ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BAĞIMSIZ PLATFORMLAR İLE WEB SERVİSLERİNİN GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Faruk Eskicioğlu

ÇANAKKALE - 2006

ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BAĞIMSIZ PLATFORMLAR İLE WEB SERVİSLERİNİN GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Hazırlayan : Faruk Eskicioğlu
Danışman : Doç.Dr. M. Ali Salahlı

ÇANAKKALE - 2006

YÜKSEK LİSANS TEZİ SINAV SONUÇ FORMU

Faruk Eskicioğlu, tarafından **Doç. Dr. M. Ali Salahlı** yönetiminde hazırlanan **"Bağımsız Platformlar İle Web Servislerinin Geliştirilmesi"** başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Yüksek Lisans tezi olarak kabul edilmiştir.

.....

Yönetici

.....

Jüri Üyesi

.....

Jüri Üyesi

.....

Jüri Üyesi
(5 üyeli jürilerde)

.....

Jüri Üyesi
(5 üyeli jürilerde)

.....

Müdür
Fen Bilimleri Enstitüsü

İÇİNDEKİLER

ÖZ……... I

ABSTRACT…… II

SİMGELER VE KISALTMALAR……. III

ÇİZELGELERİN LİSTESİ……. IV

ŞEKİLLERİN LİSTESİ…….. VI

GİRİŞ……..……..…….. 1

1. YAZILIM BÜTÜNLEŞTİRMESİ ……..…..... 2

1.1. Yazılım Bütünleştirmesi İhtiyacı..... 2

1.2. Yazılım Bütünleştirmesi Çözümleri..... 4

1.2.1 CORBA..... 5

1.2.2 COM/DCOM..... 7

1.2.3 Java/RMI..... 13

1.2.4 Yazılım Bütünleşmesi Çözümlerinde Gelişmeler..... 17

2. SOA ve WEB SERVİSLERİ.....…..... 20

2.1. Servis Yönelimli Mimari (SOA) Nedir?..... 20

2.2. Web Servisleri..... 22

2.2.1 Web Servisleri Tanımı..... ...…….. 22

2.2.2 Web Servisleri Mimarisi.....…….... 24

2.2.3 Web Servisleri ve CORBA Kıyaslaması.....…….. 40

3. Yazılım Bütünleşmesi Çözümlerinin E-Posta İstemcisi Alanında Uygulanması... 43

3.1 Uygulama Alanının Tanımlanması.....……..... 43

3.2 Uygulama Alanının Bütünleşik Yazılım Mimarisi ve Bileşenleri..... 44

3.2.1 E-Posta İstemcisi Web Servisi..... 44

3.2.2 E-Posta İstemcisi Web Servisi Kullanıcı Uygulaması..... 46

3.3 Platform ve Teknoloji Seçimleri.....…… 48

SONUÇ……..……. 49

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılan kısaltmalar açıklamaları ile aşağıda sunulmuştur.

Kısaltmalar	Açıklama
API	Application Program Interface (Uygulama Programı Arayüzü)
CLSID	Class Identificator (Sınıf Tanımlayıcısı)
COM	Component Object Model (Bileşen Nesne Modeli)
CORBA	Common Object Request Broker (Ortak Nesne İsteği Aracısı)
DCOM	Distrubuted Common Object Request Broker
FTP	File Transfer Protocol (Dosya Transfer Protokolü)
HTTP	HyperText Transfer Protocol
HTTPS	Secure HyperText Transfer Protocol
IDL	Interface Definition Language (Arayüz Tanımlama Dili)
IIOP	Internet Inter-ORB Protocol
IPSec	IP Security Architecture
JVM	Java Virtual Machine (Java Sanal Makinası)
KBS	Kurum Bilgi Sistemi
MIDL	Microsoft IDL
MIME	Multi-Purpose Internet Mail Extension
MQ	Message Queing
OMG	Object Managment Group
ORB	Object Request Broker
RPC	Remote Procedure Call
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
W3C	World Wide Web Consortium
WSDL	Web Service Description Language
XML	Extenden Markup Language

ÇİZELGELERİN LİSTESİ

Çizelge No	Çizelge Adı	Sayfa No
Çizelge 2.1	Corba ve Web Servisleri yığıtları	41

ŞEKİLLERİN LİSTESİ

Şekil No	Şekil Adı	Sayfa No
Şekil 1.1	Bir kurmun bilgi teknolojileri ortamı	3
Şekil 1.2	Bir CORBA sunucusu ve istemcisi tarafından paylaşılan tek metotlu bir dağıtık nesne	6
Şekil 1.3	ORB`ları kullanılarak gerçekleştirilen bütünleştirme	7
Şekil 1.4	Çeşitli farklı yollarla paketlenmiş bir COM nesnesiyle iletişim kuran bir COM istemcisi	10
Şekil 1.5	COM`un fabrika mekanizması	11
Şekil 1.6	COM arayüzleri, sınıfları ve nesnelerini arasındaki ilişki	11
Şekil 1.7	Davranışın tanımı ve gerçekleştirimi	14
Şekil 1.8	Servisin gerçekleştirimi ve vekili	14
Şekil 1.9	JAVA RMI`nın katmanlı yapısı	15
Şekil 1.10	Vekil Kalıbı	16
Şekil 1.11	RMI`da JVM`lerin bağlanması	17
Şekil 1.12	Temel RPC mimarisi	18
Şekil 2.1	Web servisi modeli	23
Şekil 2.2	Kavramsal Web Servisi yığıtı	25
Şekil 2.3	Temel Web Servisi Yığıtı	26
Şekil 2.4	SOAP kullanarak XML mesajlaşma	29
Şekil 2.5	Temel Servis Tanımı	31
Şekil 2.6	Servis keşif düzlemi	34
Şekil 3.1	Uygulamanın genel görünüşü	43
Şekil 3.2	Uygulamanın detaylı görünüşü	47

GİRİŞ

Geniş ölçekli Bilgi Teknolojileri alt yapısına sahip kurumlar, işlerinin büyümesiyle, kendi iç yapılarındaki farklı platformlardaki sistemler arasında, ya da dış Dünya`daki ve doğal olarak farklı platformlardaki sistemlerle, kendi içyapılarındaki sistemler arasında işletişim kurma ihtiyacını hep duymuşlardır. Başlarda bu ihtiyaçlara çözüm olarak ortaya atılan yöntemlerin, anlık ihtiyaçları karşılamaya yönelik olması, belirli bir alt yapılarının ve bir standartlarının olmaması, nedenleriyle çözümlerinin gerçekleştirimlerinde, ya da icra edilmelerinde en az çözülmeye çalışılan problemin kendisi kadar büyük problemlere veya yüksek maliyetlere sebep olmaları, bilgisayar bilimlerini yazılım bütünleştirmesi konusunda bir standartlaştırma çalışmasına itmiştir.

Başlangıç olarak önde gelen yazılım şirketleri ve bunlardan oluşan konsorsiyumlar bu konuda önemli çalışmalar gerçekleştirmişler ve CORBA, COM/DCOM, JAVA RMI teknolojileri gibi önemli çözümler sağlamışlardır.

Internet`in ve gerçek işlerin yazılım yoluyla gerçekleştirilmelerinin yaygınlaşması, yazılım bütünleştirme kavramının kapsamının farklılaşmasına ve gelişmesine neden olmuştur. Özellikle birbirleriyle bütünleşecek olan yazılım sistemlerinin daha yüksek seviyelerde bağımsız olmaları gereksinimi ortaya çıkmıştır.

Bu durum, mevcut yazılım bütünleştirmesi çözümlerinin yetersiz olmalarına ve SOA mimarisinin ve bu mimarinin bir gerçekleştirimi olan web servislerinin geliştirilmesine ve yaygınlaşmasına neden olmuştur.

1. YAZILIM BÜTÜNLEŞTİRMESİ

Yazılım bütünleştirmesi, bir bilgisayar uygulaması kümesini bütünleştirmek amacıyla, yazılım ve bilgisayar sistemleri mimari prensiplerinin belirli bir standarta uygun bir şekilde kullanılması anlamına gelir (http://en.wikipedia.org/). Veri yayılımını ve belirsiz sayıdaki ağ ile birbirlerine bağlanmış uygulama arasında iş süreci icrasını mümkün kılar (http://www.informatica.com/).

Yazılım bütünleştirmesi, bir kurum içerisindeki bilgisayar uygulamalarının, koordinasyonu, birleştirilmesi ve modernizasyonu amacıyla kullanılan metotlar, araçlar ve planlar için kullanılan bir iş bilgi işlemi terimidir. Tipik olarak bir kurumun mevcut durumda, eski teknolojilerle geliştirilmiş uygulamaları ve veritabanları vardır ve bu kurum, Internet`i, e-ticareti, kurumun kendi özel ağını kullanan yeni bir uygulama kümesine geçiş yaparken mevcut uygulamalarını kullanmak isteyecektir. Böyle bir durumda, yazılım bütünleştirme, mevcut uygulamaların bu yeni yapıya nasıl uygunluk göstereceklerini göz önünde bulundurarak, kurumun işlerinin yeni bütünsel görünümünü ve uygulamalarını yeniden geliştirmeyi ve yeni verileri ve uygulamaları eklerken mevcut varlıkların etkin olarak yeniden kullanılabilirlikleriyle ilgili yolları araştırmayı içerebilir (http://searchwebservices.techtarget.com/).

Daha yalın bir ifadeyle; bir yazılıma ait olan verinin ya da bir fonksiyonun, başka bir yazılım aracalığıyla elde edilmesi ya da kullanılması sürecidir.

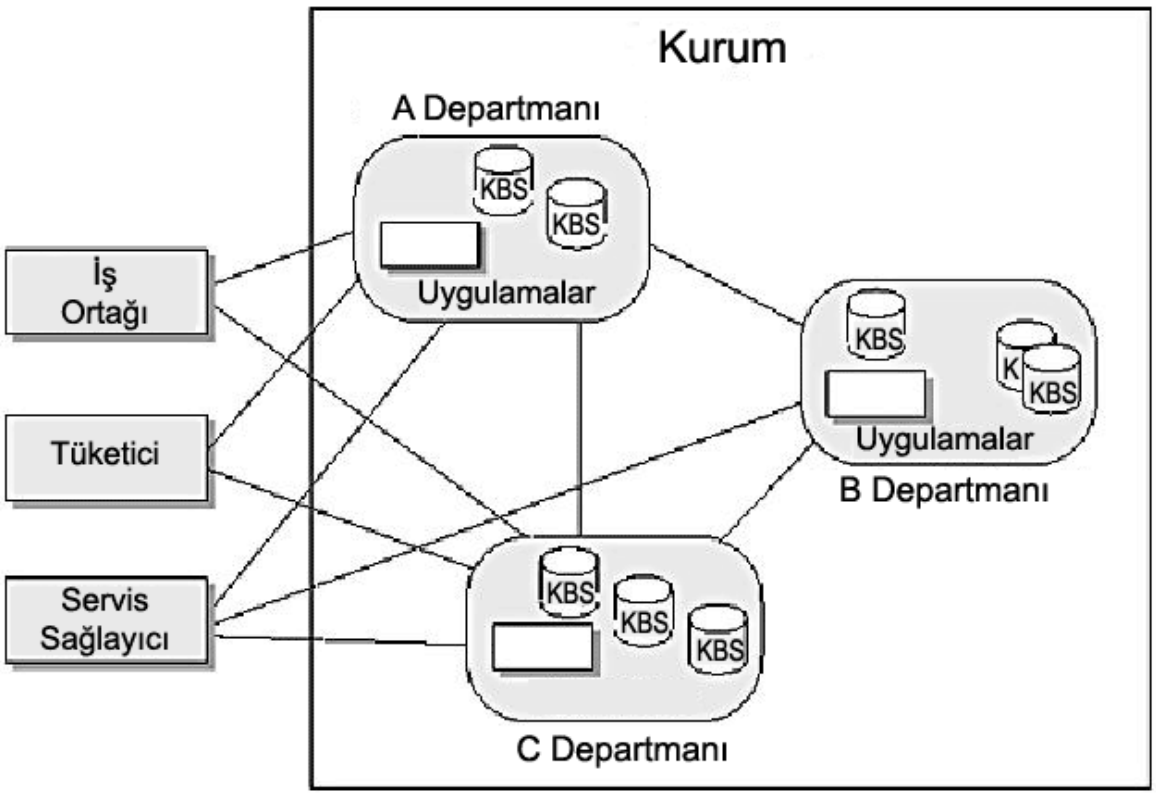
1.1. Yazılım Bütünleştirmesi İhtiyacı

Yazılım bütünleştirmesi, bir kuruma ait bilgi teknolojileri birimindeki alt birimlerin kullandığı farklı sistemlerin arasında olabileceği gibi, farklı kurumların bilgi teknoloji birimleri arasında da olabilir. Bu farklı sistemler farklı donanım ve/veya yazılım üreticileri tarafından geliştirilmiş olabilecekleri için, bütünleştirme işleminin heterojen bir ortamda yapılması gereksinimi olabilir.

