İSTANBUL TEKNİK ÜNİVERSİTESİ

CORBA İLE DAĞITILMIŞ NESNE SİSTEMİ UYGULAMASI

Bitirme Çalışması

Erdem Salihoğlu

Fakülte: Elektrik Elektronik

Bölüm: Bilgisayar Mühendisliği

Danışman: Yrd. Doç. Dr. Feza Buzluca

Temmuz 2003

ÖNSÖZ

Çalışmam sırasında bana yol gösteren danışmanım Yrd. Doç. Dr. Feza Buzluca'ya, en zorda kaldığım durumlarda yardımlarını esirgemeyen hocam Tacettin Ayar'a ve de aileme teşekkür ederim.

İÇİNDEKİLER

KISALTMALAR

API	**Application Program Interface**
COM	**Component Object Model**
CORBA	**Common Object Request Broker Architecture**
DCE RPC	**Distributed Computing Environment Remote Procedure Call**
DCOM	**Distributed Component Object Model**
DII	**Dinamic Invocation Interface**
DSI	**Dinamic Skeleton Interface**
GIOP	**General Inter-ORB Protocol**
IDL	**Interface Definition Language**
IIOP	**Internet Inter-ORB Protocol**
MSRPC	**Microsoft Remote Procedure Call**
OMG	**Object Management Group**
ONC RPC	**Open Network Computing Remote Procedure Call**
ORB	**Object Request Broker**
ORPC	**Object Remote Procedure Call**
POA	**Portable Object Adapter**
RPC	**Remote Procedure Call**
TCP/IP	**Transmission Control Protocol / Internet Protocol**
UDP	**User Datagram Protocol**
UİÇ	**Uzak İşlev Çağrımı**
XDR	**External Data Representation**

ŞEKİL LİSTESİ

ÖZET

Bu raporda dağıtılmış nesne sistemleri üzerine yapılan çalışmanın tanıtılması amaçlanmıştır. Öncelikle dağıtılmış nesne sistemleri hakkında önbilgi vermek için uzak işlev çağrımı ve kullanılan arakatman CORBA hakkında bilgi verilmiştir. Bu kuramsal bilgilerden sonra gerçeklenen çalışma tanıtılmıştır. Çalışmada bir dağıtılmış nesne sistemi uygulaması CORBA arakatmanı kullanılarak gerçekleştirilmiştir. Bunu yaparken nesneye dayalı programlamanın en önemli temel kavramları olan encapsulation, polymorphism ve de kalıtım da programlarda kullanılmıştır. Bunun yanında CORBA'nın iki temel özelliği olan ortamdan ve dilden bağımsızlık özelliklerini de destekleyen bir çalışma yapmak için istemci ve sunucu farklı işletim sistemleri üzerinde farklı nesne istek aracılarıyla ve de farklı diller kullanılarak gerçeklenmiştir. En son olarak da elde edilen sonuçlar ve de karşılaşılan zorluklar anlatılmıştır.

SUMMARY

This graduation project begins with an overview of remote procedure call paradigm and middleware.

One of the earliest facilities that was created to help programmers write client-server software is known generically as a Remote Procedure Call (RPC) mechanism. When programmers build a large program, they begin by dividing the program into major pieces. The most widely available programming language feature used for such division is the procedure. If a programmer follows the same procedure call paradigm used to build conventional programs when building client and server software, the programmer will find the task easier and will make fewer mistakes.

RPC extends the procedure call mechanism to allow a procedure in the client to call a procedure across the network in the server. That is, when such a procedure call occurs, the thread of control appears to pass across the network from the client to the server; when the called procedure returns, the thread of control appears to pass back from the server to the client

Writing client and server programs using RPCs involves many low-level details and because these programs follow a few basic architectural form, programmers can use tools to generate much of the code automatically. Tools make the resulting programs more efficient and correct.

A variety of commercial tools have been developed to help programmers construct client-server software. Such tools are generally called middleware because they provide software that fits between a conventional application and the network software.

In the 1990s, programming language research shifted focus from the procedure paradigm to the object-oriented paradigm. As a result, most new programming languages are object-oriented languages. The basic structuring facility in an object-oriented language is called an object, which consists of data items plus a set of operations for those items, which are known as methods. The basic control mechanism in an object-oriented language is method invocation. In response to the change in languages, designers are creating new middleware systems that extend method invocation across computers in the same way that remote procedure call extended procedure call. Such systems are known as distributed object systems.

Perhaps the best known object-oriented middleware is named Common Object Request Broker Architecture (CORBA).CORBA permits an entire object to be placed on a server, and extends method invocation using the same general approach as described above. One differece arises because proxies are instantiated at run-time like other objects. When a program receives a reference to a remote object, a local Proxy is created that corresponds to the object. When the program invokes a method on the object, control passes to the local proxy. The proxy then sends a message across the network to the server, which invokes the specified method and returns the results. Thus, CORBA makes method invocation for remote and local objects appear identical [3].

In this project a distributed object system application which uses CORBA as the middleware is implemented. In the project the most important concepts of object-oriented programming

(encapsulation, inheritance and polymorphism) are used. Moreover in the project the client is implemented on Linux Red Hat 8.0 with the Java Idl by using Java language, the server on Windows XP with omniORB by using C++, to test the CORBA's two most important features ; language independence and platform independence.

1. GİRİŞ

Programcıların istemci-sunucu yazılımları yazmalarına yardım etmek için yaratılmış ilk hizmetlerden biri Uzak İşlev Çağrımı (UİÇ) olarak bilinir. Programcılar büyük programlar yazarlarken önce programı büyük parçalara bölerler. Böyle bir bölme işlemi için en yaygın programlama dili özelliği işlevdir. Programcı, bu büyük parçaların her birini bir işlev ile ifade eder. Eğer yazılımcılar istemci-sunucu yazılımları yazarlarken geleneksel programları yazarlarken kullandıkları işlev çağrısı kavramını kullanırlarsa işleri kolaylaşacaktır ve daha az hata yapacaklardır.

UİÇ; istemcideki bir işlevin, ağ üzerinden sunucudaki bir işlevi çağırmasına izin vererek işlev çağrısı mekanizmasını genişletir. Böyle bir işlev çağrısı olduğunda programın kontrolü ağ üzerinden istemciden sunucuya geçer, çağrılan işlev geri döndüğünde kontrol sunucudan istemciye geçer.

UİÇ kullanarak istemci ve sunucu programları yazmak birçok alt seviye ayrıntı içerdiğinden ve de istemci-sunucu yazılımları birkaç temel mimariyi izlediğinden, programcılar kodun çoğunu otomatik olarak üretmek için araçlar kullanabilirler. Bu araçlar programları daha verimli ve hatasız yaparlar.

Programcıların istemci – sunucu yazılımları yapmalarına yardım etmek için yukarıda anlatılan kavramı kullanan çeşitli ticari araçlar geliştirildi. Bu araçlar geleneksel uygulama programıyla ağ yazılımı arasında oldukları için arakatman diye adlandırılırlar.

1990'larda, programlama dili araştırmaları işlev kavramından çok nesne kavramında yoğunlaşmaya başladılar. Bunun sonucu olarak yeni programlama dillerinin çoğu nesneye dayalı diller oldular. Nesneye dayalı dillerde temel yapısal birim nesnedir. Nesneler, veriler ve bu veriler üzerinde işlem yapan metod diye adlandırılan fonksiyonlardan oluşur. Nesne yönelimli programlamada temel kontrol mekanizması metod çağrımıdır. Dillerdeki bu değişime cevap olarak yazılımcılar, uzak işlev çağrımının işlev çağrımını genişlettiği gibi bilgisayarlar arasında metod çağrısını genişleten yeni arakatmanlar yaratıyorlar. Bu sistemlere dağıtılmış nesne sistemleri denir.

Belki de en iyi bilinen arakatman Common Object Request Broker Architecture (CORBA)'dır. CORBA bütün bir nesnenin bir sunucuya yerleştirilmesine izin verir ve yukarıda anlatılan aynı genel yaklaşımı kullanılarak metod çağrımı kavramını genişletir. CORBA'da vekiller diğer nesneler gibi çalışma zamanı yaratıldıkları için bir fark ortaya çıkar. Bir program uzak bir nesnenin referansını elde ettiğinde, bu nesneye karşılık gelen vekil yaratılır. Program bu nesnedeki bir metodu çağırdığında kontrol yerel vekile geçer. Vekil ağ üzerinden sunucuya bir ileti gönderir. Sunucu da belirtilen metodu çağırır ve geri döndürür. Bu şekilde CORBA metod çağrımını yerel ve uzak nesneler için aynı gibi gösterir. [3]

Yapılan uygulamada arakatman olarak burada kısaca bahsedilen CORBA kullanılmıştır. Raporun 2. kısmında UİÇ kavramı ve de CORBA arakatmanı daha ayrıntılı tanıtılmıştır. 3. bölümde ise gerçeklenmiş olan çalışma anlatılmıştır.Yapılmış olan çalışmanın kodları ve de programların akış şemaları 4. bölümde bulunmaktadır. Son olarak da raporun 5. bölümünde çıkarılan sonuçlar ve de karşılaşılan zorluklar anlatılmıştır.

2. KURAMSAL BİLGİLER

2.1 UZAK İŞLEV ÇAĞRIMI (REMOTE PROCEDURE CALL) VE ARA KATMAN (MIDDLEWARE)

2.1.1 İstemci ve Sunucu Programları Yazma

İstemci-sunucu yazılımı yapmış olan programcılar, bunun zor olduğunu bilirler. İstemci-sunucu yazılımı yapan programcılar, alışılmış görevlere ek olarak iletişimin karmaşık konularıyla da uğraşmak zorundadırlar. İhtiyaç duyulan bir çok fonksiyon, soket arayüzü gibi bir standart API tarafından sağlansa dahi programcılar bir çok zorlukla karşılaşırlar. Soket çağrıları; programcının isimler, adresler, protokoller ve portlar gibi bir çok alt seviye ayrıntılarla uğraşmasını gerektirir. Daha da önemlisi, koddaki, program test edilirken kolaylıkla gözden kaçabilecek küçük hatalar, program çalışırken büyük problemlere sebep olabilirler.

