne Adaptörü vasıtasıyla ORB'den bazı servisleri bulabilir. Çağrı tamamlandığında, kontrol ve çıktı değerleri istemciye geri döner.

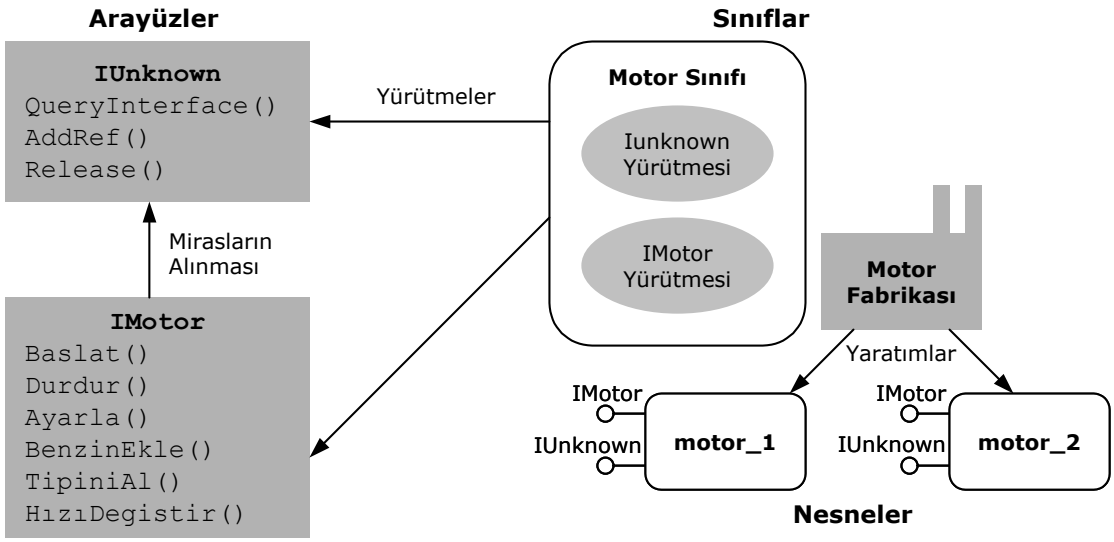
Nesne Yürütmesi hangi Nesne Adapörünün kullanılacağını seçebilir. Bu karar, Nesne Yürütmesinin hangi tipte servislere ihtiyaç duyduğuna bağlıdır (OMG, 2001).

Şekil 6, arayüz ve yürütme bilgisinin, istemciler ve nesne yürütmeleri üzerinde, nasıl hazır hale getirilebileceğini gösterir. Arayüz OMG IDL'de ve/veya Arayüz Ambarında tanımlanır; tanımlama istemci stub'larını ve nesne yürütme iskeletlerini oluşturmak için kullanılır.

Nesne yürütme bilgisi yükleme sırasında sağlanır, ve çağrı teslimatı sırasında kullanılmak üzere Yürütme Ambarında saklanır.

DCOM Hakkında Genel Bilgi

DCOM, OLE'nin ve ActiveX'in altyapısını oluşturan COM (Component Object Model)'dan geliştirilerek oluşturulmuştur. COM, yazılım bileşenlerini geliştirmek ve yaymak amacıyla kullanılan nesne tabanlı bir çatıdır. COM, geliştiricilerin soyutlamaları bileşen arayüzleri olarak elde etmelerine izin verir, ve bu arayüzleri yürüten sınıfları sağlar. İstemci uygulamaları sadece bir nesnenin arayüzünde tanımlı olan fonksiyonları çağırabilirler (encapsulation).



Şekil 7. Arayüzler, Sınıflar ve Nesneler

COM'un ikili birlikte çalışabilirlik (binary interoperability) standardı, yazılım bileşenlerinin geliştirilmesini ve bu bileşenlerin ikili (binary) biçimde yayılmasını sağlar. Sonuç olarak bağımsız yazılım geliştiriciler, tekrar kullanılabilen yapı bloklarını geliştirebilir ve kaynak kodunu taşımadan paketleyebilirler.