Yazılım bütünleştirmesi standartları geliştirilmeden önce, uygulamaların ve verilerin ortak bir ortamda birleştirilmeleri pahalı ve riskli bir işlemdi. Uygulamalar arasında iletişim kurmak isteyen şirketler genellikle ve doğal olarak farklı yazılım sistemlerine sahiptiler ve yine genellikle bu sistemler farklı donanımlar üzerinde

çalışıyorlardı. Kendi dar ortamlarının dışındaki yazılım paketleriyle iletişim kurmak için bir protokolleri yoktu.

Yukarıda bahsedilen bütünleştirme gereksinimi bir kurumun kendi içindeki sistemler arasında da olabilir. Şekil 1.1 bir kurumun bilgi teknolojileri ortamını ve kendi içindeki sistemler ve dış sistemlerle arasındaki muhtemel bütünleştirme gereksinimlerini gösterir.



Şekil 1.1 – Bir kurumun bilgi teknolojileri ortamı
(Stearns (2002))

Böyle bir senaryoda kurumların;

- Sistemlerini bütünleştirme işleminin yapılabilirliğini belirlemek,
- Bütünleştirme yaklaşımını tasarlamak,
- Ve son olarak bütünleştirmeyi sağlamak için gerekli prosedürleri gerçekleştirmek ve geliştirmek

Süreçlerini tamamlamaları gerekiyordu. Hatta bazı zamanlarda bütünleştirme işleminin yapılabilirliğinin belirlenmesi aşaması, bütünleştirme işleminin ekonomik olmadığı, ya da söz konusu sistemlerin bütünleştirilmesinin mümkün olmadığı sonuçlarını üretebiliyordu. Bütünleştirme işlemi devam etse bile tamamlanması yıllar sürebiliyordu. Genellikle başarı garantisi yoktu. Projeler, maliyetin artmasından ya da karşılaşılan önemli zorlukların önceden belirlenememiş olmasından dolayı sıklıkla sonlandırılıyordu. Başarılı bir şekilde tamamlandıklarında da, üretilen çözüm yanında birçok problemi de getirebiliyordu.

Yazılım bütünleştirmesi bu probleme farklı bir yaklaşım getiriyor. Yazılım bütünleştirmesi, uygulamaların ve veri kaynaklarının iletişim kurabilmeleri için standart bir yöntembilim ve bir yaklaşım tanımlar. Uygulamalar bu standartlara uyarak, diğer uygulamalarla kolay bir şekilde iletişim kurarlar. İletişim kuran uygulamaların parçaları (Örneğin: kullanılan ilişkisel veritabanı sistemi) değişebilir fakat kullanılan ortak yöntembilim sayesinde bu değişim iletişimin kesilmesine sebep olmaz.

1.2. Yazılım Bütünleştirmesi Çözümleri

Bu bölümde Web servislerinden önce geliştirilmiş olan yazılım bütünleştirme çözümlerini ele alacağız.

Yazılım bütünleştirmesi için ilk standartlaştırma çözümü Microsoft tarafından geliştirilen DCOM teknolojisiydi. Daha özel bir ifadeyle, DCOM, bileşen adı verilen ve belirli uygulama programı ara yüzlerinin kullanımıyla aynı türden diğer bileşenlere bağlanabilen küçük kod parçalarının geliştirmesini sağlıyordu. Windows kayıt birimi, DCOM sunucuları için merkezi havuz görevini görüyordu. İstemci uygulamaları bu kayıt birimini sorgulayarak sunucunun konumunu bulmak kullanabiliyorlardı. Bu teknoloji orijinal olarak Windows tabanlı platformlarla kısıtlandırılmıştı fakat son zamanlarda Unix tabanlı makinelerde de kullanılabilir hale gelmeye başladı. Geliştirilişindeki platform kısıtlaması, bu teknolojinin yazılım bütünleştirilmesi problemine çözüm olabilme yolundaki en büyük dezavantajıdır.

CORBA teknolojisi, Microsoft`un DCOM`u yayınlamasından çok az sonra OMG tarafından geliştirilmiştir. OMG`nin bir şirketler birliği olduğu düşünülürse, CORBA`ya Dünya`nın, Microsoft`un, DCOM`una karşılık verdiği cevap gözüyle bakılabilir. CORBA hareketi birkaç yüz organizasyon tarafından başlatıldı ve ortak bir CORBA tanımlaması

neticesini verdi. CORBA yazılım bütünleştirmesi alanında büyük oranda kabul görmüştür, çünkü heterojen yazılım platformlarının iletişimlerine izin verebilme konusunda DCOM`dan daha başarılı olmuştur.

Sun Microsystems`taki Java platformunun gelişimi, RPC`yi temel alan bir diğer yazılım bütünleştirme teknolojisi olan RMI`nın gelişimine sebep olmuştur. JAVA/RMI ile diğer teknolojiler arasındaki ana fark, bu teknolojide sunucu ve istemci taraflarının Java platformunda geliştirilmiş olmalarını gerektiriyor olmasıdır. JVM`in birçok platform için çalışabilen versiyonlarının mevcut olduğunu düşünerek, RMI`nın bu platform bağımlılığına rağmen bir yazılım bütünleştirmesi çözümü olduğunu kabul edebiliriz, fakat söz konusu platform kısıtlaması RMI`yı ideal bir yazılım bütünleştirmesi çözümü olmaktan uzaklaştırır (Rao, 1998) .

1.2.1, 1.2.2 ve 1.2.3 bölümlerinde bu üç teknolojinin yaklaşımlarını, mimarilerini ve bileşenlerini daha yakından inceleyeceğiz. Ardından 1.2.4 bölümünde bu üç teknolojinin yazılım bütünleştirmesi standardı olmayı neden başaramadıklarını anlatacağız.

1.2.1. CORBA

"Common Object Request Broker Architecture" ifadesinin kısaltması olan CORBA, OMG (Object Managment Group) tarafından, heterojen dağıtık ortamlardaki uygulamaların, nerede konumlandırıldıklarından bağımsız olarak birbirleriyle iletişim kurabilmelerini sağlamak amacıyla geliştirilmiş bir standarttır (http://www.omg.org/gettingstarted/corbafaq.htm).

OMG dağıtık nesnelerin birbirleriyle iletişim kurabilmeleri için ortak bir yol geliştiren üreticilerin oluşturduğu bir gruptur. Grup 1998 yılında yirmi ticari üretici (bunlardan önde gelenleri: IBM, BNR Europe Ltd., Expersoft Corp., ICL plc, Ione Technologies Ltd., DEC, Hewlett-Packard, HyperDesk Corp., NCR, Novell USG, Object Design Inc. ve SunSoft) tarafından kurulmuştur. Oluşum şeklinden dolayı, amacı uygulama nesnelerinin birbirleriyle iletişimlerini sağlamak için, heterojen donanım platformları ve işletim sistemleri arasında ortak bir mimari çatı sağlamak olan 600`den fazla üyesi vardır.

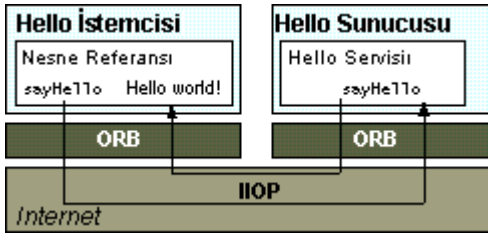
CORBA uygulamaları dağıtık nesnelerden meydana gelir. Nesne işlevselliği ve veriyi bir araya getiren ve genellikle gerçek dünyadaki bir varlığı tasvir eden özerk bir

yazılım birimidir. Bir nesnenin birçok örneği olabilir; örnek olarak bir e-ticaret uygulamasının birçok alışveriş sepeti nesnesi örneği olabilir. Bunların hepsi işlevsel olarak eşittir fakat hepsi farklı bir müşteriye ait olabilir ve ait olduğu müşterinin seçtiği ürünlerin farklılığına göre farklı verilere sahip olabilir.

CORBA mimarisindeki detaylara geçmeden 2 ana bileşenin tanımını yapalım:

- **IDL (Interface Definition Language)**: Ara yüz tanımlama dilidir. Bir yazılım bileşeni ara yüzünü tarif etmek için OMG tarafından geliştirilmiş basit bir sözdizimi yapısıdır (http://www.omg.org/gettingstarted/corbafaq.htm)..
- **ORB (Object Request Broker)**: Ağ üzerinden, bir programdan başka bir bilgisayardaki programa çağrı göndermelerini sağlayan yazılımdır (http://www.omg.org/gettingstarted/corbafaq.htm)..

Dağıtık nesneler arasındaki ilişkinin iki tarafı vardır: sunucu tarafı ve istemci tarafı. Sunucu uzaktan erişim için gerçek nesne için bir ara yüz sağlar ve istemci bir ara yüzü uzaktan çağırır. Bu ilişkiler bir RMI ve CORBA gibi birçok dağıtık nesne standardı için ortaktır. Bu anlatımdaki istemci ve sunucu kavramları uygulama seviyesinde değil nesne seviyesindedir, bir uygulama bir nesne için sunuculuk görevi görürken bir başka nesne için istemci rolü oynayabilir. Şekil 1.2 klasik bir "Hello World" uygulamasında, tek metotlu dağıtık bir nesnenin bir CORBA istemcisi ve sunucu arasında nasıl paylaşıldığını gösteriyor.

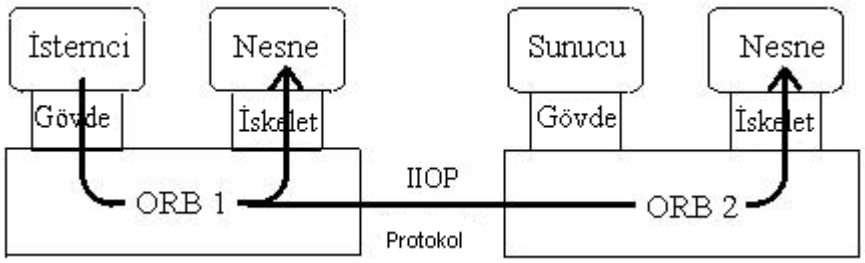


Şekil 1.2 – Bir CORBA sunucusu ve istemcisi tarafından paylaşılan tek metotlu bir dağıtık nesne (http://java.sun.com/docs/books/tutorial/idl/intro/corba.html)

İstemci tarafında, uygulama uzaktaki nesne için bir referans barındırır. Nesne referansının uzaktan erişilmek istenen metoda karşılık gelen bir metot gövdesi vardır. Bu gövde ORB`a bağlıdır. Böylece bu metot gövdesini çağrıldığında ORB`un isteğini sunucuya ileten bağlantı olanakları çağrılmış olur.

Sunucu tarafında, uzaktan yapılan metot çağırısını yerel nesne üzerindeki bir metot çağırısına dönüştürmek amacıyla iskelet kodu kullanır. İskelet çağrıyı ve parametreleri yereldeki metoda ve parametrelere, yereldeki uygulamaya bağlı olarak tercüme eder ve istenen metodu çalıştırır. Metodun çalışması tamamlandığında iskelet kodu sonuçları veya hataları istemcinin kullanabileceği biçime dönüştürür ve ORB`u kullanarak istemciye gönderir.