İstemci-sunucu yazılımı yazan programcılar, bu yazılımların genellikle aynı genel yapıyı izlediklerini ve aynı ayrıntıların çoğunu tekrar ettiklerini bilirler. Çoğu istemci-sunucu yazılım sistemi, birkaç temel mimari modelini izler. Ayrıca gerçeklemeler aynı standart APİ'yı kullanma eğiliminde olduklarından (örneğin soket ara yüzü) bir programdaki ayrıntılı kodun çoğu diğerlerinde tekrarlanır. Örneğin bağlantılı iletişim kullanan tüm istemci programları bir soket yaratmalı, sunucunun adresini belirlemeli, sunucuyla bağlantı kurmalı, sunucuya istek göndermeli, sunucudan cevap almalı ve etkileşim bittiğinde bağlantıyı kapatmalıdır.

Aynı kodu tekrar tekrar yazmamak ve hem doğru yazılmış hem de kaliteli kod üretmek için programcılar istemci ve sunucu yazılımı yazarken otomotikleştirilmiş araçlar kullanırlar. Bu araçlar, sunulacak olan servisin üst seviye bir tanımını kabul ederler ve gerekli olan kodun çoğunu otomatik olarak üretirler. Bu araçlar tüm programlamayı ortadan kaldıramazlar ama iletişim ayrıntılarını ele alırlar. Sonuç olarak kod daha az hata içerir.

Özet olarak: istemci ve sunucu programları yazmak birçok alt seviye ayrıntı içerdiğinden ve de istemci-sunucu yazılımları birkaç temel mimariyi izlediğinden, programcılar kodun çoğunu otomatik olarak üretmek için araçlar kullanabilirler. Bu araçlar programları daha verimli ve hatasız yaparlar.

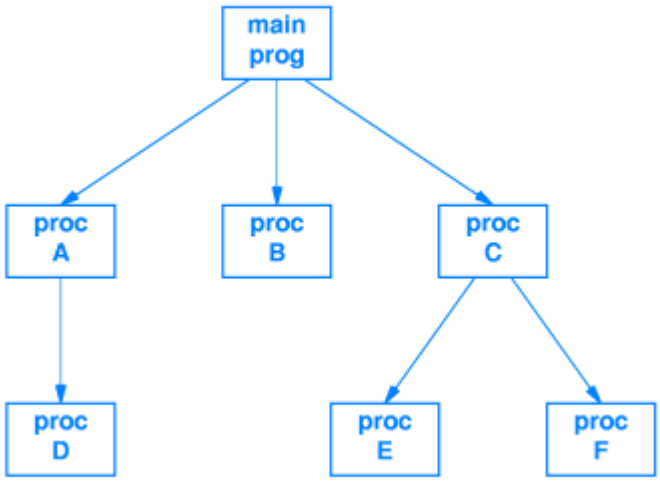
2.1.2 Uzak İşlev Çağrımı (UİÇ) Kavramı

Programcıların istemci-sunucu yazılımları yazmalarına yardım etmek için yaratılmış ilk hizmetlerden biri Uzak İşlev Çağrımı olarak bilinir. Programcılar büyük programlar yazarlarken önce programı büyük parçalara bölerler. Böyle bir bölme işlemi için en yaygın programlama dili özelliği işlevdir. Programcı, bu büyük parçaların her birini bir işlev ile ifade eder.

Programcı, programı gerçeklerken kodu daha kolay yönetebilmek için işlevleri kullanır. Programcı, birçok göreve sahip büyük bir işlev tanımlamaktansa görevleri kümelere ayırır ve her kümeyi ele almak için daha kısa işlevler kullanır. Eğer sonuçta oluşan

görevler önemli miktarda kod gerektiriyorsa, programcı görevleri de alt bölümlere ayırır ve her alt görevi gerçekleştirmek için alt işlevler kullanır. Bu şekilde bir programın tamamı işlev çağrıları hiyerarşisinden oluşur.

Bir programın işlevsel hiyerarşisi; her düğümün bir işlevi temsil ettiği ve X düğümünden Y düğümüne bir kenarın, X işlevinin Y işlevine bir çağrı içerdiğini ifade eden yönlendirilmiş bir grafla temsil edilebilir. Bu grafik gösterim, işlev çağrımı grafı olarak bilinir. Bakınız şekil 2.1.



Şekil 2.1: Örnek bir işlev çağrımı grafı. Bir işlevden diğerine bir ok, okun çıktığı işlevin okun vardığı işleve bir çağrı içerdiğini ifade eder.

Şekildeki kavramların Türkçe karşılıkları:
main prog: ana program
proc: işlev

Uzak çağrılar açısından, geleneksel işlev çağrılarının başka bir açısı olan parametreler önemlidir. Her işlev bir parametre kümesiyle bildirilir. Bu, işlevin birbiriyle bağlantılı görevleri çözebilmesini ve daha geniş bir problem kümesi için çözüm olmasını sağlar. İşlevi çağırırken; çağıran, işlevin bildirimindeki parametrelere karşılık gelen argümanları da işleve gönderir.

Bir programlama soyutlaması olan parametreli işlevler iyi çalıştıkları için araştırmacılar, istemci ve sunucu oluşturmak için işlevsel soyutlamayı denediler. Bir programcı, istemci-sunucu yazılımı yaparken aynı işlev çağrımı kavramını izlerse daha az hata yapacak ve programcının işi kolaylaştıracaktır.

Araştırmacılar bunu gerçekleştirmek için istemci-sunucu programlamayı olabildiğince geleneksel programlama gibi yapmak için yollar aradılar. Maalesef çoğu programlama dili tek bir bilgisayara kod üretmek için tasarlanmıştır. Diller de derleyiciler de dağıtılmış bir sistem için tasarlanmamıştır. Üretilen kodlar, kontrolün akışını ve parametre aktarımını tek bir adres uzayıyla sınırlarlar.

Bazı araştırmacılar, dillere istemci-sunucu yeteneklerini de kazandırabilmek için dillerde küçük değişiklikler yapmaya çalıştılar. Örneğin Modula-3 ve Java Uzaktan Metod Çağrımı (Java Remote Method Invocation (Java RMI)), programcıların diğer bilgisayarlardaki işlevleri çağırabilmelerine olanak sağlayan yeteneklere sahiptirler.

İstemci-sunucu programlama konusundaki araştırmalarda programlama dillerinde doğrudan değişiklik yapmaktan kaçınıldı. Bunun yerine programcıların dağıtılmış programlar yaparken kullanabilecekleri araçlar geliştirildi. Özellikle; yüksek seviye program girdilerini kabul eden, ağ iletişimi için gerekli kodu ekleyen ve sonuçta oluşan çıktıyı istemci-sunucu programları üretmek için otomatik olarak çeviren araçlar vardır.

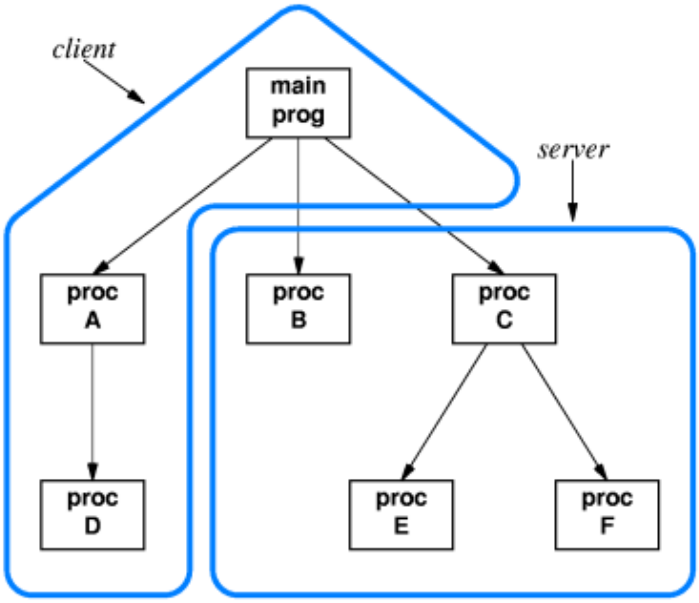
Özet olarak: geleneksel programlama dilleri, ağa bağlı bir bilgisayardaki bir programın ağdaki diğer bir bilgisayardaki bir programa işlev çağrısı yapmasına izin vermezler. Bu işlemi gerçekleştirebilmek için çeşitli araçlar kullanılır.

2.1.3 UİÇ (RPC) Kavramı

UİÇ kullanırken, programcı dikkatini bilgisayar ağında veya protokolünde yoğunlaştırmaz. Bunun yerine programcı, çözülmesi gereken problem hakkında düşünür. Programcı tek bir bilgisayarda çalışan geleneksel bir programla programın nasıl çözülebileceğini düşünerek başlar. Programcı, programı işlev çağrılarını kullanarak tasarlar ve oluşturur.

Programcı verilen problemi çözmek için geleneksel bir program oluşturduktan sonra, programı nasıl parçalara ayıracağını düşünür. Aslında programcı çağrı grafını iki parçaya ayırmalıdır. Programı iki parçaya ayırdıktan sonra asıl programı ve onun çağırdığı işlevleri içeren parça istemci, kalan işlevler sunucu olur. İstemciyi oluşturan bir küme işlevi ve sunucuyu oluşturan bir küme işlevi seçtikten sonra, programcı veriyi düşünmelidir. Her işlevin başvurduğu global veri aynı bilgisayara yerleştirilmelidir.

Şekil 2.2'deki bölme şekil 2.1'deki programın iki parçaya bölünmesine bir örnek teşkil eder.