DCOM (Distributed COM), uzak yöntem çağrıları, güvenilirlik, ölçeklenebilirlik ve yer şeffaflığı sağlayarak, COM'un ağ üzerinde çalışmasını sağlamıştır.

COM ve DCOM mimarilerinin temelinde Arayüzler, Sınıflar ve Nesneler vardır (Şekil 7).

Arayüzler

Bir COM arayüzü, bir yazılım bileşeninin davranışlarını ve yeteneklerini, bir yöntemler ve özellikler kümesi olarak tanımlar. Bir arayüz, onu destekleyen nesnelerden anlamsal olarak tutarlı bilgileri alacağını garanti altına alan bir anlaşmadır. Her COM nesnesi birden çok arayüzü aynı anda destekleyebilir. Nesneler en az bir arayüzü (IUnknown arayüzü) mutlaka desteklemelidirler (ROY & EWALD, 1997).

Bileşen tasarımcıları, arayüzleri, DCE RPC IDL'nin nesneye-yönelik olacak şekilde geliştirilmiş hali olan MIDL (Microsoft's Interface Definition Language) kullanarak tanımlarlar. Microsoft, bir arayüz tanımından C veya C++ dillerinde proxy ve stub kodu oluşturan bir MIDL derleyicisi sağlar. Oluşturulan proxy kodu, arayüzü destekleyen nesneler için, bir istemci tarafı uygulama programlama arayüzü (API – Application Programming Interface) sağlar. Stub nesneleri gelen istemci isteklerinin şifresini çözer (decode) ve sunucudaki ilgili nesneye bunları iletirler. Arka planda, istekleri ve cevapları değerlendirmek için proxy ve stub kodları ilgili çalışma zamanı kütüphanelerle etkileşime girerler. COM çalışma zamanı yazılımı, nesneler istemci ile aynı işlemde bulunuyorsa, bu fazladan işi yapmayacak kadar akıllıdır.

Bileşen tasarımcıları, her arayüze, her zaman ve her yerde tek olan bir belirleyici (UUID – Universally Unique Identifier) atayarak, isim çakışmalarından doğabilecek belirsizlikleri ortadan kaldırırlar. Arayüz belirleyicisi (IID – Interface Identifier) olarak adlandırılan bu belirleyici, aynı zamanda COM'un arayüz uyarlama modelinin temel taşıdır.

Her COM nesnesinin en azından IUnknown standart arayüzünü desteklemesi gerektiğini söylemiştik. IUnknown, nesne yaşam döngüsünün yönetilmesi için temel yapı bloklarını sağlayan, ve bir nesne tarafından desteklenen arayüzlerin gelişimine izin veren yöntemleri tanımlar.

IUnknown arayüzünün QueryInterface yöntemi, bir IID olarak tanımlanan belirli bir arayüzün bir nesne tarafından desteklenip desteklemediğini belirlemek için istemciler tarafından kullanılır. Zamanla, bir nesne yeni arayüzleri veya her birinin farklı IID'si olan aynı mantıksal arayüzün yeni sürümlerini destekleyebilir. Varolan istemciler, yeniden derlenmeden bir arayüzün daha eski bir sürümünü kullanmaya devam edebilirler, ve yeni istemciler arayüzün son sürümünü sorgulayabilir ve bu sayede eklenen yeniliklerden faydalanabilirler.

QueryInterface, arayüz işaretçisi (interface pointer) denilen bir işaretçi döndürür. Arkaplanda, bir arayüz işaretçisi COM'un ikili birlikte çalışabilirlik standardı

tarafından dikte ettirilen bir veri yapısına işaret eder. Standart, istemci ve sunucu programlarının yürütme bileşenleri arasındaki farkları önemsemeden, çağrılması gereken yön arayüz fonksiyonlarını dikte eder.