Şekil 1.3`te bu akışı görebilirsiniz.



Şekil 1.3 – ORB`ları kullanarak gerçekleştirilen bütünleştirme (http://www.omg.org/gettingstarted/corbafaq.htm)

ORB`lar arasındaki bağlantı ortak bir protokol olan IIOP –Internet Inter-ORB Protocol- kullanılarak yapılır. IIOP TCP/IP Internet protokolü standartlarına dayanan, CORBA uyumlu ORB`ların ileriye ve geriye veri göndermelerini sağlayan bir protokoldür. CORBA ve IDL gibi IIOP`un da standartları OMG tarafından tanımlanmıştır.

1.2.2. COM/DCOM

COM (Component Object Model) farklı zamanlarda, farklı platformları, araçları ve programlama dillerini kullanan farklı üreticiler tarafından yazılım bileşenleri geliştirilebilmesini mümkün kılmak için Microsoft tarafından geliştirilmiş bir nesneye

yönelik programlama modelidir (Roy ve Ewald (2001)). COM bileşenleri geliştirildiklerinde hedef sistemde kolayca çalıştırılabilirler. COM`un geliştirilmesinin ve çalıştırılmasının nasıl gerçekleştiğini anlamak için, ara yüzler, sınıflar ve sunucular, nesne yaşam çevrimi, ikili beraber işleyebilirlik ve konumun transparanlığı gibi bazı önemli COM kavramlarını anlamak önemlidir.

- **Ara yüzler**: Bir COM ara yüzü bir yazılım bileşeninin davranışlarını ve kabiliyetlerini bir metotlar ve özellik kümesiyle tanımlar. Bir ara yüz, kendisini destekleyen nesnelere ait tutarlı anlamlar temin eden bir sözleşme gibidir. Her bir COM nesnesi, aynı anda birden fazla ara yüzü destekleyebilir ancak en azından bir tane (IUnknown adındaki) ara yüzü desteklemelidir.

Bileşen tasarımcıları ara yüzleri MIDL`(Microsoft`s Interface Definition Languaage) yi kullanarak tanımlarlar. Microsoft, ara yüz tanımlamasından vekil (proxy) ve çatı kodlarını üreten bir MIDL derleyicisi sağlar. Üretilen vekil kodu, ara yüzü destekleyen nesneler için, istemci tarafında kullanılmak üzere bir uygulama programcısı ara yüzü (API) sağlar. Çatı nesneleri gelen istemci isteklerini işler ve bunları sunucudaki ilgili nesneye iletir. İç tarafta, vekil ve çatı kodları istekleri ve cevapları takas etmek için ilgili çalışma zamanı kütüphanelerini kullanarak iletişim kurarlar. COM çalışma zamanı yazılımı bu fazladan işi nesneler aynı süreçte çalıştırıldığı durumlarda gerçekleştirmez.

Bileşen tasarımcıları, isim çakışmalarının sebep olabileceği herhangi bir belirsizliği ortadan kaldırmak için, her bir ara yüze evrensel tekil tanımlayıcı (Universally Unique Identifier=UUID) atar. Bu tanımlayıcıya ara yüz tanımlayıcısı (Interface Identifier=IID) denir ve aynı zamanda COM`un ara yüz versiyonlama modelinin temel taşıdır.

Her bir COM nesnesi en azından standart IUnknown ara yüzünü desteklemelidir. IUnkown, nesne yaşam döngüsünü yönetmek ve nesne tarafından desteklenen ara yüzün evrimini sağlamak için temel yapı taşlarını sağlayan metotları tanımlar.

IUnknown`un QueryInterface metodu, istemciler tarafından, bir IID ile ifade edilen belirli bir ara yüzün bir nesne tarafından desteklendiğini belirlemek için kullanılır. Bir nesnenin desteklediği ara yüzler zamanla değişebilir (bir nesne yeni bir ara yüzü destekleyebilir) ya da desteklediği bir ara yüzün farklı bir IID`ye sahip olan yeni bir versiyonunu destekleyebilir. Mevcut istemciler yeniden derlenmeden bir ara yüzün eski

versiyonunu kullanmaya devam edebilirler ve yeni istemciler ara yüzün son versiyonunu sorgulayıp onun sağladığı avantajlardan faydalanabilirler.

QueryInterface metodu ara yüz işaretçisi denen bir işaretçi döndürür. Ara yüz işaretçisi bellekte, COM`un ikili beraber işleyebilirlik standardı tarafından tanımlanan bir veri yapısını gösterir. Bu standart, istemci ve sunucu programlarının geliştirildikleri ortamların farklığından bağımsız olarak, ara yüz fonksiyonlarının nasıl çağırılmaları gerektiğini belirler.

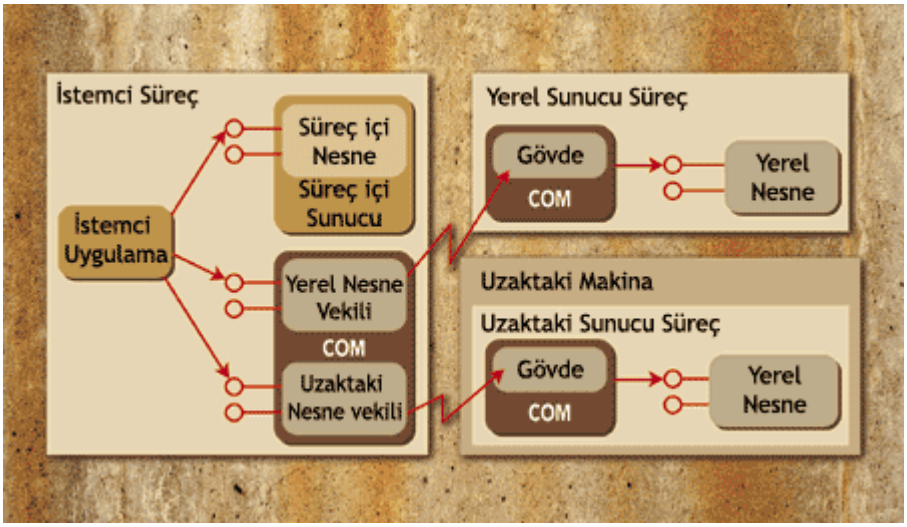
- **Sınıflar ve Sunucular**: Ara yüzler tek başlarına tam bir COM uygulamasını gerçekleştirmek için yeterli değillerdir. Bir COM sınıfı kaynak kodun gövdesidir, bir veya birden fazla COM ara yüzünün gerçekleştirimidir (implementation). Desteklediği her bir ara yüz fonksiyonu için, desteklenen herhangi bir dilde gerçek işlevsellik sağlar.

Her bir ara yüzün tekil bir IID(=Interface Indetifier)`sinin olduğu gibi her bir sınıf da tekil olarak adreslenebilecekleri bir CLSID`ye sahiptir. Bir istemci uygulaması, bir bileşenle iletişim kurmak için, en az bir CLSID`yi ve o CLSID ile adreslenen sınıf tarafından desteklenen bir IID`yi bilmek zorundadır. Bu bilgiyi kullanarak bir istemci bir COM`dan bir nesne yaratmasını ve ilgili bir ara yüz işaretçisini döndürmesini isteyebilir.

Bir veya birden fazla COM sınıfı belirli tekniklerden birini kullanarak bir sunucuda paketlenebilir. Bir COM sunucusu, bir istemci sürecine, sunucudaki bir sınıfa erişim talebinde bulunmasıyla yüklenen bir dinamik bağlanabilir kütüphane (dynamic link library) olarak paketlenebilir. Buna süreç içi sunucusu (in-process server) denir. Bir ActiveX kontrolü bir süreç içi sunucudur. Bir COM sunucusu ayrı bir çalıştırılabilir program olarak da paketlenebilir. Bu tip sunucular istemciyle aynı makine üzerinde çalışabileceği gibi, DCOM ile erişilebilen uzaktaki bir makinede de çalışabilir. Bu sunuculara süreç dışı sunucu (out-of process server) denir.

İstemci uygulamaları COM nesneleriyle ara yüz işaretçilerini kullanarak iletişim kurarlar. COM tarafından sağlanan kapsülleme, bir istemcinin COM nesnesinin gerçekleştirimindeki herhangi bir detaydan bağımsız olmasını sağlar. Süreç içi sunularında, bir istemci tarafından ara yüz işaretçisini kullanarak yapılan bir çağrı, direkt olarak istemci sürecinde yaratılan nesneye gider. Süreç dışı sunucuya gönderilen istekler ise ilk olarak, uzaktan prosedür çağrı işlemini yapmakla sorumlu bir süreç içi vekile gider. Süreç dışı

sunucuda, bir gövde sınıfından gelen her bir çağrıyı alır ve ilgili COM nesnesine gönderir. Şekil 1.4 çeşitli farklı yollarla paketlenmiş bir COM nesnesiyle iletişim kuran bir COM istemcisini gösteriyor.

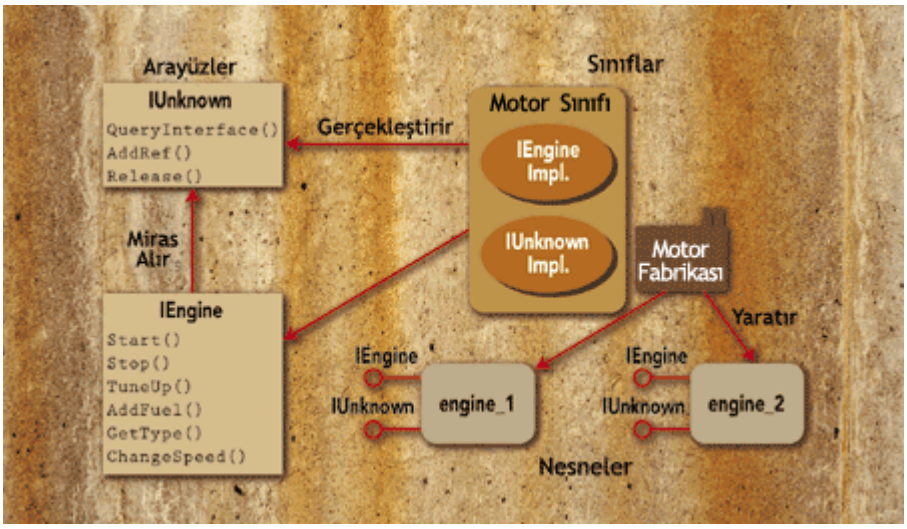


Şekil 1.4- Çeşitli farklı yollarla paketlenmiş bir COM nesnesiyle iletişim kuran bir COM istemcisi
(Roy ve Ewald (2001))

- **Nesne Yaşam Çevrimi:** Şimdi bir istemcinin bir COM nesnesinin yaratılmasına nasıl sebep olduğunu inceleyelim. Her bir COM nesnesi COM sınıfının bir örneğidir. Her bir COM sınıfı, işi COM sınıflarından örnekler üretmek olan, COM fabrikası adı verilen bir COM sınıfıyla birleşmiştir. COM`un fabrika mekanizması Şekil 1.5`de gösterilmiştir. Fabrika tipik olarak, COM tarafından tanımlanmış bir IClassFactory adı verilen bir ara yüzü destekler. Verilmiş bir CLSID ve ilişkili COM sunucusuna ait tanımlama ile COM, CLSID için bir fabrika konumlandırabilir.

Şekil 1.5- COM`un fabrika mekanizması (Roy ve Ewald (2001))

Bir istemci uygulaması, bir COM sınıfı için, bu sınıfın bir örneğini yaratmak ve sonuçta oluşan COM nesnesini gösteren bir ara yüz işaretçisi elde etmek için, standart bir yol kullanarak fabrika ile iletişim kurar. Şekil 1.6 COM ara yüzlerinin, sınıflarının ve nesnelerinin nasıl birbirlerine bağlandığını gösterir.