Şekil 2.2: Şekil 2.1'deki örnek program istemci ve sunucuya bölünmüştür.

Şekildeki kavramların Türkçe karşılıkları client: istemci server: sunucu main prog: ana program proc: işlev

Şekil 2.2'teki istemci; ana program, A işlevi ve D işlevinden; sunucu ise B,C,E ve F işlevlerinden oluşmaktadır. Çağrı grafının diğer şekillerde bölünmesi de olasıdır.

UİÇ; istemcideki bir işlevin, ağ üzerinden sunucudaki bir işlevi çağırmasına izin verir. Böyle bir işlev çağrısı olduğunda programın kontrolü ağ üzerinden istemciden sunucuya geçer, çağrılan işlev geri döndüğünde kontrol sunucudan istemciye geçer.

Hangi işlevlerin sunucuya, hangilerinin istemciye yerleştirileceğine karar verdikten sonra, programcı bir UİÇ aracı kullanmaya hazırdır. Programcı, uzakta olacak olan işlevler kümesini ve bu işlevlerin parametrelerini tanımlayan bir bildiri oluşturur. (Sunucuyu oluşturacak olan işlevler.) Kullanılan araç da iletişim için gerekli yazılımı oluşturur.

2.1.4 İletişim Koçanları (Stubs)

Programın akışı bir bilgisayardaki bir programdan başka bir bilgisayardaki bir işleve doğrudan geçemez. Bir istemci uzak bir işlevi çağırdığında, ağ üzerinden sunucuya istek

iletisi göndermek için geleneksel protokolleri kullanır. İstek, çağırılacak olan işlevi belirtir. İstek gönderildikten sonra istemci taraftaki süreç bir cevap almak için bekler. Uzaktaki program (sunucu) istemciden bir istek aldığında belirtilmiş olan işlevi çağırır ve sonucu istemciye geri gönderir.

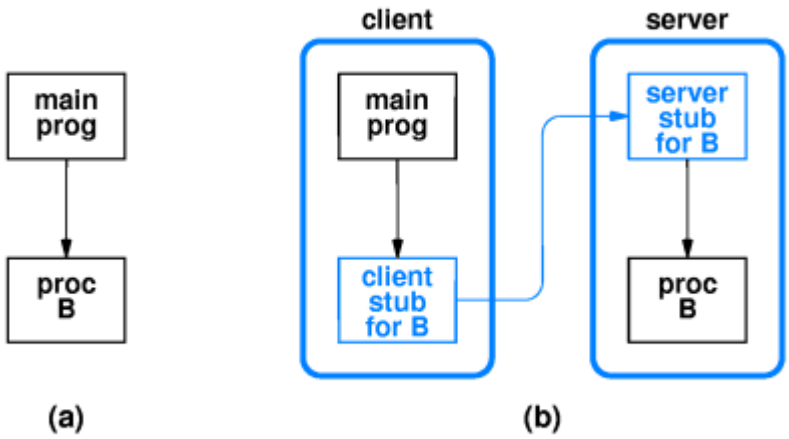
Etkileşimi sağlamak için programın iki tarafına da (istemci ve sunucu) ilave yazılım eklenmesi gerektiği açıktır. İstemci tarafındaki ilave yazılım, ağ üzerinden ileti göndermenin ve bir cevap beklemenin ayrıntılarını ele alır. Sunucu tarafındaki ilave yazılım ise gelen iletiyi almanın, belirlenmiş işlevi çağırmanın ve bir cevap göndermenin ayrıntılarını ele alır.

Teknik olarak, eklenen iki yazılım parçası da iletişim koçanı veya vekil (proxy) olarak bilinir. Biri istemcide biri de sunucuda olan iki koçan da bütün iletişim ayrıntılarını ele alır. Koçanlar yerleştirildikten sonra ana program tüm işlevler yerelmiş gibi işlev çağrılarını yapar. Yerel olmayan bir işleve çağrı yapıldığında ilgili iletişim koçanı işlev çağrısını yakalar, argümanlar için değerleri bir araya getirir ve ağ üzerinden sunucudaki iletişim koçanına ileti gönderir. Sunucu tarafındaki iletişim koçanı, belirlenmiş olan işlevi çağırmak için geleneksel işlev çağrı mekanizmasını kullanır ve sonuçları istemci koçana geri gönderir. İstemci koçan sonucu aldığında, sonucu kendisini çağırana (istemciye) aynen yerel bir işlev geri dönüyormuş gibi geri döner. Şekil 2.3 iletişim koçanlarının programın istemci ve sunucu taraflarının nasıl eklendiklerini göstermektedir.

Şekil 2.3a asıl programdaki işlev çağrısının UİÇ koçanları eklenmeden önceki halini gösteriyor. Ana program B işlevini çağırdığında, gönderdiği argümanlar B'nin parametreleriyle tamamen uyuşmalıdır. Yani çağrı doğru sayıda argüman içermeli ve bütün argümanların tipleri parametreler için bildirilenlerle uyuşmalıdır.

Şekil 2.3b program, istemci ve sunuculara ayrılınca programa eklenmesi gereken iletişim koçanlarını gösteriyor. b kısmındaki işlevsel arayüz a'daki asıl arayüzle aynı sayıda ve tipte argüman kullanmaktadır. Yani ana programdan istemci koçana ve sunucu koçandan B işlevine olan çağrı, ana programdan B işlevine olan geleneksel çağrıyla tamamen aynı arayüzü kullanır. Daha da önemlisi, istemci koçanlar yerlerini aldıkları koçanlarla aynı isimleri alabilirler. Sonuç olarak asıl program değiştirilmek zorunda değildir.

Özet olarak: uzak işlev çağrısını gerçeklemek için iki işlev arasına yerleştirilen iletişim koçanları, asıl işlev çağrısıyla aynı arayüz ve ismi kullanabileceğinden, çağıran işlev de çağrılan işlev da değiştirilmek zorunda değildir.



Şekil 2.3: a) Geleneksel programdaki asıl çağrı
b) Aynı çağrının iletişim koçanlarıyla gerçeklenmiş hali

Şekildeki kavramların Türkçe karşılıkları:
main prog: ana program
proc: işlev
client: istemci
server: sunucu
client stub for B: B için istemci koçan
server stub for B: B için sunucu koçan

2.1.5 Verilerin Dışarıdaki Temsili (External Data Representation)

Bilgisayar sistemlerinin tamamında veriler aynı şekilde temsil edilmezler. Örneğin bazı bilgisayarlar tam sayıların en anlamlı sekizlisini en küçük numaralı adreste saklarken bazıları da en az anlamlı sekizlisini en küçük numaralı adreste saklarlar. Bu yüzden istemci ve sunucular ayrı cinsten bilgisayarlar arasında tam sayı gönderdiklerinde, istemcinin mi sunucunun mu dönüşüm yapacağı ve de hangi temsilin ağ üzerinden gönderilirken kullanılacağı konularında anlaşmalıdırlar.

Verilerin dışarıdaki temsili terimi ağ üzerinden gönderilen verilerin biçimini ifade eder. Arayüz tanımlama diline ek olarak uzak işlev çağrımı teknolojisi verilerin dışarıdaki temsilini de tanımlar. Bazı sistemlerde istemci ve sunucu koçanlar, her birinin hangi temsili kullandığını öğrenmek ve de hangisinin dönüştüreceği konusunda anlaşmak için birbirlerine ileti gönderirler. Birçok sistem dış temsilin sabit olduğu başka bir yol izler. Bu sistemlerde, gönderen her zaman yerel temsilden dış temsile çevirir ve alan da her zaman dış temsilden yerel temsile çevirir.

Özetlersek: bilgisayar sitemlerinin hepsinin verileri aynı şekilde temsil etmemelerinden ve de istemci ve sunucuların iki tip bilgisayarda da çalışabilmesinden dolayı uzak işlev çağrımı teknolojisi bir veri temsilinden diğerine çevirme problemini ele almalıdır. Geniş bir çevre tarafından kabul edilen bir yöntem vardır. Bu yöntemde standart bir dış temsil belirlidir ve iki taraf da dış temsil–yerel temsil arasında çevirim yapar.

2.1.6 Arakatman ve Nesne Yönelimli Arakatman

Programcının UİÇ kullanan programlar yapmasına yardım eden araçlar vardır. Programcı arayüz ayrıntılarını vererek uzak olacak işlevleri belirler. Bunu yapmak için programcı arayüz tanımlama dili kullanır. Araçlar arayüz tanımlama girdisini okuyarak gerekli istemci ve sunucu koçanlarını oluştururlar. Programcı bundan sonra iki ayrı program derler ve bağlar. Sunucu koçanlar uzak işlevlerle birleştirilerek sunucu program oluşturulur. İstemci koçanlar da ana program ve yerel işlevlerle birleştirilerek istemci program oluşturulur.

Programcıların istemci – sunucu yazılımları yapmalarına yardım etmek için yukarıda anlatılan kavramı kullanan çeşitli ticari araçlar geliştirildi. Bu araçlar geleneksel uygulama programıyla ağ yazılımı arasında oldukları için arakatman diye adlandırılırlar.

1990'larda, programlama dili araştırmaları işlev kavramından çok nesne kavramında yoğunlaşmaya başladılar. Bunun sonucu olarak yeni programlama dillerinin çoğu nesneye dayalı diller oldular. Nesneye dayalı dillerde temel yapısal birim nesnedir. Nesneler, veriler ve bu veriler üzerinde işlem yapan metod diye adlandırılan fonksiyonlardan oluşur. Nesne yönelimli programlamada temel kontrol mekanizması metod çağrımıdır. Dillerdeki bu değişime cevap olarak yazılımcılar, uzak işlev çağrımının işlev çağrımını genişlettiği gibi bilgisayarlar arasında metod çağrısını genişleten yeni arakatmanlar yaratıyorlar. Bu sistemlere dağıtılmış nesne sistemleri denir.