Sınıflar ve Sunucular

Arayüzler kendi başlarına COM uygulamaları oluşturmak için yeterli değildirler. Bir COM sınıfı, bir yada daha çok COM arayüzünün bir yürütmesi olan bir kaynak kodunun gövdesidir. Desteklediği her arayüz yöntemi için, desteklenen herhangi bir programlama dilinde, gerçek fonksiyonlar sağlar.

Her arayüz bir IID tarafından tekil olarak tanımlandığından, her COM sınıfı CLSID adı verilen bir tekil belirleyici ihtiva eder. Bir istemci uygulamanın, bir bileşenle etkileşime girmesi için, en az bir CLSID ve bu sınıf tarafından desteklenen arayüz için bir IID hakkında bilgi sahibi olması veya öğrenebilecek konumda olması gerekmektedir. Bir istemci bu bilgiyi kullanarak, bir nesne üretmek ve bir ilgili arayüz işaretçisi döndürmek için COM'dan izin ister (ROY & EWALD, 1997).

Bir yada daha çok COM sınıfı, kullanılabilecek çeşitli tekniklerden biri kullanılarak bir sunucunun içine yerleştirilir. Sunucunun içindeki bir sınıfa bir istemci tarafından ilk kez erişildiğinde, bir COM sunucusu istemci işleminde yüklenen bir dinamik bağlantı kütüphanesi (DLL – dynamic link library) olarak paketlenebilir. Buna, işlem içi sunucu (in-process server) denir. ActiveX kontrolü, bir işlem içi COM sunucusudur. Bir COM sunucusu ayrı ayrı çalışabilecek şekilde de paketlenebilir. Bu tip bir sunucu, bir istemci ile aynı makinede veya DCOM kullanılarak erişilebilen uzak bir makinede yürütülebilir. Bu sunuculara, işlem dışı sunucular (out-of-process server) denir. Aynı istemci kodu, farklı COM sunucu tipleri ile etkileşebilir.

Nesneler

Her COM nesnesi bir COM sınıfının bir örneğidir. Her COM sınıfı, görevi bir COM sınıfının örneklerini oluşturmak olan, sınıf fabrikası denilen başka bir COM sınıfı ile ilişkilidir. Fabrika tipik olarak, COM tarafından tanımlanmış standart bir arayüz olan IClassFactory'i destekler.

İstemci uygulaması yeni bir nesne yaratma isteği gönderdiğinde, COM, sunucuyu bulur, yükler yada başlatır, ve nesneyi yaratabilmek için fabrikaya danışır. COM bir arayüz işaretçisini istemciye döndürür, ve sonraki çağrılar direkt olarak COM nesnesine gider.

Bir istemci uygulaması, bir COM sınıfı için, sınıfın bir örneğini oluşturmak üzere, standart bir biçimde fabrika ile etkileşime girer, ve oluşan COM nesnesine bir arayüz işaretçisi bulur (ROY & EWALD, 1997). Şekil-7'de görüldüğü gibi, bir COM arayüzü, ilgili yöntemler kümesini tanımlar, ve tekil bir arayüz belirleyicisine atanır. Bir COM sınıfı, bir yada daha çok arayüzü yürütür, ve tekil bir sınıf belirleyicisine atanır. Her COM sınıfı, COM nesneleri yaratmaktan sorumlu olan bir sınıf fabrikasına sahiptir.

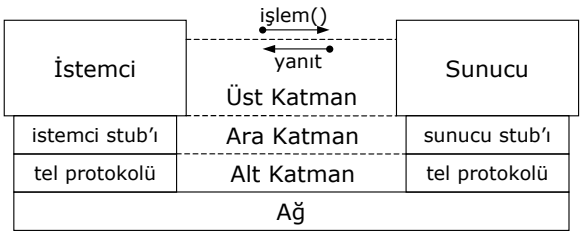
Bir COM nesnesi oluşturulduğunda, COM bir nesneye doğru ne kadar önemli referans bulunduğunu, ve bu nesnenin ne zaman yok edilebileceğini takip etmek için standart bir protokol belirler. COM nesneleri, IUnknown arayüzünde tanımlanan AddRef ve Release yöntemleri ile işletilen bir referans sayısı muhafaza eder. Her COM sınıfı, örneklerinin varolan kullanımlarının sayısını takip etmek için bu yöntemleri yürütmelidir. Geliştiricinin tercihine bağlı olarak bu referans takibi, nesnenin tümü için tek bir sayı ile yapılabileceği gibi, nesnenin desteklediği her arayüz için ayrı sayılar ile de yapılabilir.