Şekil 1.6- COM ara yüzleri, sınıfları ve nesnelerini arasındaki ilişki (Roy ve Ewald (2001))

COM nesnesi yaratıldığında, COM bir nesneye kaç tane referansın mevcut olduğunu ve o nesnenin ne zaman yok edilebileceğini takip etmek için bir protokol belirler. COM nesneleri, IUnknown ara yüzünde tanımlanmış AddRef ve Release metotları tarafından işlenen bir referans sayacını muhafaza eder. COM sınıfı, örneklerinin anlık

kullanım adetlerini takip edebilmek için bu metotların içini doldurmalıdır. Programcının isteğine göre, referans takibi işlemi, tüm nesne için tek bir sayacın kullanılmasıyla, ya da nesnenin desteklediği her bir ara yüz için ayrı bir sayacın kullanılmasıyla yapılabilir.

Referans sayacı sıfır olduğunda, bu nesneyi referans olarak gösteren herhangi bir istemcinin kalmadığı ve nesnenin yok edilebileceği anlamına gelir. QueryInterface dahil ara yüzü işaretçisi döndüren metotlar, bir nesneyi göstermek için AddRef metodunu çağırmak zorundadırlar. Ara yüzleri kullanan istemciler, ara yüze ait işaretçiye erişmeleri sonra erdiğinde Release metodunu çağırmak zorundadırlar.

COM sadece yerel yazılımların birbirleriyle iletişim kurabilmeleri için geliştirilmiş bir teknolojidir. Microsoft farklı yerlerdeki bilgisayarlarda çalışan yazılımların birbirleriyle iletişim kurabilmeleri için DCOM (Distrubuted COM)`u geliştirmiştir. DCOM, DCE RPC alt yapısını kullanarak COM`un geliştirilmiş halidir.

RPC (remote procedure call) uzaktaki bir metodu çalıştırmaya yarayan bir mekanizmadır. İstemci program bu mekanizmayı kullanarak; TCP/IP protokolü ile çağırmak istediği fonksiyonu ve parametreleri sunucuya bildirir. Sunucuda fonksiyon icra edilir ve dönüş değeri yine TCP/IP protokolü üzerinden istemci uygulamaya verilir. Bu mimari 1976 yılında geliştirilmiştir ve sonraki bütün dağıtık programlama mimarileri için bir temel oluşturur. Uzaktan çağırılacak olan prosedürün gövdesinin istemcide konumlandırılması, sunucuda, gelen isteği karşılayan ve gerçek çalıştırma işlemini başlatacak olan bir iskelet yapının konumlandırılması gibi yaklaşımlar bu mimari ile ortaya atılmıştır.

DCE RPC ise kabaca, RPC`nin nesnelerle çalışan versiyonudur. Sunucudaki prosedüre parametre olarak bir nesnenin gönderilmesi ve dönüş değeri olarak bir nesnenin döndürülmesini sağlar.

1.2.3. JAVA RMI

JAVA RMI (Remote Metot Invocation) uzaktan prosedür çağrısı yapabilmeye yarayan bir Java uygulama programı ara yüzüdür (http://en.wikipedia.org/).

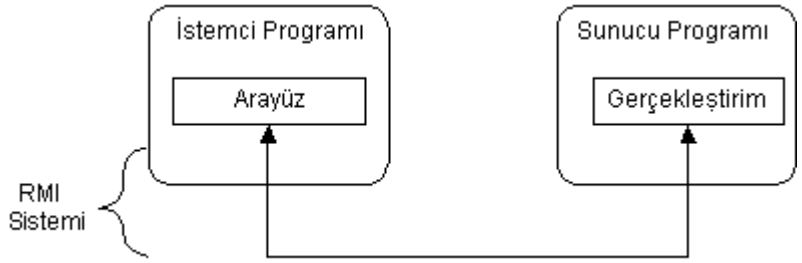
RMI uygulamaları genellikle iki programdan meydana gelir: bir istemci ve bir sunucu. Tipik bir sunucu uygulaması uzaktan erişim için nesneler ve bu nesnelere erişim için referans noktaları oluşturur ve bir istemci uygulamasının bu uzaktan erişim nesnelerinin metotlarını çağırmaları için bekler. Tipik bir istemci uygulamasıysa, sunucudaki bir veya daha fazla uzaktan erişim nesnesini gösteren bir referans alır ve bu nesnelerin metotları için çağrılar gönderir. RMI, sunucu ve istemcinin ileri ve geri veri iletişimi yapabilmeleri için gerekli olan mekanizmayı sağlar.

RMI mimarisinin tasarım amacı, Java programlama diliyle ve yerel nesne modeliyle doğal bir şekilde bütünleşen bir Java dağıtık nesne modeli yaratmaktır.

- **Ara yüzler:** RMI mimarisi önemli bir prensibe dayanır: Bir davranışın tanımlaması ve o davranışın gerçekleştirimi birbirlerinden farklı kavramlardır. RMI, bir davranışı tanımlayan kodun ve gerçekleştiren kodun farklılıklarını sağlar ve farklı JVM (Java Virtual Machine)`lerde çalışmalarına olanak sağlar.

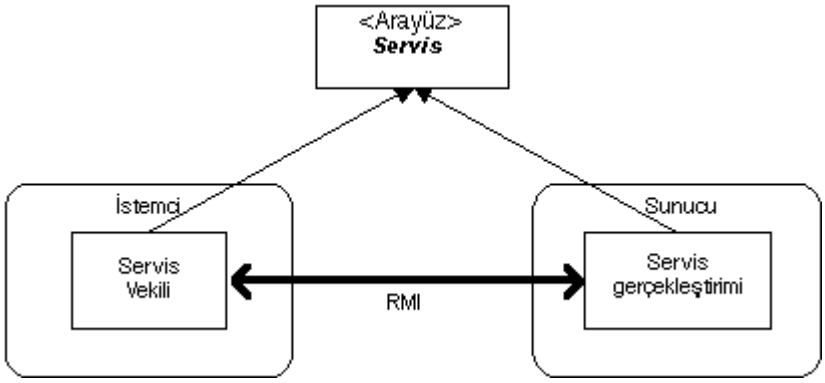
Bu yaklaşım, istemcilerin sadece davranışların tanımlamalarıyla ilgilendiği ve sunucuların bu davranışların gerçekleştiriminden sorumlu olduğu dağıtık sistemlerde de bir gereksinimdir.

RMI`da uzaktaki bir servisin tanımını bir Java ara yüzünün kodlanmasıyla yapılır. Uzaktaki servisin gerçekleştirimi bir sınıf içinde yapılır. Bu yüzden RMI mimarisindeki temel yaklaşım; ara yüz davranışının tanımlaması ve, nesne davranışı gerçekleştirimidir. Şekil 1.7 bu ayrımı gösterir.



Şekil 1.7- Davranışın tanımı ve gerçekleştirimi (Seshadri (2000))

Bir Java ara yüzü çalıştırılabilir kodu barındırmaz. RMI bir ara yüzü gerçekleştiren iki tane sınıf sağlar. Birinci sınıf davranışın gerçekleştirimidir ve sunucuda çalışır. İkinci sınıf uzaktaki servisin istemcideki vekilliğini yapar ve istemcide çalışır. Bu durum Şekil 1.8`de gösterilmiştir.



Şekil 1.8- Servisin gerçekleştirimi ve vekili (Seshadri (2000))

Bir istemci programı vekil nesne üzerinde bir metot çağırısı yapar, RMI bu isteği uzaktaki JVM`e gönderir ve gerçekleştirime iletir. Gerçekleştirim tarafından sağlanan dönüş değeri varsa bu değer geriye vekile ve sonra da istemciye gönderilir.

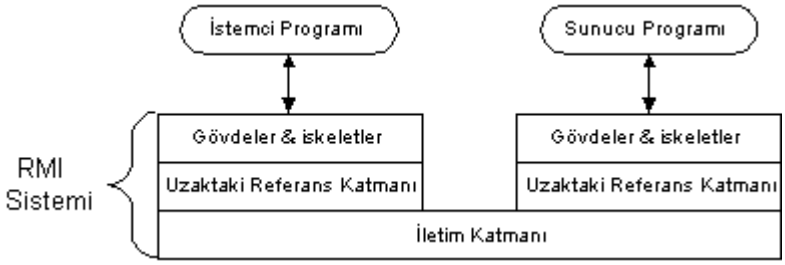
RMI mimarisinin genel yapısını anlattıktan sonra bu mimarinin gerçekleştirimini inceleyelim.

RMI gerçekleştirimi üç soyutlama katmanından oluşur. Birincisi, programcının direkt olarak iletişim kurduğu gövde ve iskelet katmanıdır (Stub and Skeleton Layer). Bu

katman istemci tarafından ara yüz referans değişkenine yapılan metot çağrılarını yakalar ve bu çağrıları uzaktaki RMI servisine yönlendirir.

Sonraki katman Uzaktaki Referans Katmanıdır (Remote Reference Layer). Bu katman, istemci tarafından uzaktaki servis nesnesine yapılan referansların nasıl yönetileceğini ve yorumlanacağını anlar.

Taşıma katmanı (Transport Layer) bir ağdaki makineler arasındaki TCP/IP bağlantılarını temel alır. Temel bağlanabilirliği sağlar. Şekil 1.9`da bu katmanlı yapı gösterilmiştir.

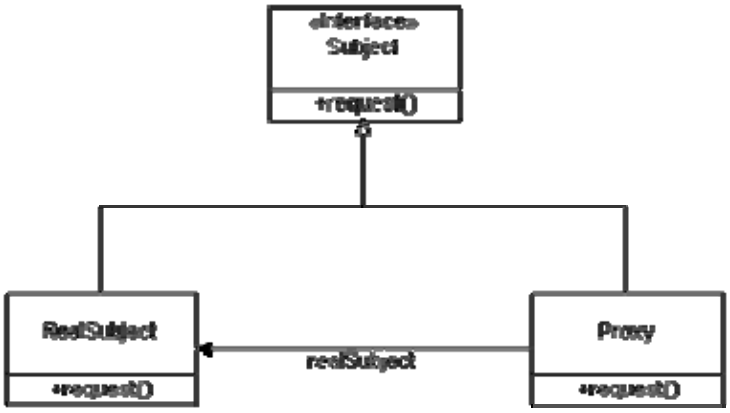


Şekil 1.9- JAVA RMI`nın katmanlı yapısı (Seshadri (2000))

Bu katmanlı yapının kullanılmasıyla, her bir katman, sistemin geri kalan kısmını etkilemeden, değiştirilebilir ya da geliştirilebilir. Örnek olarak taşıma katmanı, üstteki katmanları etkilemeden UDP/IP katmanıyla değiştirilebilir.

Şimdi bu katmanları inceleyelim:

- **Gövde ve İskelet Katmanı:** Gövde ve iskelet katmanı programcının direkt olarak iletişim kurduğu katmandır. RMI bu katmanda Vekil tasarım kalıbını (Proxy Design Pattern) kullanır. Vekil tasarım kalıbında, bir nesne başka bir içerik (vekil) tarafından temsil edilen ayrı bir içeriktir. Vekil katılımcı nesneler arasındaki metot çağrılarını nasıl ileteceğini bilir. Şekil 1.10 Vekil kalıbını gösteriyor.



Şekil 1.10- Vekil Kalıbı (Seshadri (2000))

RMI`nın Vekil kalıbının kullanımında, gövde sınıfı Proxy rolünü oynar ve uzaktaki servisin gerçekleştirim sınıfı da RealSubject rolünü oynar.

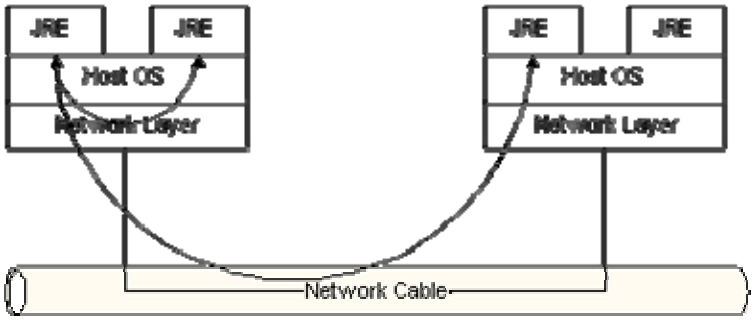
İskelet, RMI`nın kullanması için üretilmiş (JDK tarafından sağlanmış RMI derleyicisine[rmic] gerçekleştirim sınıfının girdi olarak verilmesiyle üretilir) yardımcı bir sınıftır. İskelet, RMI bağlantısını kullanarak, gövde ile nasıl iletişim kurulacağını anlar. İskelet, gövdeyle gerçekleşen diyalogu taşır; bağlantı üzerinden metot çağırısı ile gelen parametreleri okur, uzaktaki servis gerçekleştirim nesnesi üzerinde metot çağrısı işlemini gerçekleştirir, dönüş değerini alır ve son olarak dönüş değerini gövdeye geri gönderir.