2.1.7 Çok Kullanılan Arakatman ve Nesne Yönelimli Arakatman Teknolojileri

2.1.7.1 ONC RPC

Yaygın olarak kabul edilen ilk UİÇ mekanizmalarından biri Sun Microsystems, Incorporated tarafından tasarlandı. Open Network Computing Remote Procedure Call (ONC RPC) adındaki teknolojiye genelde Sun RPC denilir. Arayüz tanımlama dili, istemci ve sunucu koçanların haberleşmek için kullandıkları ileti biçiminin yanında ONC eXternal Data Representation (XDR) (Verilerin Dışarıdaki Temsili) olarak bilinen veri temsil standardını da içerir. ONC RPC, diğer çoğu UİÇ teknolojileri gibi TCP/IP protokollerini kullanır ve programcının iletişim sırasında UDP veya TCP protokollerinden birini seçmesine izin verir.

2.1.7.2 DCE RPC

Open Software Foundation, birlikte çalışmak için tasarlanmış birçok araç ve bileşenden oluşan Distributed Computing Environment (DCE)'yi tanımladı. DCE genellikle DCE/RPC adı verilen kendi uzak işlev çağrımı teknolojisini ve diğerlerinden ufak farklarla ayrılan kendi arayüz tanımlama dilini içerir. DCE/RPC, bir istemcinin birden

fazla sunucuya erişmesine izin verir. Yani bazı uzak işlevler bir sunucuya bazıları da başka bir sunucuya yerleştirilebilir. DCE/RPC TCP/IP protokollerini kullanır ve programcının iletişim sırasında UDP veya TCP protokollerinden birini seçmesine izin verir.

2.1.7.3 MSRPC

Microsoft Corporation kendi uzak işlev çağrımı teknolojisi olan Microsoft Remote Procedure Call'u tanımladı. MSRPC DCE/RPC'den türetildiğinden, DCE/RPC ile aynı merkezi kavramları ve genel program yapısını paylaşır. Bunun yanında MSRPC birkaç noktada DCE/RPC'den farklıdır. MSRPC kendi arayüz tanımlama dilini ve istemci ve sunucu koçanlarının haberleşirken kullanacakları kendi protokolünü tanımlamıştır.

2.1.7.4 CORBA

Belki de en iyi bilinen arakatman Common Object Request Broker Architecture (CORBA)'dır. CORBA bütün bir nesnenin bir sunucuya yerleştirilmesine izin verir ve yukarıda anlatılan aynı genel yaklaşımı kullanılarak metod çağrımı kavramını genişletir. CORBA'da vekiller diğer nesneler gibi çalışma zamanı yaratıldıkları için bir fark ortaya çıkar. Bir program uzak bir nesnenin referansını elde ettiğinde, bu nesneye karşılık gelen vekil yaratılır. Program bu nesnedeki bir metodu çağırdığında kontrol yerel vekile geçer. Vekil ağ üzerinden sunucuya bir ileti gönderir. Sunucu da belirtilen metodu çağırır ve geri döndürür. Bu şekilde CORBA metod çağrımını yerel ve uzak nesneler için aynı gibi gösterir.

İşlevler yerine nesnelere odaklanmasının yanında, CORBA geleneksel UİÇ teknolojilerinden daha dinamik olması yönüyle de ayrılır. Geleneksel UİÇ teknolojilerinde programcı programı oluştururken koçan işlevleri oluşturmak için bir araç kullanır. Fakat CORBA'da yazılım çalışma zamanında, ihtiyaç duyulduğunda bir vekil yaratır. Yani vekili olmayan uzak bir nesne için nesnenin metodu çağrıldığında vekil oluşturulur.

2.1.7.5 MSRPC2

Microsoft MSRPC2 adında MSRPC'nin ikinci bir sürümünü geliştirdi. Microsoft, ikinci sürüm basit bir gözden geçirmeden öte ciddi bir değişim gösterdiğinden dolayı, sık sık MSRPC2 yerine Object RPC (ORPC) ismini kullanır. Bu sürüm nesneler için daha fazla destek sağlamak için temel kavramlarını genişletti.

2.1.7.6 COM/DCOM

Microsoft Corporation, aynı zamanda nesne yönelimli teknolojiler olan COM ve DCOM'u da üretti. Component Object Model (COM) 1994'te tanımlanmış nesne yönelimli bir yazılım mimarisidir. Standart, verilen bir bilgisayardaki bileşenler (component) arasında ikilik sistemde arayüz tanımlar. Bu arayüze genelde paketleme düzeni (packaging scheme) denir. COM'da tüm nesnelere global, tekil bir isim verilir ve bütün nesne referansları global düzeni kullanır.

1998'de tanımlanan Distributed Component Object Model (DCOM) nesneye dayalı uzak işlev çağrıları için uygulama seviyesi bir protokol yaratarak COM'u genişletir. DCOM taşıma için ORPC'yi kullanır. ORPC, DCE/RPC'den türetildiğinden, DCOM, DC/RPC'nin uzak işlev çağrımı için kullandığı paket formatı ve semantiğin aynısını kullanır. [3]

2.2 CORBA (Common Object Request Broker Architecture) (Ortak Nesne İstek Aracısı Mimarisi)

2.2.1 Giriş

Bilgisayar ağları genel olarak heterojen bir yapıya sahiptir. Örneğin, küçük bir yazılım şirketinin yerel alan ağı birden fazla bilgisayar platformundan oluşuyor olabilir. Veri tabanı erişimini ele alan bir ana bilgisayar, donanım simülasyon ortamını ve yazılım geliştirme omurgasını sağlayan UNIX tabanlı bilgisayarlar, Windows işletim sistemine sahip masaüstü ofis araçlarını sağlayan kişisel bilgisayarlar ve ağ bilgisayarları, telefon sistemleri, yönlendiriciler ve ölçüm araçları gibi özelleşmiş bilgisayarlar bulunabilir. Bir ağın küçük birimleri homojen olabilir ama ağ ne kadar büyürse bileşimi o kadar çeşitli olur.

Bu heterojenliğin birkaç sebebi vardır. En önemli sebep teknolojinin zamanla değişmesidir. Ağlar kurulduktan sonra, değişen teknolojiyle parçalarını değiştirebildiklerinden farklı zaman dilimlerinin farklı teknolojileri ağ içinde bir araya gelmiş olur. Bu heterojenliğin diğer bir sebebi de her işte çok iyi olan bir sistem yoktur. Bilgisayarların, işletim sistemlerinin ve ağ platformlarının oluşturabileceği her hangi bir bileşim, o ağ içinde yapılacak işlemlerin sadece bazıları için çok iyidir. Bir başka neden de, ağ içindeki çeşitlilik ağı çıkabilecek sorunlara karşı daha esnek yapar. Çünkü belirli bir makinada, işletim sisteminde veya uygulamada çıkabilecek bir sorunun başka uygulama veya işletim sistemi kullanan ağ parçalarını etkilemesi çok olası değildir.

Bilgisayar ağlarının heterojenliği engellenemeyeceği için, dağıtılmış sistem geliştiricileri bunu göz önünde bulundurmalıdırlar. Herhangi bir dağıtılmış sisteme uygun bir yazılım geliştirmenin zorluğuna ek olarak, bu yazılımı üstelik heterojen olan bir dağıtılmış sistem için geliştirmek neredeyse imkansızdır. Böyle bir yazılım, dağıtılmış sistemlerin alışılagelmiş sorunları olan; ağın güvenliği ve bölünmesi, bazı sistemlerin çökmesi ve kaynak erişimi ve paylaşımı sorunları dışında heterojenliğin ortaya çıkardığı diğer sorunları da halledebilmelidir.

Örneğin; ağdaki yeni bir platformda kullanılmak için bir ağ uygulamasını taşırken ortaya çıkan sorunlar, uygulamanın birden fazla çeşidini yapmamıza sebep olabilir. Eğer uygulamanın herhangi bir çeşidinde bir değişiklik yaparsanız, aynı uygulamanın tüm çeşitlerini değiştirmeli, test etmeli ve hepsinin kendi bileşenleri içinde çalıştığına emin olmalısınız. Bu işin zorluğu da ağ içindeki platform sayısı arttıkça kat kat artmaktadır.

Heterojenlik bu çerçevede sadece donanım ve işletim sistemleriyle ilgili değildir. Gürbüz bir dağıtılmış uygulama yazmak, aşırı karmaşık olmasından ve fazla ayrıntı içermesinden dolayı çok zordur. Sonuç olarak bu uygulamaların geliştiricileri, kütüphane ve araçları çok kullanırlar. Bu da dağıtılmış uygulamaların katmanlı uygulamalar ve kütüphanelerden oluşmasından dolayı heterojen olması demektir.

Aşağıdaki iki anahtar kural ile heterojen dağıtılmış sistemlere, uygulama yazmak kolaylaşabilir.

1) Çok çeşitli sorunları halledebilmenize yardım edecek ortamdan bağımsız modeller ve

soyutlamalar bulmak.

2) Performanstan çok kaybetmeden alt seviyedeki karmaşıklıkları olabildiğine saklamak.

Bu iki kural dağıtılmış olsun yada olmasın, herhangi bir taşınılabilir uygulamayı geliştirmek için yeterlidir. Fakat, dağıtımın eklediği karmaşıklıklar, her kuralı ayrı ayrı daha önemli kılmaktadır. Doğru soyutlama ve modelleri kullanmak, tüm dağıtılmış heterojen karmaşıklık üzerine yeni bir homojen uygulama geliştirme katmanı oluşturmayı olanaklı kılabilir. Bu katman, tüm alt seviye ayrıntıları saklar ve geliştiricilerin uygulamalarının kullanıldığı, alt seviye ağ ayrıntılarıyla uğraşmadan doğrudan sorunları halletmelerini sağlar.

OMG tarafından hazırlanmış olan CORBA, heterojen sistemlerde dağıtılmış programlamanın ihtiyacı olan esnek soyutlamaları ve hizmetleri dengeli bir şekilde karşılar.