Referans sayısı sıfır olduğunda, bu nesneye işaret eden herhangi bir istemci kalmadığı, ve bu yüzden artık yok edilebileceği kabul edilir. QueryInterface'in de aralarında bulunduğu arayüz işaretçileri döndüren yöntemler, bir nesneye yeni bir referans belirtmek için AddRef'i çağırmalıdırlar. Arayüz işaretçileri kullanan istemciler, arayüz işaretçisine erişimlerini tamamladıklarında Release'i çağırmalıdırlar.

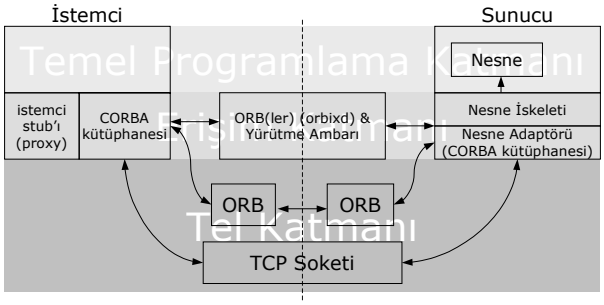
CORBA ve DCOM Arasındaki Benzerlik ve Farklılıklar

Bir COM sunucusu, birden çok nesne sınıfının nesne örneklerini yaratabilir. Bir COM nesnesi, her biri nesnenin farklı bir görünümünü veya davranışını temsil eden birden çok arayüz destekleyebilir. Bir arayüz, fonksiyonellikle ilgili bir yöntemler kümesinden ibarettir. Eğer nesne istemcinin adres uzayında yer alıyorsa, bir COM istemcisi bir COM nesnesi ile, o nesnenin arayüzlerine doğru olan bir işaretçiyi elde ederek etkileşime girer, ve bu işaretçi üzerinden yöntemleri başlatır. COM, C++ sanal fonksiyon tablosunda olduğu gibi, herhangi bir arayüzün standart hafıza planını takip etmesini zorunlu kılar. Tanımlama ikili seviyede olduğundan, C++, Java ve Visual Basic gibi faklı dillerde yazılması muhtemel ikili bileşenlerin entegrasyonuna izin verir (MICROSOFT, 1998).

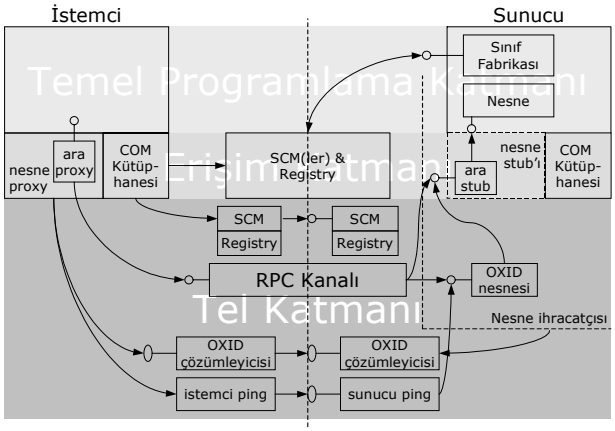
Bir CORBA nesnesi, dış dünyaya bir yöntemler kümesi ile bir arayüz vasıtasıyla anlatılır. Bir nesnenin belirli bir örneği, bir nesne referansı ile tayin edilir. Eğer nesne istemcinin adres uzayında yer alıyorsa, bir CORBA nesnesinin istemcisi, kendi nesne referansını



Şekil-8 RPC Yapısı



Şekil 9. CORBA Mimarisi



Şekil 10. DCOM Mimarisi

elde eder ve bunu yöntem çağrıları yapmak için kullanır. Nesnenin yürütmesinin bulunması, isteği alması için hazırlanması, isteğin onunla haberleşmesi, ve eğer varsa yanıtın istemciye taşınması için gerekli tüm mekanizmalardan ORB sorumludur. Nesne yürütmesi, ya bir Nesne Adaptörü, yada ORB arayüzü vasıtasıyla ORB ile etkileşime girer.