- **Uzaktaki Referans Katmanı:** Uzaktaki referans katmanı, RMI bağlantısının çağrı mekanizmasını tanımlar ve destekler. Bu katman, uzaktaki servis gerçekleştirim nesnesini bağlanmayı sağlayan RemoteRef nesnesini sağlar.

Gövde nesnesi, metot çağrısını iletmek için RemoteRef nesnesinin invoke() metodunu kullanır.

- **Taşıma Katmanı:** Taşıma katmanı JVM`ler arasındaki bağlantıyı sağlar. Tüm bağlantılar TCP/IP`yi kullanan akış tabanlı (stram based) ağ bağlantılarıdır.

İki JVM de aynı fiziksel bilgisayarda çalıştırılıyor olsalar da sunucu bilgisayarlarının TCP/IP ağ protokol yığıtını kullanarak iletişim kurarlar. Şekil 1.11`de JVM`ler arasındaki bağlantı gösteriliyor.



Şekil 1.11- RMI`da JVM`lerin bağlanması (Seshadri (2000))

Bilindiği gibi, TCP/IP iki makine arasında, iki ucun IP adresleri ve port numaralarını kullanarak, kalıcı ve akış tabanlı bir bağlantı sağlar. Genel olarak IP adresi yerine DNS ismi kullanılır. RMI`nın şu anki sürümünde, bütün makineler arası bağlantılarda TCP/IP kullanılır.

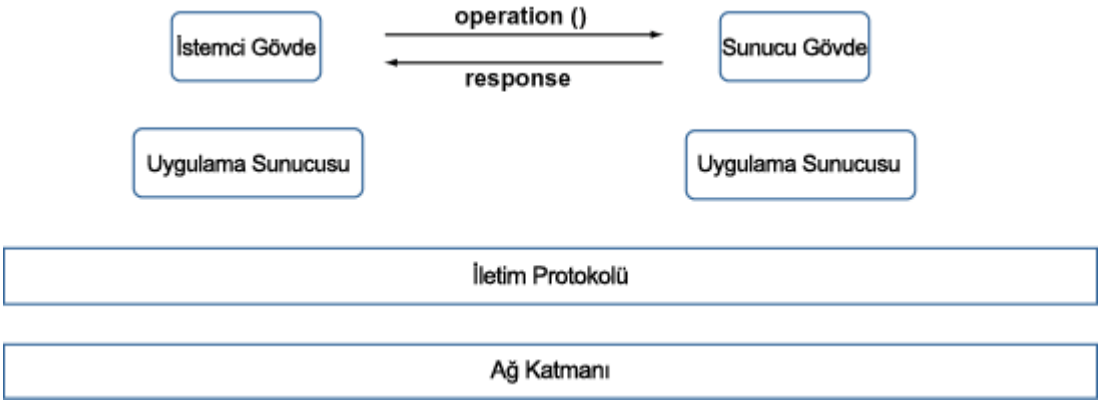
RMI, TCP/IP`nin üzerinde Java Remote Protocol Metot (JRMP) denen bir protokol kullanır.

Sun ve IBM RMI`nın bir sonraki sürümü olan RMI-IIOP`u geliştirmek için birlikte çalışmışlardır. RMI-IIOP`taki ilginç nokta, istemci ve sunun arasındaki taşıma katmanında JRMP`nin kullanılması yerine OMG tarafından geliştirilmiş 1.2.1`de anlatılan CORBA mimarisinde kullanılan IIOP`nin kullanılmasıdır.

1.2.4. Yazılım Bütünleşmesi Çözümlerinde Gelişmeler

Şimdi CORBA, DCOM ve Java RMI çözümlerinin ortak özelliklerinden, yazılım bütünleştirmesi konusundaki eksik yönlerinden ve beklenen mimarinin özelliklerinden bahsedelim.

Üç teknolojide de istemci süreci ve nesne sunucusu arasındaki etkileşim, nesneye yönelik RPC stili iletişimi temel alıyor. Şekil 1.12`de bu üç mimarideki iletişimin temel aldığı RPC mimarisini görebilirsiniz.



Şekil 1.12- Temel RPC mimarisi (Gisolfi (2001))

İstemci, uzaktaki bir fonksiyonu çağırmak için istemcideki gövde üzerinde bir çağrı gerçekleştirir. Gövde çağrı parametrelerini bir istek mesajı halinde paketler ve mesajın sunucuya iletilmesi için bir taşıma protokolünü çalıştırır. Sunucu tarafında, taşıma protokolü, mesajı, sunucudaki gövdeye iletir. Sunucudaki gövde mesajı açar ve asıl nesne üzerindeki metot çağrı işlemini gerçekleştirir.

Bu teknolojiler farklı platformlar için gerçekleştirilmiş olsalar da, gerçek odur ki, bu protokolleri temel alan herhangi bir çözüm, tek bir üreticinin gerçekleştirimine bağlı olacaktır. Bir programcı DCOM`u kullanarak bir yazılım bütünleştirmesi çözümü uygularsa, dağıtık uygulamadaki bütün düğümler Windows üzerinde çalışmak zorunda olacaktır. CORBA`nın kullanılması durumundaysa, uygulama ortamındaki bütün düğümler, aynı ORB ürününü çalıştırmak zorunda olacaklardır. Sonradan, farklı ORB`ları kullanan uygulamaların beraber çalışmalarını sağlayan CORBA sürümleri çıkmıştır fakat bu sürümler, güvenlik ve işlem yönetimi (transaction management) gibi daha üst seviyeli işlemleri destekleyecektir. JAVA RMI ise zaten sadece JAVA diliyle yazılan programların birlikte çalışmalarını sağlamaya yarayan bir teknolojidir.

Bu mimarilerin kısıtlamaları mevcut iken, yaygın olarak kullanılabilecek, gerçek bir dağıtık programlama modeline şiddetle ihtiyaç duyulmaktadır. Internet`in yaygınlaşmasıyla, şirketler kurumlarının dışındaki dağıtık uygulamalara daha şiddetli bir ihtiyaç duymaktadırlar. Başarılı bir dağıtık programlama modelinden beklenenler nelerdir? Çözüm, üreticiden (vendor), platformdan ve kullanılan dilden bağımsız olmalıdır. Birlikte işleyebilirliği (interoperability) gerçekten sağlayabilmelidir. Buna ilaveten, programcılar

için kullanılması kolay bir protokol olmalıdır. Daha basit bir ifadeyle, ihtiyaç duyulan protokol açık Internet standartlarını (open Internet standarts) temel alan bir protokol olmalıdır.

2. SOA ve WEB Servisleri

SOA (Service Oriented Architecture=Servis Yönelimli Mimari) genellikle Web servisleri teknolojisiyle beraber anılan bir mimaridir. Yazılım geliştirme dünyasında seneler sonunda elde edilen tecrübelere dayanan yaklaşımlar içerdiği gibi yazılım bütünleştirmeyle ilgili, mevcut teknolojilerin eksikliklerine de çözüm getirmiştir.

Bu kısımda SOA ve Web servislerini detaylı bir şekilde inceledikten sonra, Web servisleri ile CORBA arasındaki farklılıkları ortaya koyarak, SOA mimarisinin getirdiği yenilikleri daha somut bir şekilde ele almış olacağız.

Öncelikle SOA`nın çıkış sebebini, prensiplerini, bu prensiplerin hangi problemlere çözüm getirdiğini anlatarak SOA`nın tanımını yapalım.

2.1 Servis Yönelimli Mimari (SOA) Nedir?

"İşler mümkün olduğunca basit bir şekilde yapılmalıdır ama bundan daha basit yapılmamalıdır". Einstein bu sözü onlarca sene önce söylemiştir ve bu söz bugün hala üstün yetenekli yazılım sistemlerinin inşa edilmesiyle ilgili problemler için geçerliliğini korumaktadır. Yazılım sektöründe geliştirilen büyük projelerin bir kısmı, yerine getirmekle yükümlü oldukları işleri gerçekleştirmek için aşırı basit iken, diğer kısmı aşırı karmaşık bir şekilde inşa edilmiştir ve inşasının ve/veya sonradan yönetilmesinin maliyeti çok fazla yüksek boyutlara ulaşabilmektir. Bunun yanında 1. kısımda anlatılan farklı sistemlerin bütünleştirmesi konusu da önemlidir. Bu gerçeklerden yola çıkarak Einstein`ın da işaret ettiği, doğru seviyede basitliğe ulaşabilmek önemli ve çözülmesi zor bir problemdir (He (2003)).

En başta SOA`nın en önemli çıkış noktalarından biri olan hafif bağlaşımı (loose coupling) anlatmakta fayda vardır.

Bağlaşım (coupling) birbiriyle iletişim kuran sistemler arasındaki bağımlılıktır. Bu bağımlılık gerçek bağımlılık (real dependency) ve yapay bağımlılık (artificial dependency) şeklinde ikiye ayrılır:

1. Gerçek bağımlılık: Bir sistemin diğer sistemlerden faydalanarak kullandığı özellik veya servislerden meydana gelen bir kümedir. Gerçek bağımlık hep mevcuttur ve azaltılamaz (Haas ve Brown (2004)).

2. Yapay bağımlılık: Bir sistemin, diğer sistemler tarafından sağlanan servis veya özelliklerden faydalanabilmesi için uyum sağlaması gereken unsurlardır. Tipik yapay bağımlılık unsurları, dil bağımlılığı, platform bağımlılığı, API (uygulama programı ara yüzü) bağımlılığı vs.`dir. Yapay bağımlılık hep mevcuttur fakat kendisi ya da maliyeti azaltılabilir (Haas ve Brown (2004)).

Gerçek ve yapay bağımlılık kavramlarının daha iyi anlaşılması için yazılımdan bağımsız, daha somut bir örnek verelim. Amerika`da yaşayan insanlar, başka bir ülkeye gittiklerinde güç adaptörlerini de yanlarında götürmelidirler, götürmezlerse elektrikle çalışan cihazlarınızı çalıştıramazlar. Gerçek bağımlılık söz konusu cihazlarınızın elektriğe olan ihtiyacıdır, yapay bağımlılık ise, adaptörlerin fişinin gidilen ülkedeki prize uymasının gerekmesidir.

Hafif bağlaşım, yapay bağımlılığın veya maliyetinin en aza indirgendiği yapılanmayı tarif eden bir kavramdır (Haas ve Brown (2004)). Sistemler arasındaki yapay bağımlılık ideal yani olabileceği en az miktara indirgenirse, hafif bağlaşım kriteri sağlanmış olur. Bu açıdan bakıldığında, Einstein`ın işaret ettiği nokta tam olarak hafif bağlaşımdır diyebiliriz ve bu sözünü şu şekilde değiştirebiliriz: "yapay bağımlılıklar olabilecekleri en az seviyeye indirgenmelidir, fakat gerçek bağımlılıklar değiştirilmemelidir.".

Bu anlattıklarımızın üzerine SOA`nın tanımını yapabiliriz. SOA, birbirleriyle iletişim halinde olan yazılım sistemleri arasında hafif bağlaşımlı bir etkileşim kurmayı amaçlayan bir mimaridir. Servis, servis sağlayıcı tarafından, bir servis istemcisinin istediği son neticeye ulaşmak için yapılan işler birimidir. İstemci ve sağlayıcı uçları birer yazılım sistemidir.