2.2.2 Kavramlar ve Terminoloji

CORBA taşınabilir dağıtılmış nesneye dayalı uygulamalar için ortamdan bağımsız bir programlama arayüzü ve modeli sağlar. CORBA programlama dillerinden, bilgisayar ortamlarından ve ağ protokollerinden bağımsızlığı sayesinde, yeni uygulamalar geliştirmeye ve bu uygulamaların var olan dağıtılmış sistemlerle ortaklaşa çalışabilmesine uygundur.

CORBA'daki en önemli terimler aşağıda verilmiştir:

- CORBA nesnesi, ORB (Nesne İstek Aracısı) tarafından bilinen ve de istemcilerin kendisi üzerinde isteklerde bulunabilecekleri sanal bir varlıktır.

- Hedef nesne, CORBA istek çağrısı bağlamında isteğin hedefi olan nesnedir. Bir isteğin hedef nesnesi sadece isteği çağırmak için kullanılan nesne referansı aracılığıyla belirlenir.

- İstemci, bir CORBA nesnesi üzerinde istek çağıran varlıktır. İstemci ve CORBA nesnesi, aynı adres uzayında da, farklı adres uzaylarında da bulunabilirler. İstemci terimi sadece belirli bir istek bağlantısında geçerlidir. Bir istek için istemci olan bir uygulama, başka bir istek için sunucu olabilir.

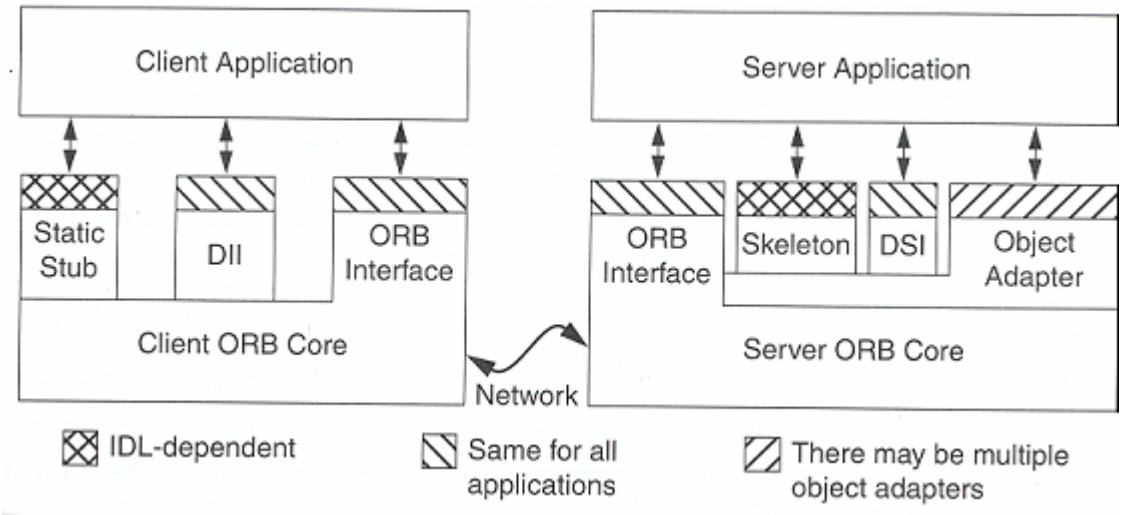
- Sunucu, içinde bir veya daha fazla CORBA nesnesi bulunan uygulamadır. Bir istek için sunucu olan bir uygulama, başka bir istek için istemci olabilir.

- İstek, bir istemci tarafından, bir CORBA nesnesi üzerinde bir işlemin çağırılmasıdır. İstekler, istemciden sunucudaki hedef CORBA nesnesine doğrudur. Hedef nesne de eğer istek gerektiriyorsa sonuçları istemciye geri gönderir.

- Nesne referansı, bir CORBA nesnesini belirlemeye, bulmaya ve adreslemeye yarar. İstemciler için nesne referansları, opak varlıklardır. İstemciler, nesne referanslarını, isteklerini nesnelere göndermek için kullanırlar. Bir nesne referansı sadece bir CORBA nesnesini ifade eder.

- Hizmetçi, bir veya birden fazla CORBA nesnesini gerçekleyen programlama dili varlığıdır. Hizmetçiler, CORBA nesnelerini gerçekleştirirler. C++'da hizmetçiler, verilen bir sınıfın nesneleridir.

2.2.3 CORBA'nın Özellikleri



Şekil2.4 : CORBA

Şekildeki kavramların Türkçe karşılıkları:
Client Application: İstemci Uygulama
Statik Stub: Statik Koçan
DII: Dinamik Çağrı Arayüzü
ORB Interface: ORB Arayüzü
Client ORB Core:İstemci ORB'unun Merkezi
Network: Ağ
Server Application: Sunucu Uygulama
Skeleton: İskelet
DSI: Dinamik İskelet Arayüzü
Object Adapter: Nesne Adaptörü
Server ORB Core: Sunucu ORB'unun Merkezi
IDL-dependent: IDL'e bağlı
Same for all applications: Tüm uygulamalar için aynı
There may be multiple object adapters: Birçok nesne adaptörü olabilir

2.2.3.1 Genel İstek Akışı

1) İstemci isteği ORB'unun merkezine iletir. İstemci statik koçanları kullanarak da Dinamik Çağrı Arayüzünü kullanarak da istek yapabilir.

2) İstemci ORB'unun merkezi, isteği sunucu uygulamaya bağlanmış olan ORB'unun merkezine iletir.

3) Sunucu ORB'unun merkezi isteği nesne adaptörüne iletir.

4) Nesne adaptörü de isteği hedef nesneyi gerçekleyen hizmetçiye iletir. İstemci gibi sunucu da statik veya dinamik iletim mekanizmalarından birini seçebilir.

5) Hizmetçi isteği yerine getirdikten sonra, cevabı istemci uygulamaya geri gönderir.

2.2.3.2 OMG Arayüz Tanımlama Dili (Interface Definiton Language (IDL))

İstemci dağıtılmış bir nesne üzerinde işlem çağırabilmek için bir nesne tarafından çağırılan arayüzü bilmelidir. Bir nesnenin arayüzü, o nesnenin desteklediği işlemleri ve bu işlemlere gönderilen ve bu işlemlerden geri dönen parametrelerini gösterir. Aynı zamanda istemcilerin, bu istemlerin amaçları ve anlamları hakkında da bilgileri olmalıdır.

CORBA'da, nesne arayüzleri, OMG arayüz tanımlama dili ile tanımlanır. IDL, Java veya C++ gibi bir programlama dili değildir. Nesneler veya uygulamalar IDL'de gerçeklenemezler. IDL'nin tek amacı, nesne arayüzlerinin programlama dillerinden bağımsız olarak tanımlanabilmesidir. [4]

2.2.3.3 Dile Karşı Düsürme (Language Mapping)

Dile karşı düşürme IDL'nin değişik programlama dillerine nasıl dönüştüreleceğini belirtir. [4] CORBA, IDL tanımlamalarının, hedef programlama dillerindeki karşılıklarının nasıl olduklarını belirten belgeleri içermektedir. Örneğin, hedef dil C++ veya Java ise dile karşı düşürme;

- IDL veri tiplerinin C++ veya Java'daki veri tiplerine olan karşılıklarını belirler.

- IDL arayüzlerinin, sınıflara, IDL işlemlerinin üye fonksiyonlara (member function) (C++) veya metodlara (Java) nasıl karşılık düşürüldüklerini belirler.

- Sunucuda CORBA nesnesinin nasıl gerçekleneceğini açıklar. [1]

OMG şu anda C, C++, Smalltalk, COBOL, Ada ve Java için dile karşılık düşürmeleri standartlaştırmıştır. [4]

2.2.3.4 İşlem Çağırma ve Gönderim Hizmetleri

CORBA uygulamaları istek alarak veya CORBA nesneleri üzerinde istek çağırarak çalışırlar. İstek çağrımı için iki genel yaklaşım vardır.

a) Statik çağrı ve gönderim,
Bu yaklaşımda, OMG IDL, uygulamalarla birleştirilen koçan ve iskeletlere dönüştürülür. Koçan ve iskeletlerin uygulamayla birleştirilmesi sayesinde uygulama, uzaktaki nesnenin IDL tanımından hedef dile karşılık düşürülen fonksiyon ve programlama dili tipleri hakkında statik bilgiye sahip olur.

b) Dinamik Çağrı ve Gönderim

Bu yaklaşımda CORBA isteklerinin oluşturulması ve de gönderilmesi çalışma zamanı olur.

C++ gibi statik tiplere sahip bir dili kullanan geliştiriciler, genellikle statik çağrı yaklaşımını kullanırlar. Dinamik yaklaşım, istekler alıp iletirken, içerdikleri arayüzlerin ve tiplerin derleme zamanı bilgilerine ihtiyaç duymayan kapı ve köprü uygulamaları için uygundur.

2.2.3.5 Nesne Adaptörleri

CORBA'da nesne adaptörü hizmetçiler ve ORB arasında bir tutkal rolü oynar. CORBA nesne adaptörünün üç temel görevi vardır.

- İstemcilerin nesneleri adreslemelerini sağlayan nesne referanslarını yaratırlar.

- Hedef nesnelerin her birinin bir hizmetçi tarafından gerçekleşmesini sağlar.

- Sunucu ORB'u tarafından gönderilen isteği alır ve bu isteği hedef nesneyi gerçekleyen hizmetçiye iletir.

2.2.3.6 ORB'lar Arası Protokoller

CORBA 2.0'dan önce en çok CORBA'nın standart bir protokol spesifikasyonundan yoksunluğundan şikayet ediliyordu. ORB uygulamalarının haberleşebilmeleri için ORB üreticileri ya kendi ağ protokollerini yazıyorlar yada diğer bir dağıtılmış sistem teknolojisininkini kullanıyorlardı. Bu durumda ORB'lar birbirleriyle haberleşemiyorlardı.