Hem DCOM hem de CORBA, istemci-sunucu tipi haberleşmeleri sağlarlar. Bir istemci, bir servise istek yapmak için, istemci-sunucu modelinde sunucu olarak rol oynayan bir uzak nesne tarafından yürütülen bir yöntemi başlatır. Sunucu tarafından sağlanan servis bir nesne olarak korunur ve bir nesnenin arayüzü bir IDL'de tanımlanır. IDL dosyasında tanımlanan arayüzler, bir sunucu ile istemcileri arasında bir anlaşma vazifesi görür. İstemciler IDL'de tanımlanan yöntemleri başlatarak bir sunucu ile etkileşime girerler. Gerçek nesne yürütmesi istemciden saklanır. Veri korunması, polymorphism ve tekli miras gibi bazı nesneye yönelik programlama özellikleri IDL düzeyinde vardır. CORBA, IDL düzeyinde çoklu miras özelliğini de destekler, ama DCOM desteklemez. Bunun yerine, DCOM aynı amaç için birden çok arayüze sahip nesne kavramını kullanır. CORBA IDL'de istisnai hatalar da tanımlıdır (exceptions).

Hem DCOM'da hem de CORBA'da, bir istemci işlemi ile bir nesne sunucusu arasındaki etkileşimler nesneye yönelik RPC tipi haberleşmelerle yürütülür. Şekil-8, tipik bir RPC yapısını gösterir. İstemci, uzak fonksiyonu başlatmak için, istemci stub'a bir çağrı yapar. Stub, çağrı parametrelerini bir istek mesajı içine paketler, ve mesajı sunucuya iletmek için bir tel protokolü başlatır. Sunucu tarafında, tel protokolü mesajı sunucu stub'ına iletir. Sunucu stub'ı mesajın paketini açar ve nesnedeki gerçek fonksiyonu çağırır. DCOM'da istem-ci stub'ına proxy, sunucu stub'ı da stub denir. CORBA'da ise istemci stub'ına stub, sunucu stub'ına iskelet denir. Bazen, CORBA'daki stub'ın çalışmakta olan bir örneğine proxy denir.

CORBA ve DCOM mimarilerinin ayrıntılı yapıları sırasıyla Şekil 9 ve Şekil 10'da gösterilmiştir. Üst katman, istemci ve nesne sunucusu programlarının geliştiricilerinin gördüğü temel programlama mimarisi'dir. Ara katman, arayüz işaretçilerinin ve nesne referanslarının farklı işlemler üzerinde anlamlı olmasını sağlayan erişim mimarisi'dir. Alt katman ise, erişim mimarisini farklı makineler üzerinde çalışabilecek şekilde genişleten tel protokol mimarisi'dir.

Sonuçlar

DCOM ve CORBA mimarileri temel olarak benzerdirler. Her ikisi de şeffaf aktivasyonlar ve uzak nesne erişimi için dağıtık nesne altyapısı sağlar. Aralarındaki temel farklar şunlardır:

Tablo 1. Birbirinin yerini tutan terimlerin ve varlıkların özeti

	DCOM	CORBA
Üst katman: Temel Programlama Mimarisi		
Temel sınıf	IUnknown	CORBA::Object
Nesne sınıf belirleyicisi	CLSID	arayüz ismi
Arayüz belirleyicisi	IID	arayüz ismi
İstemci tarafı nesne aktivasyonu	CoCreateInstance()	bir yöntem çağrısı/bind()
Nesne idaresi	arayüz işaretçisi	nesne referansı
Ara katman: Erişim Mimarisi		
Yürütme eşlemesinin adı	Registry	Yürütme Ambarı
Yöntemler için tip bilgisi	Tip Kütüphanesi	Arayüz Ambarı
Yerleştirme yürütmesi	SCM	ORB
Aktif etme yürütmesi	SCM	OA
İstemci tarafındaki stub	proxy	stub/proxy
Sunucu tarafındaki stub	stub	iskelet (skeleton)
Alt katman: Tel Protokol Mimarisi		
Sunucu son nokta çözümleyicisi	OXID çözümleyicisi	ORB
Sunucu son noktası	nesne ihracatçısı	OA
Nesne Referansı	OBJREF	IOR (veya nesne referansı)
Nesne Referans oluşumu	nesne ihracatçısı	OA
Veri biçimini dönüştürme	NDR	CDR
Arayüz örneği belirleyicisi	IPID	object_key

1. DCOM birden çok arayüzlü nesneleri destekler ve arayüzler arasında gezinmek için QueryInterface() yöntemini sağlar. Bu aynı zamanda, bir nesne proxy'sinin (stub) erişim katmanında birden çok proxy'i dinamik olarak yüklemesi fikrini ortaya çıkarır. CORBA'da bunun yerine çoklu miras alma özelliği vardır.

2. Her CORBA arayüzü, nesne kaydı, nesne referans oluşumu, iskelet örneklendirilmesi, gibi genel görevleri açık bir biçimde yerine getiren constructor olan CORBA::Object'ten miras alır. DCOM'da, bu gibi görevler ya sunucu programları tarafından üstü kapalı bir şekilde yerine getirilir, ya da DCOM çalışma zamanı sistemi tarafından dinamik olarak elde alınır.

3. DCOM'un tel protokolü RPC'ye sıkı bir şekilde bağlanmıştır. CORBA bu konuda serbesttir.

4. DCOM tanımlaması yürütme konusu olarak düşünülen ve CORBA tarafından belirlenmeyen bir çok detayı içerir.

Tartışma

Farklı platformlar ve farklı programlama dilleri ile birbirlerinden bağımsız olarak tasarlanmış yazılım bileşenlerinin, birlikte çalışabilmelerine olanak sağlayan dağıtık nesne yönetimi mimarileri sayesinde yeniden kullanılabilir yazılımlar geliştirmek daha hesaplı olacak ve daha geniş platforma sahip olacaktır.

En çok kullanılan iki dağıtık nesne yönetimi mimarisi olan DCOM ve CORBA mimarilerinin geleceğine baktığımızda; COM ve DCOM'un .NET altında çalışmaya devam edeceğini, ve CORBA'nın da MDA altında varlığını sürdüreceğini söyleyebiliriz. Diğer dağıtık nesne yönetimi mimarilerinin, şu an için bu iki mimari ile rekabet edebilecek seviyelerde olmadıklarını görmekteyiz.

Kaynaklar

1 MICROSOFT, "Microsoft Developer Network", http://msdn.microsoft.com

2 MICROSOFT, "DCOM Architecture White Paper", Microsoft, 1998

3 OMG, "The Common Object Request Broker: Architecture and Specification", OMG, Editorial Revision: CORBA 2.4.2, http://www.omg.org, 2001

4 RAJ GS, "A Detailed Comparison of CORBA, DCOM and Java/RMI", Object Management Group (OMG) whitepaper, 1998

5 ROSENBERGER JL, "Teach Yourself CORBA In 14 Days", SAMS Publishing, 1998

6 ROY M, EWALD A, "Inside DCOM", http://www.dbmsmag.com/9704d13.html, 1997

7 VINOSKI S, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", IEEE Communications Magazine Vol. 35, No. 2, February 1997

8 VINOSKI S, "Distributed Object Computing With CORBA", C++ Report Magazine, July/August 1993