Geniş kapsamlı alt projeler içeren büyük bir yazılım projesinde, alt projelerin gerçekleştiriminin de ana projede gerçekleştirilmesi, ana projenin ilerlemesini zorlaştırabilir hatta imkânsızlaştırabilir. SOA mimarisinde, alt projelerin gerçekleştirimleri tamamen farklı birer proje gibi kabul edilir ve bu projeler, sonradan başka projelerde kullanılabilecek birer servis gibi gerçekleştirilir. Fakat bunu yaparken yukarıda da bahsettiğimiz gibi birimler arasındaki yapay bağımlılıkların en aza indirgenmesini gerekli kılar.

SOA, iletişim halindeki yazılım birimleri arasındaki hafif bağlaşımı nasıl başarır? Bunu aşağıdaki iki mimari kısıtlamayı kullanarak yapar:

1- Bütün katılımcı yazılım birimlerinde basit birer ara yüz kümesi sağlanmalıdır. Ara yüzlerde sadece yapılan işe has, genel anlamda tasvirler içeren kodlar bulundurulur. Ara yüzler bütün servis sağlayıcıların ve istemciler tarafından elde edilebilir olmalıdır.

2- Sınırları genişletilebilir bir şema ile belirlenmiş bir tanımlayıcı mesaj, ara yüzler aracılığıyla iki tarafa iletilir. Bu mesajlar hiçbir (veya mümkün olduğunca az miktarda) sistem davranışı tarifi içermelidir. Kullanılan şema, mesajların kelime hazinesinin ve yapısının sınırlarını belirler. Bu şemanın genişletebilir olması servisin yeni bir sürümünün tanımlanması durumunda servisin kesintiye uğramadan yeni sürümün gerçekleştirmesini mümkün kılar.

Şimdi bir SOA gerçekleştirimi olan Web servislerini inceleyelim. Böylece SOA`yı daha somut bir şekilde anlatmış olacağız.

2.2 Web Servisleri

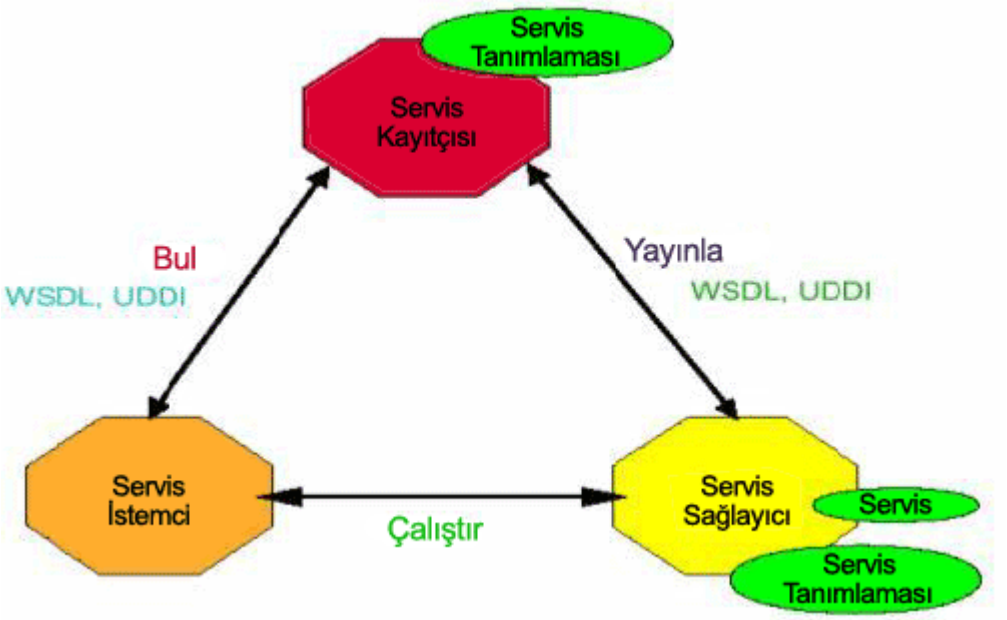
2.2.1. Web Servisleri Tanımı

Web servisleri teknolojisi W3C konsorsiyumu tarafından bir SOA gerçekleştirimi olarak ortaya atılmıştır. W3C Web servislerini şu şekilde tanımlıyor; Bir Web servisi, bir bilgisayar ağı üzerinde, birlikte işleyebilen, bilgisayardan bilgisayara etkileşimi sağlamak için tasarlanmış bir yazılım sistemidir (http://en.wikipedia.org/wiki/Web_services). Bilgisayar tarafından işlenebilen bir biçimde tanımlanmış bir ara yüzü vardır (WSDL). Web servisiyle etkileşimde bulunan diğer sistemler, tipik olarak HTTP protokolü ile diğer Web ile ilgili standartlarla birlikte XML formatında taşınan SOAP mesajlarını kullanarak Web sevisinin tanımıyla (WSDL) belirlenmiş bir şekilde etkileşimde bulunurlar.

Web servislerinin mimarisini anlatmaya başlamadan önce Web servislerinin kullanıldığı bir ortamı detaylı bir şekilde tarif edelim. Şekil 2.1`de böyle bir ortamdaki rolleri ve bu roller arasındaki etkileşimleri görüyoruz. Öncelikle rollerin tanımlarını yapalım:

- Servis sağlayıcı (service provider): Web servisi teknolojisiyle sunulan servisin barındırıldığı birim.

- Servis istemcisi (service requestor): Servis sağlayıcıda sunulan servisten faydalanmak isteyen birim.
- Servis kayıtçısı (service registry): Servis sağlayıcıda (örnekteki servis sağlayıcı ve diğer servis sağlayıcılarda) sunulan servislerin tanımlarının (WSDL) barındırıldığı birim.



Şekil 2.1 Web servisi modeli
(Kreger (2001))

Bu üç birim farklı konumlarda, farklı ağlarda olabilirler, Internet ile birbirlerine bağlı olabilirler. Tüm servis sağlayıcıları sağlamakta oldukları servislerin tanımlarını servis kayıt birimi aracılığıyla yayınlarlar (publish). Servis istemcisi faydalanmak istediği servisi servis kayıtçısına gönderir ve buradan ilgili servisin tanımını (WSDL) elde eder. Daha sonra istemci, elde ettiği tanımdan faydalanarak, servis sağlayıcıdaki servisi kullanmakla ilgili istek mesajını servis sağlayıcıya gönderir. İstenen servisle ilgili işlemler sağlayıcıda gerçekleştirildikten sonra, servisle ilgili dönüş değerleri istemciye geri gönderilir. Burada servis kayıtçısı rolü isteğe bağlıdır. Servis istemcisi servisin tanımına zaten sahip ise, bunu elde etmek için servis kayıtçısına ihtiyacı yoktur.

Şimdi Web servisi mimarisini, mimarideki bileşenleri detaylı bir şekilde inceleyeceğiz. Ardından yukarıda anlatılan modeli daha detaylı bir şekilde ala alarak Web servislerinin incelenmesi kısmını bitirmiş olacağız.

2.2.2. Web Servisleri Mimarisi

Web servisleri mimarisini bir kaç katmanda inceleyebiliriz. İlk olarak kavramsal bir Web servisi yığıtını ve bu yığıtın detaylarını inceleyeceğiz. Ardından, kullanılacak ağ protokolünün seçimindeki kriterleri tartışacağız. Ayrıca dağıtık programlamada XML tabanlı mesajlaşmayı gözden geçireceğiz. Temel XML mesajlaşmasını, servis tanım yığıtındaki servis tanımlamasına genişleteceğiz. Bunun ardından, servis yayınlama tekniklerini göz önünde bulundurarak, servis tanımlamasının Web servisleri mimarisindeki rolünü tartışacağız. Servis yayınlamasıyla ilgili olarak servis keşfinin rolünü tartışacağız. Son olarak, Web servislerini e-iş`te, gerçek dünyadaki yaşayabilirliğini sağlayan temel Web servisleri mimarisinin uzantılarını inceleyeceğiz.

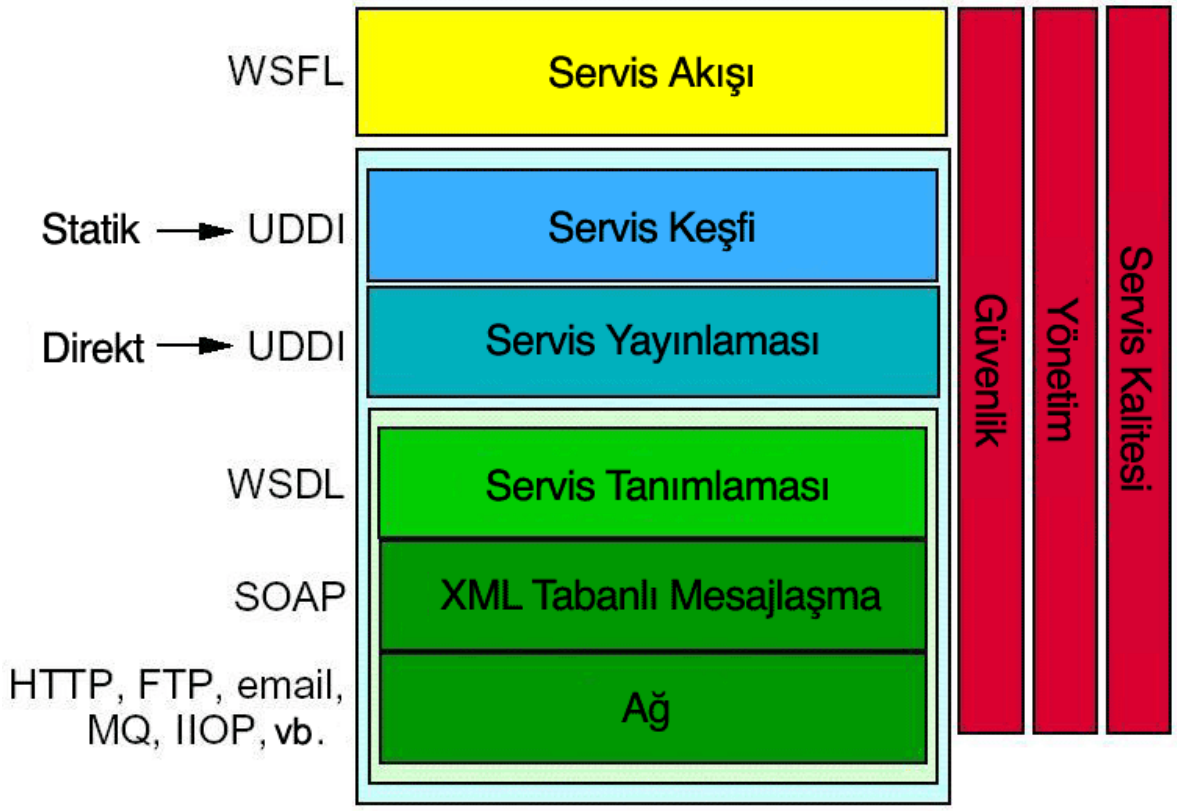
Web Servisleri Yığıtı:

Yayınlama, bulma ve çalıştırma işlemlerini birlikte çalışabilir (interoperable) bir tarzda gerçekleştirebilmek için, standartları her bir katmanda birleştiren bir Web servisleri yığıtı olmalıdır. Şekil 2.2`de kavramsal Web servisleri yığıtı gösteriliyor. Yukarıdaki bir katman, alttaki katmanlar tarafından sağlanan kabiliyetler üzerine kurulmuştur. Dikey katmanlar yığıtın her katmanında sağlanması gereken gereksinimleri gösterir.

Web servisi yığıtının esas dayanak noktası bilgisayar ağıdır. Web servislerinin bir istemci tarafından çağrılabilmeleri için ağ üzerinden erişilebiliyor olmaları gerekir. Internet üzerine kamuya açılmış Web servisleri ortak olarak kullanılan ağ protokollerini kullanırlar. HTTP protokolü, tüm ortamlardaki kullanılabilirliğinden dolayı, Internet üzerinde kullanılabilen Web servisleri için fiili bir standarttır. SMTP ve FTP dahil diğer Internet protokolleri de desteklenebilir.