CORBA 2.0 General Inter-ORB Protocol (GIOP) adındaki ORB'ların birlikte çalışabilmelerini sağlayan mimariyi de tanıttı. GIOP transfer sentaksını ve standart bir mesaj formatı kümesini tanımlayan soyut bir protokoldür. GIOP farklı ORB'ların herhangi bir bağlantılı iletişim üzerinden haberleşebilmelerine izin verir. Internet Inter-ORB Protocol (IIOP), GIOP'nin TCP/IP üzerinde nasıl gerçeklendiğini belirtir. CORBA 2.0'ı destekleyen tüm ORB'lar GIOP ve IIOP'yi gerçeklemelidirler.

2.2.4 İstek Çağrısı

İstemciler nesneleri mesaj göndererek kullanırlar. İstemci bir işlemi çağırdığında ORB nesneye bir mesaj gönderir. Bir istemcinin, bir nesneye istek gönderebilmesi için o nesnenin referansına sahip olması gerekir. Nesne referansı, hedef nesneyi belirler ve ORB'un doğru yere mesajı iletebilmesi için gerekli tüm bilgileri içerir.

Bir istemci, bir nesne referansını kullanarak bir işlemi çağırdığında ORB aşağıdaki işlemleri yapar.

- Hedef nesneyi bulur.
- Sunucuyu aktifleştirir (eğer aktif değilse)
- Argümanları nesneye iletir.
- Gerekliyse, nesne için bir hizmetçiyi aktifleştirir.

- İsteğin işlemlerinin yapılması için bekler.
- Eğer çağrı başarı ile tamamlanırsa; out ve inout parametreleriyle geri dönüş değerini istemciye gönderir.
- Eğer çağrı başarısız olursa exception geri döner.

2.2.4.1 İstek Çağrısını Özellikleri

- Hedef nesne istemciyle aynı adres uzayında da olabilir, aynı makinada farklı bir süreç tarafından gerçekleşiyor da olabilir, farklı bir makinadaki bir süreç tarafından da gerçekleniyor olabilir. Bunu istemci bilmez. Sunucular hep aynı makinede de olmak zorunda değildir, istemcinin haberi olmadan, sunucu, bir bilgisayardan diğerine taşınabilir.

- İstemci hangi sunucunun hangi nesneyi gerçekleştirdiğini bilmek zorunda değildir

- Dilden bağımsızlık: İstemci için, sunucu tarafından kullanılan dilin önemi yoktur. Örneğin C++ bir istemci Java bir sunucudan hizmet görebilir ve bundan haberi olmaz. Varolan bir nesne için kullanılan programlama dili, istemciyi etkilemeden değiştirilebilir.

- Gerçeklemeden bağımsızlık: İstemci, gerçeklemenin nasıl olduğunu bilmez. Örneğin sunucu C++ gibi nesneye dayalı bir dille gerçekleyebileceği gibi, C gibi işlevsel bir dille de gerçekleyebilir. İstemci, nesnenin nasıl gerçeklendiğini bilmeden ayrı nesneye dayalı manayı anlar.

- Mimariden bağımsızlık: İstemci, sunucu tarafından kullanılan MİB mimarisi hakkında bilgiye sahip değildir ve böylece byte sıralama gibi alt seviye ayrıntılarla da uğraşmaz.

- İşletim sisteminden bağımsızlık: İstemci, sunucu tarafından kullanılan işletim sisteminden bağımsızdır.

- Protokolden bağımsızlık: İstemci, mesajı göndermek .için hangi haberleşme protokolünün kullanıldığını bilmez.

- Taşımadan bağımsızlık: İstemcinin, mesajı iletmek için kullanılan taşıma ve veri bağı katmanı hakkında bilgisi yoktur. ORB'lar, Eternet, ATM, jetonlu halka gibi çeşitli ağ teknolojileri kullanılabilir.

2.2.4.2 Nesne Referansının Anlamı

Nesne referansları, C++'daki nesne işaretçilerine benzerler. Fakat bir nesne referansı istemcinin kendi adres uzayında gerçeklenmiş bir nesneyi gösterebileceği gibi farklı bir süreç tarafından gerçeklenen bir nesneyi de gösterebilir. Aşağıda nesne referanslarının bazı özellikleri verilmiştir.

- Her nesne referansı sadece bir nesneyi belirtir.
- Farklı referanslar, aynı nesneyi belirtebilirler.
- Referanslar boş olabilirler (hiçbir yere işaret etmeyebilir).
- Referanslar geçersiz olabilirler (C++'daki işaretçilerin yok edilmiş olan bir nesneyi

işaret etmesi gibi)
- Referanslar opaktır. İstemcinin referansın içeriğine bakmaya izni yoktur.
- Her nesne referansı, o nesnenin desteklediği arayüzü hakkında bilgi içerir.
- Referanslar geç bağlamayı desteklerler.
- Referanslar sürekli olabilirler.
- Bir üreticin ORB'u, başka bir üreticinin ORB'u tarafından yaratılmış olan bir referansı kullanabilir.

2.2.4.3 Referans Elde Etme

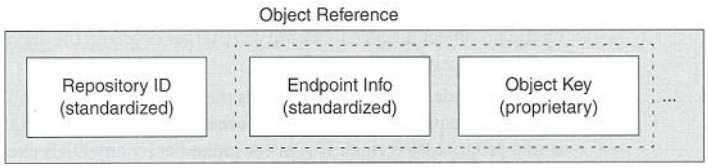
En çok kullanılan 3 nesne referansı elde etme yolu:

- Sunucu, referansı yazı katarı haline dönüştürür ve bir dosyaya atar. [4] İstemci nesne referansını bu dosyadan elde eder. Bu yol genelde, gerçek dağıtılmış sistemlerde kullanılmayan, fakat programları test etmek için kullanılan bir yöntemdir.

- CORBA adlandırma servisini (naming service) kullanma: Adlandırma servisi, isimlerle nesne referanslarını ilişkilendirir ve saklar. İstemci sadece adlandırma servisinin yerini ve kullanacağı nesnenin ismini bilmek zorundadır.

- CORBA bayilik nesne servisini kullanma: Bayilik servisi (trading service) adlandırma servisine benzer fakat daha esnektir. Bayilik servisiyle, nesnelerin özelliklerini sorgulayarak arama da yapılabilir. [1]

2.2.4.4 Bir Nesne Referansının İçeriği



Şekil 2.5: Nesne referansının içeriği

Şekildeki kavramların Türkçe karşılıkları:
Object Reference: Nesne Referansı
Repository ID: Depo Kimliği
Endpoint Info: Son Nokta Bilgisi
Object Key: Nesne Anahtarı
standardized: standartlaşmıştır
proprietary: hususi

Bir nesne referansı üç temel bilgi içerir:

- Depo Kimliği: Depo kimliği, arayüzün ayrıntılı bir açıklamasını, nesneyi arayüz deposuna yerleştirmemize yarar (eğer ORB tarafından arayüz deposu sağlanıyorsa).

- Son Nokta Bilgisi: Bu alan, nesneyi gerçekleyen sunucuyla fiziksel bir bağlantı kurmak için, ORB'un ihtiyaç duyduğu tüm bilgileri içerir.

- Nesne Anahtarı: Depo kimliği ve son nokta bilgisi standartlaşmıştır fakat nesne anahtarı ORB'a bağlı bilgi içerir. Bu bilginin nasıl düzenlendiği ve kullanıldığı ORB'a bağlıdır.

2.2.5 C++ ile Bir CORBA Uygulaması Geliştirmenin Aşamaları

C++ ile bir CORBA uygulaması yazarken genellikle aşağıdaki adımlar izlenir:

1) Uygulamanın nesneleri belirlenir ve bu nesnelerin arayüzleri IDL'de yazılır.

Herhangi bir programı, nesneye dayalı bir programlama dilini kullanarak geliştirirken yapıldığı gibi önce nesneler belirlenir daha sonra da nesnelerin arayüzleri ve de ilişkileri belirlenir. İşin bu kısmı genellikle zordur ve de CORBA bu kısmı basitleştiremez.

2) IDL tanımlamaları derlenir ve IDL derleyicisi tarafından C++ koçan ve iskeletleri oluşturulur.

ORB'lar genellikle IDL'leri dile karşılık düşürme kurallarını izleyerek derleyip, istemci koçanlarını ve de sunucu iskeletlerini oluşturan derleyicileri sunarlar. IDL derleyicileri, C++ için genellikle başlık dosyaları oluştururlar. Bu dosyalarda vekil sınıflar, sunucu iskeletleri ve desteklenen tipler için bildirimler bulunur. Aynı zamanda IDL derleyiciler, başlık dosyasında bildirilen sınıf ve tipleri gerçekleyen C++ ile yazılmış gerçekleme dosyalarını da oluştururlar. IDL tanımlarını C++'ya dönüştürerek, IDL arayüzünü destekleyen CORBA nesnelerine erişen istemciler ve bu nesneleri gerçekleyen hizmetçilerin yazılabilmesi için gerekli kod tabanına sahip olunur.

3) CORBA nesnelerini gerçekleyebilecek olan C++ hizmetçi sınıfları bildirilir ve de gerçeklenir.

ORB, istekleri iletmeden önce, her CORBA nesnesi bir C++ hizmetçi sınıfıyla gerçeklenmelidir. CORBA nesnesinin, istemcilere sunacağı servisleri gerçekleştirmek için hizmetçi sınıflar bildirilir ve bu sınıfların üye fonksiyonları gerçeklenir.

4) Sunucu için ana program yazılır.

Tüm C++ programlarında olduğu gibi ana fonksiyon C++ CORBA uygulamasının başlangıç ve bitiş noktalarını sağlayacaktır. Sunucu için ana program ORB'u ve POA'yı başlatmalı, gerekli sayıda hizmetçi oluşturmalı, hizmetçileri CORBA nesnelerini gerçeklemeleri için CORBA nesneleriyle ilişkilendirmeli ve son olarak istekler için dinleme durumuna geçmelidir.

5) Sunucu için yazılan gerçekleme dosyaları koçan ve iskeletlerle birlikte derlenip bağlanarak çalışabilir sunucu oluşturulur.

Bir C++ sunucusu için yazılımcı metodların gerçeklemelerini yazar. Derleyici tarafından

üretilen koçan ve iskeletler, IDL ile tanımlanmış olan arayüzdeki tiplerin gerçeklemelerini ve gelen CORBA isteklerini hizmetçilerde fonksiyon çağrılarına dönüştürmek için gerekli istek iletim kodunu içerirler.

6) İstemci kodu yazılır ve derleyici tarafından üretilen koçanlarla birlikte derlenir ve bağlanır.

İstemci önce nesne referanslarını elde eder. Sonra nesne tarafından sunulan hizmetlerden faydalanmak için CORBA nesnesi üzerinde işlem çağrıları yapar. İstemci kodu, istek çağrılarını normal C++ fonksiyon çağrıları yaparmış gibi yapar. Koçanlar, bu fonksiyon çağrılarını sunucudaki nesneler üzerinde CORBA istek çağrılarına dönüştürür. [4]

3. GERÇEKLENEN ÇALIŞMA

Çalışmada, CORBA'nın iki temel özelliği olan ortamdan bağımsızlık ve dilden bağımsızlık özelliklerini destekleyen ve nesneye dayalı programlamanın temel kavramları olan encapsulation, kalıtım ve de polymorphism kavramlarının gerçeklendiği, bir dağıtılmış nesne sistemi uygulaması yapılmıştır.

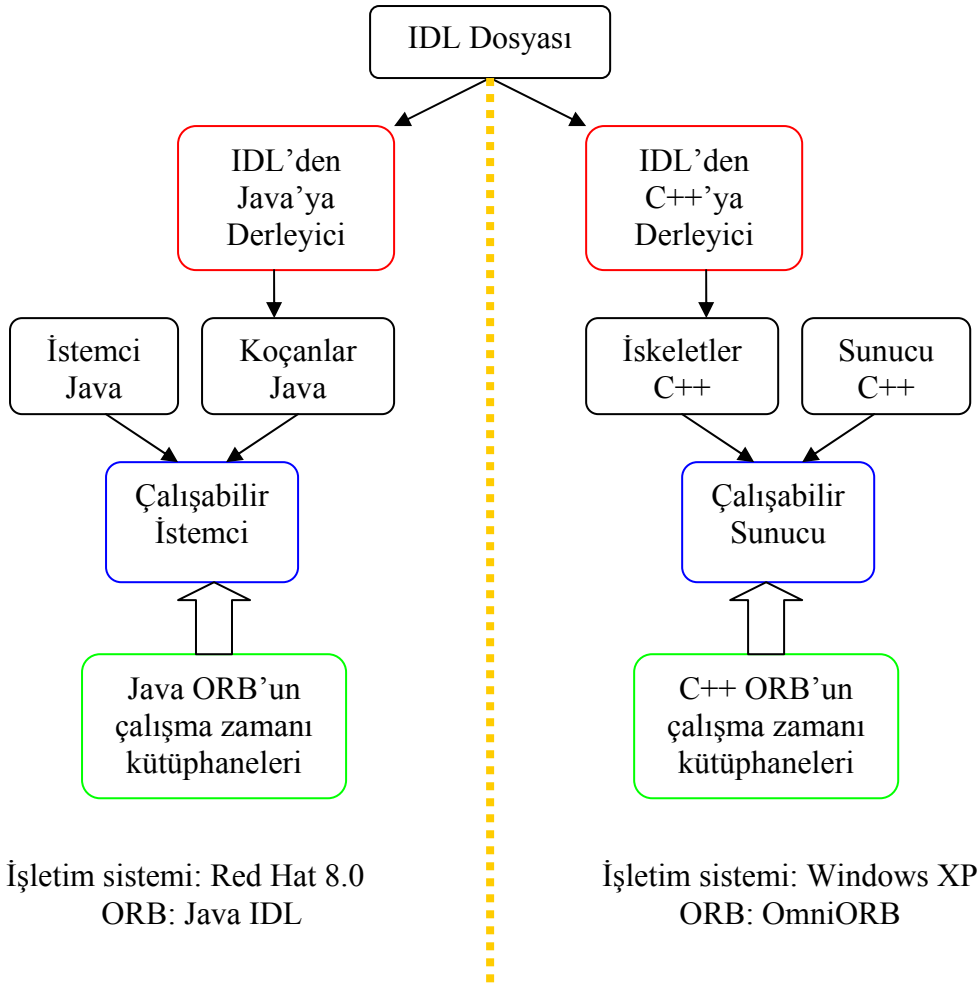
3.1 Uygulamada Çözülen Problem

Aşağıda özelliklere sahip sınıflar için OMG IDL ile bir arayüz tanımlama dosyası oluşturun. Bu sınıfların gerçeklemelerini içeren sunucu bir program ve de bu sınıfların özelliklerini kullanan bir istemci program yazın. Programınızda nesneye dayalı programlamanın 3 temel özelliği olan encapsulatin, polymorphism ve de kalıtım özelliklerini de gerçekleyin. Bunun yanında uygulamanızı yazarken CORBA'nın iki temel özelliği olan dilden bağımsızlık ve de ortamdan bağımsızlık özelliklerini kullanın.

Nokta: İki koordinatı vardır. Hareket edebilir, iki nokta arasındaki uzaklığı geri dönen bir fonksiyonu ve de noktanın koordinatlarını geri dönen bir fonksiyonu vardır.
Doğru Parçası: Hareket edebilir, uzunluğunu geri dönen bir fonksiyonu vardır.
Üçgen: Hareket edebilir, uzunluğunu ve alanını geri dönen fonksiyonları vardır.
Dikdörtgen: Hareket edebilir. Bir kenarı x eksenine paraleldir. Uzunluğunu ve alanını geri dönen fonksiyonları vardır .
Daire: Hareket edebilir. Uzunluğunu ve alanını geri dönen fonksiyonları vardır .
Yay: Daire dilimi şeklindedir. Aşağıda gösterildiği şekilde iki açısı(a1, a2) vardır. Hareket edebilir.uzunluğunu ve alanını geri dönen fonksiyonları vardır.

Her sınıfın özelliklerini ekrana basmasına yarayan bir fonksiyonu da vardır. [2]

3.2 Programın Geliştirilme Aşamaları



Şekil 3.1: Programın geliştirilme aşamaları

Uygulama geliştirilirken şu aşamalar izlendi.

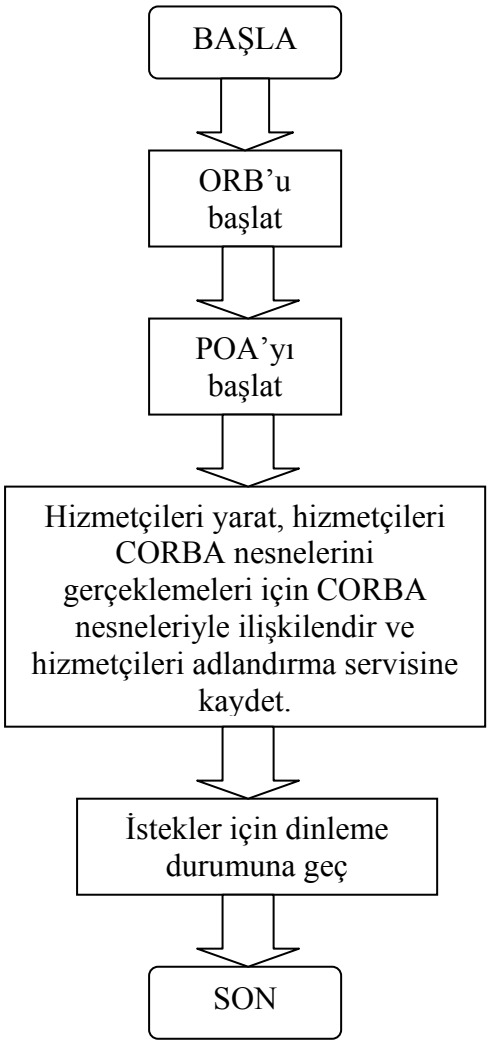
- Uygulamanın sunucusunda bulunacak nesneler için gerekli IDL dosyası arayüz tanımlama diliyle yazıldı.

- IDL dosyası, sunucu için omniORB nesne istek aracısının C++ için olan derleyicisiyle derlendi ve iskelet dosyaları oluşturuldu, istemci için Java IDL nesne istek aracısının Java için olan derleyicisiyle derlendi ve koçan dosyaları oluşturuldu.

- Sunucu tarafında, arayüzde bildirilmiş olan CORBA nesnelerini gerçekleyecek olan hizmetçi sınıflar bildirildi ve de gerçeklendi. Bundan sonra sunucu için bir ana program yazıldı. Oluşturulmuş olan gerçekleme dosyası ve ana program ile birlikte iskelet dosyaları derlenip bağlanarak çalışabilir sunucu elde edildi.

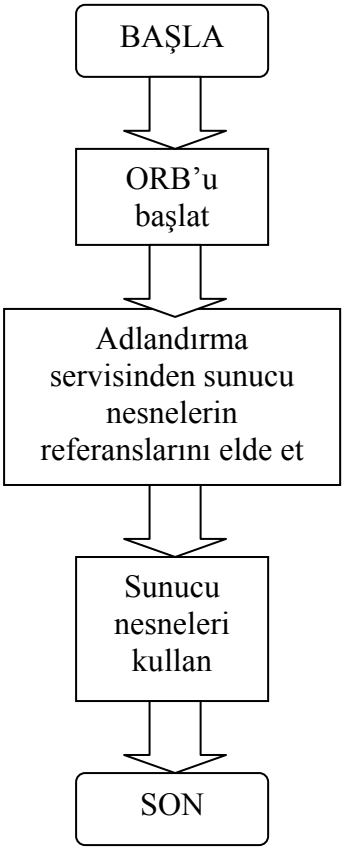
- İstemci tarafında, arayüzdeki nesnelerin sunduğu işlemleri test etmek için gerekli kod yazıldı. Bu dosyayla koçan dosyaları birlikte derlenip bağlandı ve de çalışabilir istemci elde edildi.