Bir sonraki katman, XML tabanlı mesajlaşma katmanıdır. Mesajlaşma protokolünün temel şartı olarak XML`in kullanımını gösterir. Bir çok sebepten dolayı XML mesajlaşma protokolü olarak SOAP seçilmiştir. Bu sebeplerin bazılarını şu şekilde sıralayabiliriz:

- XML kullanarak uzaktan metot çağırma işlemi ve doküman merkezli mesajlaşma için standartlaştırılmış bir kaplama mekanizmasıdır
- Basittir; yük olarak sadece bir XML zarfı ile gerçekleştirilen bir HTTP POST metodundan ibarettir



Şekil 2.2 Kavramsal Web Servisi yığıtı (Kreger (2001))

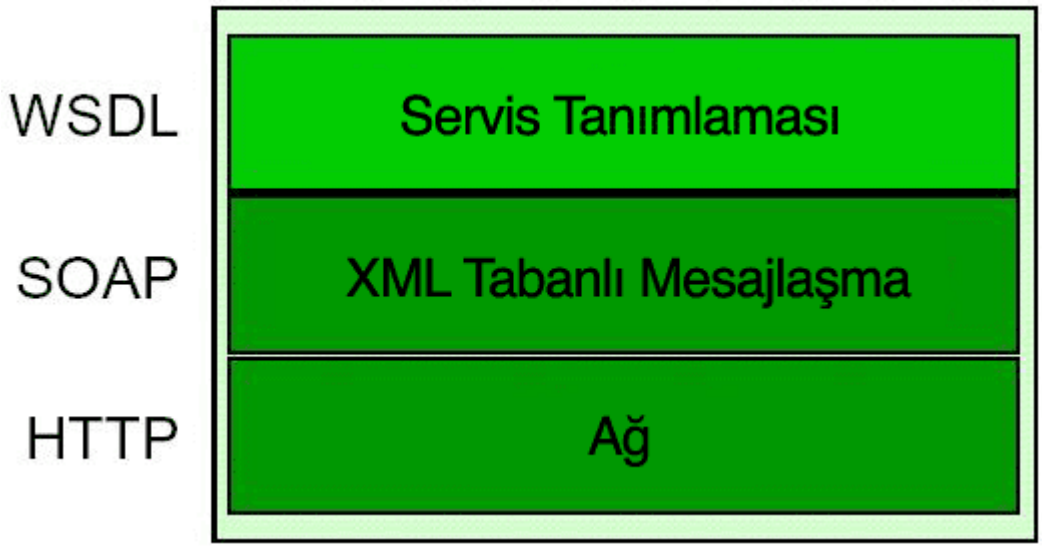
- HTTP POST üzerinde tercih edilir çünkü iletilen mesaja, SOAP başlıklarını ve işlemin ve fonksiyonun standart kodlamasını ekleyebilir

- SOAP mesajları, Web servisleri mimarisindeki, yayınlama, bulma ve çalıştırma işlemlerini destekler. "XML mesajlaşması tabanlı dağıtık programlama" kısmında bu katman daha detaylı anlatılacaktır.

Servis tanım katmanı, tanım dokümanları katmanıdır. İlk olarak, WSDL XML tabanlı servis tanımı için fiili bir standarttır. Yazılım bütünleştirmede kullanılan Web servislerini desteklemek için gerekli olan minimum servis tanımıdır. WSDL, ara yüzü ve servis etkileşimindeki mekaniği tanımlar. İş içeriğini, servis kalitesini ve servisten servise ilişkileri belirtmek için ilave tanımlar gereklidir. WSDL dokümanı, Web servislerine ait bu daha ileri seviyedeki varlıkların tanımlanması için diğer tanımlayıcı dokümanlarla tamamlanabilir. Örnek olarak iş içeriği, WSDL dokümanına ek olarak UDDI veri

yapılarının kullanımıyla tanımlanır. Servise ait akış ve tümleşimi bir Web Servisi Akış Dili (Web Services Flow Language=WSFL) dokümanı ile tanımlanır. "Servis Tanımlaması: XML mesajlaşmasından Web Servislerine" kısmında bu katman daha detaylı anlatılmıştır.

Bir Web servisi için kaçınılmaz olan özellikler; ağ ile erişebilirlik (HTTP protokolü) ve bu ağ üzerinden kullanılabilirlik (SOAP) ve son olarak servisin tanımlanabilirliği (WSDL) olduğundan, bu yığıttaki ilk üç katman herhangi bir Web servisini kullanmak için koşulsuz olarak gereklidir. En basit yığıt; ağ katmanı için HTTP, XML mesajlaşma katmanı için SOAP ve servis tanımla katmanı için WSDL`den oluşur. Bu üç katmanlı yığıt kurumlar arası veya kamusal tüm Web servislerinin desteklemesi gereken birlikte işleyebilir temel yığıttır. Özellikle kurum içi ya da özel Web servisleri diğer ağ protokollerini ve dağıtık programlama teknolojilerini destekleyebilir. Şekil 2.3 birlikte işleyebilen temel yığıtı gösterir.



Şekil 2.3- Temel Web Servisi Yığıtı
(Kreger (2001))

Şekil 2.3`te gösterilen yığıt birlikte işleyebilirliği sağlar ve Web servislerinin mevcut Internet altyapısının kullanmasını mümkün kılar. Mevcut ortamın kullanılması için düşük maliyetli bir yöntem sağlar. Esneklik sadece birlikte işleyebilirlik gereksinimi tarafından sağlanmaz, alternatif ve yardımcı teknolojiler kullanılarak ilave destek

eklenebilir. Örnek olarak HTTP üzerinde çalışan SOAP desteklenmelidir fakat bunun yanında MQ üzerinde SOAP da desteklenebilir.

Yığıtın alttaki üç katmanı, uyum ve birlikte işleyebilirlik teknolojilerini tanımlarken, sonraki iki katman (servis yayınlama ve servis keşfi) çeşitli çözümler ile gerçekleştirilebilir.

Bir WSDL dokümanının bir servis istemcisi tarafından bulunabilmesini sağlayan, servis istemcisinin yaşam döngüsünün herhangi bir seviyesindeki herhangi bir faaliyet servis yayınlamasını niteler. Bu katmandaki en basit ve en statik örnek, servis sağlayıcısının, WSDL dokümanını direkt olarak servis istemcisine göndermesidir. Buna direkt yayınlama denir. E-posta direkt yayınlamada kullanılabilecek araçlardan biridir. Direkt yayınlama statik olarak sınırlandırılmış uygulamalar için kullanışlı bir yöntemdir. Alternatif olarak, servis sağlayıcısı servisi tanımlayan WSDL dokümanını yerel bir WSDL kayıtçısında, özel bir UDDI kayıtçısında veya bir UDDI operatör düğümünde yayınlayabilir. Çeşitli yayınlama mekanizmaları "Servis Yayınlaması" kısmında daha detaylı olarak ele alınacaktır.

Bir Web servisi gerçekleştirimi bir yazılım modülü olduğu için, bir Web servisini Web servislerini birleştirerek üretmek normal ve mümkündür. Bir Web servisi bileşimi çeşitli amaçlar için çeşitli roller oynayabilir. Kurum içi Web servisleri, kamuya açık tek bir web servisi ara yüzünü oluşturmak için işbirliği yapabilirler, ya da farklı kurumlara ait Web servisleri, makineden makineye, şirketten şirkete hareketleri (transaction) gerçekleştirmek için iş birliği yapabilirler. En üstteki servis akışı katmanı, servisten servise iletişimlerin, işbirliklerinin ve akışların nasıl gerçekleştirildiklerini tanımlar. WSFL bu etkileşimleri tanımlamak için kullanılır. Web servisleri akışı başlığı "İş Süreçleri, İş akışları ve Web Servisleri" başlığında anlatılmıştır.

Web servisleri uygulamaları, günümüzün e-iş gereksinimlerini karşılamak için, güvenlik, yönetim ve servis kalitesi altyapılarını desteklemelidir. Bu dikey katmanlar yığıttaki her bir katmanda sağlanmalıdır. Bu amaca hizmet eden her bir katmandaki çözümler birbirlerinden bağımsız olabilir. Bu dikey katmanları "Web Servisleri ve Gerçek E-iş" kısmında daha detaylı olarak inceleyeceğiz.

Yığıttaki, temel Web servisleri alt yığıtını tanımlayan alt katmanlar, yığıttaki üst katmanlara nazaran daha olgunlaşmış ve standartlaşmışlardır. Web servisi teknolojisinin

olgunlaşması ve adaptasyonu üst seviyedeki katmanların ve dikey katmanların gelişmelerini ve standartizasyonunu sağlayacaktır.

Şimdi katmanların detaylarını inceleyelim.

Ağ Katmanı

Web servisi yığıtının taban katmanı ağ katmanıdır. Bu katmanda çeşitli ağ protokollerini kullanılabilir (HTTP, FTP, SMTP, MQ, RMI, IIOP, e-posta, vs.). Kullanılan protokol, uygulama gereksinimlerine bağlıdır.

Internet ile erişilebilen Web servisleri için, her platformda mevcut bulunan HTTP protokolü en uygun protokoldür. Bir yerel ağ üzerinde sağlanan ve kullanılan Web servisleri için, alternatif ağ teknolojilerinin kullanılmasına karar verilebilir. Ağ teknolojisi seçimi, güvenlik, kullanılabilirlik, performans, güvenilirlik gibi diğer gereksinimlerin sebep olduğu kriterlere göre seçilebilir. Bu, Web servislerinin mevcut daha yüksek işlevsellikleri olan ağ teknolojilerini ve MQSeries gibi mesaj tabanlı özel yazılımların kullanılmasını mümkün kılar. Çeşitli ağ altyapılarına sahip bir kurum kullandığı bu alternatif protokolleri HTTP ile birbirlerine bağlayabilir.

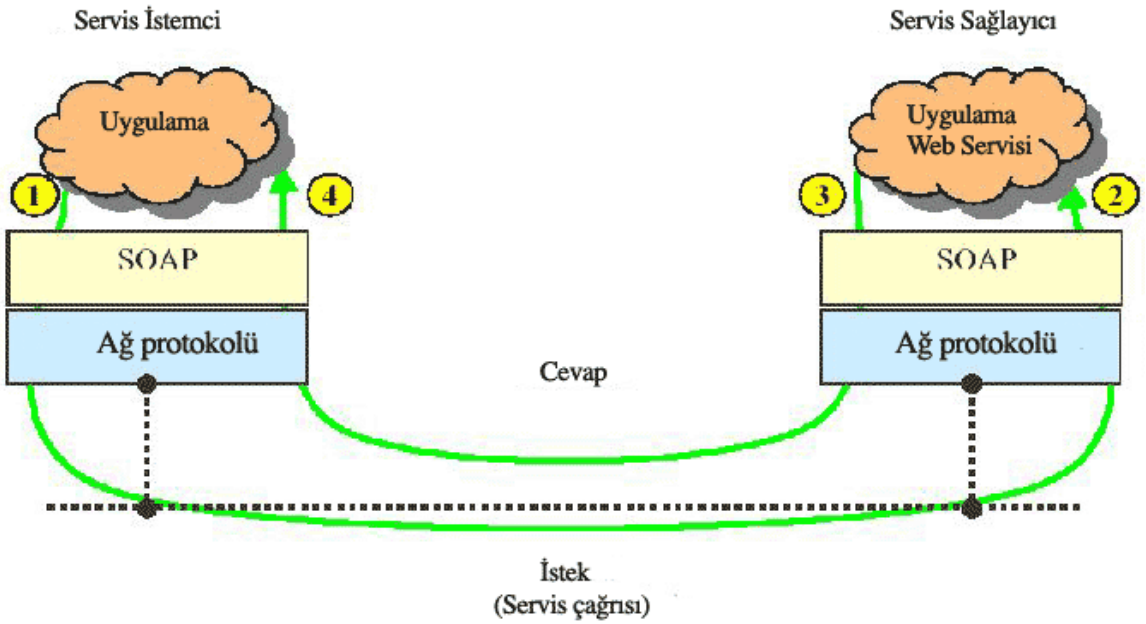
Web servislerinin iyi yanlarından biri, özel yerel ağlar ve kamuya açık Internet servisleri için ortak olarak kullanılabilecek bir programlama modeli sağlıyor olmasıdır. Sonuç olarak, ağ teknolojisinin seçimi servisin geliştirilmesinden bağımsızdır.