3.3 Sunucu Programın Akış Şeması



Şekil 3.2: Sunucu programın akış şeması

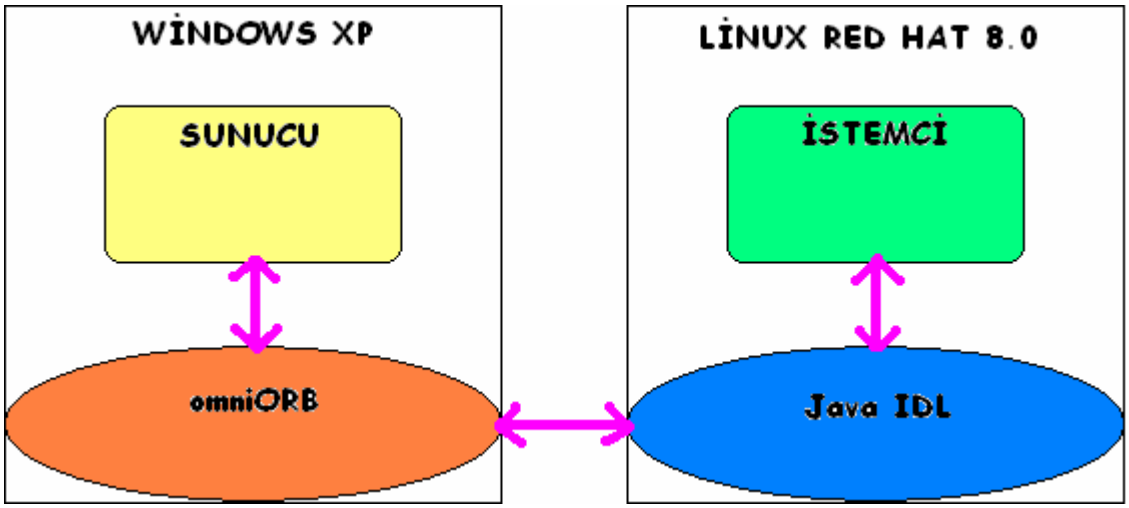
3.4 İstemci Programın Akış Şeması



Şekil 3.3: İstemci programın akış şeması

3.5 Ortamdan Bağımsızlık Özelliğinin Gösterilmesi

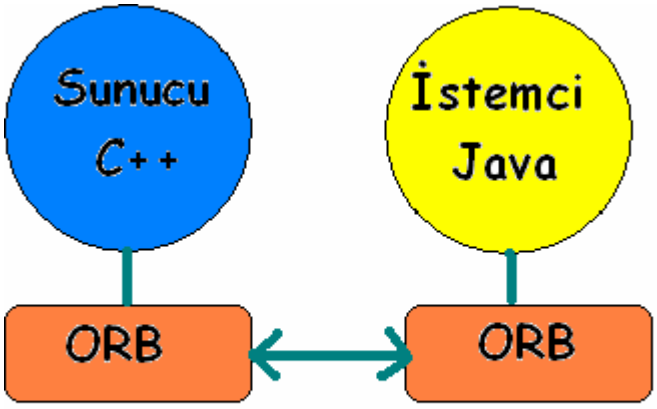
Sunuzu program Windows XP işletim sistemi üzerinde, omniORB nesne istek aracısını kullanarak gerçeklenmiştir. İstemci program ise Linux Red Hat 8.0 işletim sistemi üzerinde Java IDL nesne istek aracısını kullanarak gerçeklenmiştir. Bakınız şekil 3.4.



Şekil 3.4: Ortamdan bağımsızlık

3.6 Dilden Bağımsızlık Özelliğinin Gösterilmesi

Sunucu program C++ dili kullanılarak gerçeklenmiştir, istemci program ise Java dili kullanılarak gerçeklenmiştir. Bakınız şekil 3.5.



Şekil 3.5: Dilden bağımsızlık

3.7 Encapsulationın Gerçeklenmesi

Gerçekleme sırasında IDL'deki arayüzler sınıflarla gerçeklenerek nesneye dayalı programlamanın encapsulatin özelliği gösterilmiş oldu. Sınıfların içinde metodlar ve de veriler encapsule edilmiştir. Aşağıdaki kodlara bakınız.

IDL Dosyası:

```
interface Point {
    long xkac();
```

```
    long ykac();
    double fark(in Point inObject1);
    Coordinates koordinatlar();
    void move(in long yon, in long birim);
    double length();
    double area();
    void print();
};
```

Sunucu Program:

```
class Point_i : public virtual POA_Point,
        public PortableServer::RefCountServantBase
{
public:
    virtual CORBA::Long xkac();
    virtual CORBA::Long ykac();
    virtual CORBA::Double fark(Point_ptr inObject1);
    virtual Coordinates koordinatlar();
    virtual void move(CORBA::Long yon, CORBA::Long birim);
    virtual CORBA::Double length();
    virtual CORBA::Double area() ;
    virtual void print();
    Point_i(CORBA::Long x_in, CORBA::Long y_in) : x(x_in), y(y_in) ;
    Point_i() : x(0), y(0) ;
private:
    CORBA::Long x, y;
};
```

3.8 Kalıtımın Gerçeklenmesi

IDL dosyasındaki kalıtım hiyerarşisi sunucu programın sınıflarıyla gerçeklenmiştir. Aşağıdaki kodları inceleyiniz.

IDL Dosyası:

```
interface Point {
    long xkac();
    long ykac();
    double fark(in Point inObject1);
    Coordinates koordinatlar();
    void move(in long yon, in long birim);
    double length();
    double area();
    void print();
};
interface Line : Point {
};
```

Sunucu Program:

```
class Point_i : public virtual POA_Point,
        public PortableServer::RefCountServantBase
{
public:
   virtual CORBA::Long xkac();
   virtual CORBA::Long ykac();
   virtual CORBA::Double fark(Point_ptr inObject1);
   virtual Coordinates koordinatlar();
   virtual void move(CORBA::Long yon, CORBA::Long birim);
   virtual CORBA::Double length();
   virtual CORBA::Double area() ;
   virtual void print();
   Point_i(CORBA::Long x_in, CORBA::Long y_in) : x(x_in), y(y_in) ;
   Point_i() : x(0), y(0) ;
private:
   CORBA::Long x, y;
};

class Line_i : public virtual Point_i, public virtual POA_Line
{
public:
   Line_i() ;
   Line_i(CORBA::Long x_in1, CORBA::Long y_in1, CORBA::Long x_in2,
CORBA::Long y_in2);
   virtual void move(CORBA::Long yon, CORBA::Long birim);
   virtual CORBA::Double length();
   virtual void print();
private:
   Point_i p1, p2;
};
```

3.9 Polymorphismin Gerçeklenmesi

Nesneye dayalı programlanın 3. bir özelliği olan polymorphism özelliği de kullanılan virtual üye fonksiyonlarla gerçeklendi. Aşağıdaki kodları inceleyiniz.

```
class Point_i : public virtual POA_Point,
        public PortableServer::RefCountServantBase
{
public:
    virtual CORBA::Double length()
   {
            return 0;
   }
   :
}
```

```
class Line_i : public virtual Point_i, public virtual POA_Line
{
public:
   CORBA::Double Line_i::length()
{
   return sqrt((p1.xkac() - p2.xkac()) * (p1.xkac() - p2.xkac()) + (p1.ykac() - p2.ykac()) *
(p1.ykac() - p2.ykac()));
}
   :
}
```

Point cinsinden bir nesneye Line cinsinden bir nesne atanmıştır ve bu nesnenin length metodu çağırıldığında Point nesnesinin değil, Line nesnesinin length metodu çağırılmaktadır.

```
Line[] lines = new Line[2];
:
Point pm;
pm = lines[1];
System.out.println("L2'nin uzunluğu = " + pm.length());
```

Yukarıdaki kod parçasının çıktısı aşağıda verilmiştir.

```
L2'nin uzunluğu = 5.0
```

4. SONUÇLAR

Bu çalışma sonucunda bir dağıtılmış nesne sistemi uygulaması gerçekleştirilmiştir. Bu çalışmada nesneye dayalı programlamanın en önemli temel kavramları da gerçeklenmiştir. Bu sayede bir dağıtılmış nesne sistemi uygulamasında kullanılabilecek temel özellikler de denenmiştir.

Çalışmada arakatman olarak CORBA kullanılmıştır. CORBA'nın temel iki özelliği olan ortamdan ve dilden bağımsızlık özelliklerini de destekleyen bir çalışma yapabilmek için istemci ve sunucu farklı işletim sistemleri üzerinde farklı nesne istek aracılarıyla ve de farklı diller kullanılarak gerçeklenmiştir.

Çalışmada en çok zaman harcanan kısım ORB kurma ve de en basit bir programı derleme kısmı olmuştur. Bu işlemi gerçekleştirdikten sonraki program yazma kısmının tek bir bilgisayarda çalışacak olan bir nesneye dayalı program yazmaktan çok büyük bir farkı yoktur.

5. KAYNAKLAR

[1] Bolton, F. **Pure CORBA,** Sams Publishing 2002
[2] Buzluca, F. **Object Oriented Programming Second Homework,** 2003
[3] Comer, D. E. **Computer Networks and Internets with Internet Applications,** Prentice Hall 2001
[4] Henning, M., Vinoski, S. **Advanced CORBA Programming with C++,** 1999