XML Mesajlaşma Tabanlı Dağıtık Programlama

XML mesajlaşması için şu anki standart SOAP`tır. Büyük yazılım şirketleri, W3C`ye XML mesajlaşmasında standart protokol olarak SOAP`ın kullanılmasını önerdiler. W3C`nin Web servisi mimarisinde mesajlaşma protokolü olarak SOAP`ı yayınlaması bekleniyor.

SOAP, ağ uygulamalarında yapısal verinin transferinde kullanılan, XML tabanlı basit bir mekanizmadır. SOAP üç kısımdan oluşur; bir mesajın içinde ne olduğunu tarif eden bir zarf, uygulama tarafından tanımlanmış veri tiplerinin örneklerini ifade etmek için bir kodlama kuralları kümesi ve uzaktan metot çağrılarını ve cevaplarını gösteren bir mukavele kısmı. SOAP, HTTP, SMTP, FTP, RMI IIOP veya MQ gibi çeşitli ağ protokolleri tarafından yeniden zarflanabilir ya da bunlarla birlikte kullanılabilir.

Şekil 2.4`te XML mesajlaşmasının (SOAP) ve ağ protokollerinin Web servisi mimarisinin temelini nasıl şekillendirdiği gösteriliyor.



Şekil 2.4- SOAP kullanarak XML mesajlaşma
(Kreger (2001))

Bir ağ düğümünün, XML tabanlı dağıtık programlama ortamında istemci veya sağlayıcı rolü oynayabilmesi için temel gereksinimler, bir SOAP mesajını oluşturma, ayrıştırma veya bunların ikisini birden yapabilme yeteneklerine sahip olmak ve bir ağ üzerinde iletişim kurabilme (mesaj iletebilme, alabilme veya ikisi birden) yeteneğine sahip olmaktır.

Tipik olarak bir Web uygulama sunucusundaki bir SOAP sunucusu bu işlevleri gerçekleştirebilir. Alternatif olarak, bu işlevlerini içeren programlama diline bağlı bir çalışma zamanı kütüphanesi bir API içinden kullanılabilir. SOAP ile yazılım bütünleştirmesi aşağıdaki dört temel adımı kullanarak sağlanabilir:

1- Şekil 2.4`teki 1. adımda, bir servis istemcisi uygulaması bir SOAP mesajı yaratır. Bu SOAP mesajı servis istemcisi tarafından sağlanan Web servisini çağıran istektir. Mesajın gövdesindeki XML mesajı, bir SOAP RPC isteği ya da servis tanımlamasında belirtilen şekilde hazırlanmış doküman merkezli bir mesaj olabilir. İstemci bu mesajı, servis

sağlayıcısının ağ adresiyle birlikte SOAP altyapısına (Örneğin: bir SOAP istemci kütüphanesi) sunar. SOAP istemci kütüphanesi, isteği ağın dışına göndermek için, altta duran ağ protokolüyle (Örneğin: HTTP) etkileşimde bulunur.

2- Şekil 2.4`teki 2. adımda, ağ altyapısı mesajı servis sağlayıcının SOAP kütüphanesine (Örneğin: bir SOAP sunucusu) gönderir. SOAP sunucusu istek mesajını servis sağlayıcısının Web servisine yönlendirir. SOAP kütüphanesi, uygulama tararından gereksinimi duyuluyorsa, XML mesajını programlama diline bağlı nesnelere çevirmekle sorumludur. Bu çevrim, mesaj içinde bulunan kodlama şemaları tarafından yönetilir.

3- Web servisi istek mesajını işlemek ve bir cevap oluşturmakla yükümlüdür. Cevap da bir SOAP mesajıdır. Şekil 2.4`teki 3. adımda, cevap SOAP mesajı hedef olarak servis istemcisini göstererek, SOAP kütüphanesine sunulur. Senkronize HTTP üzerinde istek/cevap yönteminin kullanılması durumunda, mesajlaşmanın istek/cevap mekanizmasını gerçekleştirmek için, alttaki HTTP protokolünün istek/cevap mekanizması kullanılır. SOAP kütüphanesi, ağı kullanarak, SOAP mesaj cevabını servis istemcisine döndürür.

4- Şekil 2.4`teki 4. adımda cevap mesajı, servis istemcisi düğümündeki ağ altyapısının kullanımıyla alınır. Mesaj SOAP Kütüphanesi tarafından gerekiyorsa XML mesajlarının hedef programlama dilindeki nesnelere çevrilerek, yönlendirilir. Son olarak cevap mesajı istemciye sunulur.

Bu örnekte çoğu dağıtık programlama ortamında ortak bir şekilde kullanılan istek/cevap yöntemi kullanılmıştır. İstek ve cevabın alınması ve iletilmesi senkronize ya da asenkronize olabilir. Tek yönlü mesajlaşma (cevapsız) veya bildirme (notification) gibi diğer iletişim yöntemleri kullanılabilir.

İstemci, istek mesajının hangi formatı kullanması gerektiğini nasıl bilebilir? Bu soru bir sonraki kısımda cevaplandırılmıştır.

Servis Tanımı: XML Mesajlaşmasından Web Servislerine

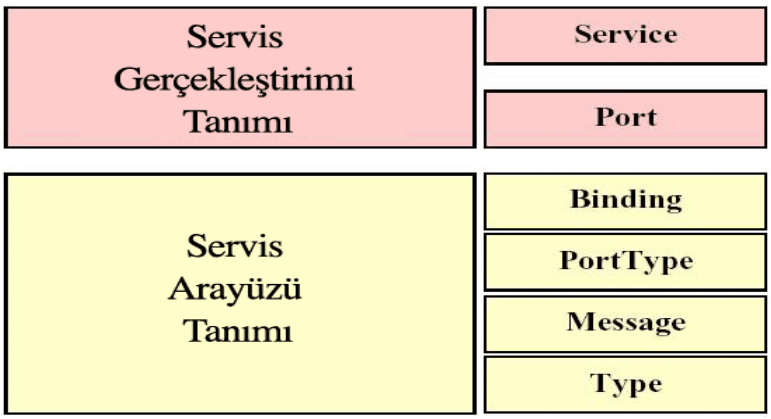
Servis tanımı Web servisleri mimarini hafif bağlaşımlı yapan ve servis sağlayıcı ve istemcisi arasındaki ortak anlayışı ve özel programlamayı ve ilişkiyi azaltan anahtardır. Örnek olarak ne istemcinin ne de sunucucunun, diğerinin kullandığı platformdan, programlama dilinden ya da varsa kullandığı dağıtık nesneden haberdar olması gerekmez.

Alttaki SOAP altyapısı ile birleştirilmiş servis tanımı, yeterli bir şekilde bu detayları servis istemcisi uygulamasından ve servis sağlayıcı web servisinden izole eder.

Temel Servis Tanımı

Temel seviyedeki servis tanımı için WSDL kullanılır. WSDL, Web servislerini doküman tabanlı ya da RPC tabanlı mesajlar içeren mesajlar üzerinde işlemler yapan uç noktalar şeklinde tanımlayan XML dokümanlarıdır. İşlemler ve mesajlar soyut bir şekilde tanımlanır, sonra da bir uç noktayı tanımlama için somut bir ağ protokolünü ve bir mesaj formatını ifade eder. İlişkili somut uç noktalar soyut uç noktalar ve servislerinin birleştirilmiş halidir. WSDL, kullanılan mesaj formatının ve ağ protokolünün etkisini ortadan aldırmak için genişletilebilir bir yapıdadır. Fakat şu anda SOAP için tanımlanan iletişim yöntemleri HTTP POST ve MIME`dır.

WSDL`in Web servisi mimarisindeki kullanımı geleneksel olarak servis tanımlamasını iki kısma ayırır: Bunlar, servis ara yüzü ve servis gerçekleştirimidir. Bu her bir kısmın ayrı olarak, bağımsız bir şekilde tanımlanmasını ve diğer kısımlar tarafından tekrar kullanılabilirliklerini sağlar.



Şekil 2.5- Temel Servis Tanımı

(Kreger (2001))

Bir servis ara yüzü tanımlaması, çoklu servis gerçekleştirim tanımlamaları tarafından işaret edilebilir ya da kullanılabilir. Servis ara yüz tanımlamalarını CORBA`daki IDL`ler gibi düşünebiliriz.

Servis ara yüz tanımlaması, yeniden kullanılabilir servis tanımlamalarını içeren WSDL öğelerden oluşur. Bunlar Şekil-2.5`te gösterilen WSDL:binding, WSDL:portType, WSDL:message, WSDL:type öğeleridir. WSDL:portType öğesinde web servisindeki işlemler (operation) tanımlanır. Bu tanımlamada, girdi ve çıktı veri akışlarında hangi XML mesajlarının görülebileceği bilgisi tanımlanır. Bir işlemi, bir programlama dilindeki bir metot imzası gibi kabul edebiliriz. WSDL:message öğesi hangi XML veri tiplerinin, bir mesajın çeşitli kısımlarını oluşturduğunu tanımlar. WSDL:message öğesi bir işlemin girdi ve çıktı parametrelerini tanımlamak için kullanılır. Mesajdaki karmaşık veri tiplerinin kullanımları WSDL:type öğesinde tanımlanır. WSDL:binding öğesi, protokolü ve veri biçimini tanımlar. Güvenlik ve diğer özel servis ara yüzü özellikler WSDL:portType öğesinde tanımlanır.

Servis gerçekleştirim tanımlaması, belirli bir servis ara yüzünün verilmiş bir servis sağlayıcısı tarafından nasıl gerçekleştirildiğini tarif eden bir WSDL dokümanıdır. Bir web servisi WSDL:service öğesiyle gösterilir. Bir servis öğesi bir grup (genellikle bir tane) WSDL:port öğesinden oluşur. Bir port, bir uçnokta (Örneğin: bir ağ adresi konumu ya da bir URL) ile bir servis ara yüz tanımlamasındaki WSDL:binding öğesini ilişkilendirir.

Servis ara yüz tanımlaması, servis gerçekleştirim tanımlamasıyla birlikte servisin tam bir WSDL tanımlamasını oluşturur. Bu ikili servis istemcisine, web servisini nasıl çağıracağını anlatmak için yeterlidir.

Servis Tanımlarının Yayınlanması ve Keşifleri

Servis Yayınlanması

Web servislerinin yayınlanması servis tanımlarının üretimi ve sonuç olarak izleyen yayınlama aşamalarını içerir.

Servis tanımlaması otomatik olarak üretilebilir, elle kodlanabilir ya da mevcut ara yüz tanımlamalarının birleştirilmesiyle oluşturulabilir. Geliştiriciler UDDI de dahil olmak üzere bir servis tanımlamasının tamamını elle kodlayabilirler. WSDL ve UDDI parçalarını programlama modelinden ve Web servisinin çalıştırılabilir modülünden faydalanarak elde etmeye yarayan araçlar vardır. Servis tanımlamasının bazı parçaları mevcut durumda elde olabilir (örneğin Web servisi bir endüstriyel servis tanımlama standardını temel alıyor olabilir). Böylece üretilmesi çok az bir gayret gerektirebilir.

Bir servis tanımlaması çeşitli mekanizmaların kullanımıyla yayınlanabilir. Bu çeşitli mekanizmalar servisten faydalanmak isteyen uygulamanın ne kadar dinamik olacağına bağlı olarak farklı yetenekler sunabilir. Servis tanımlaması çeşitli mekanizmalar kullanılarak çoklu servis kayıtçılarında yayınlanabilir.

En basit durum direkt yayınlamadır. Direkt yayınlama, servis sağlayıcının, servis tanımlamasını, servis istemcisine direkt olarak göndermesi anlamına gelir. Bu bir e-post eki, bir FTP sunucusu hatta bir CD-ROM`u kullanarak yapılabilir. Direkt yayınlama, iş ortaklarının, Web üzerinden e-iş yapma kuralları konusunda anlaşmalarını üzerine ya da servis istemcisinin servisle ilgili ödemesi gereken parayı ödemesinin üzerine yapılabilir. Bu durumda, servis istemcisi servis tanımlamasının yerel bir kopyasını saklayabilir.

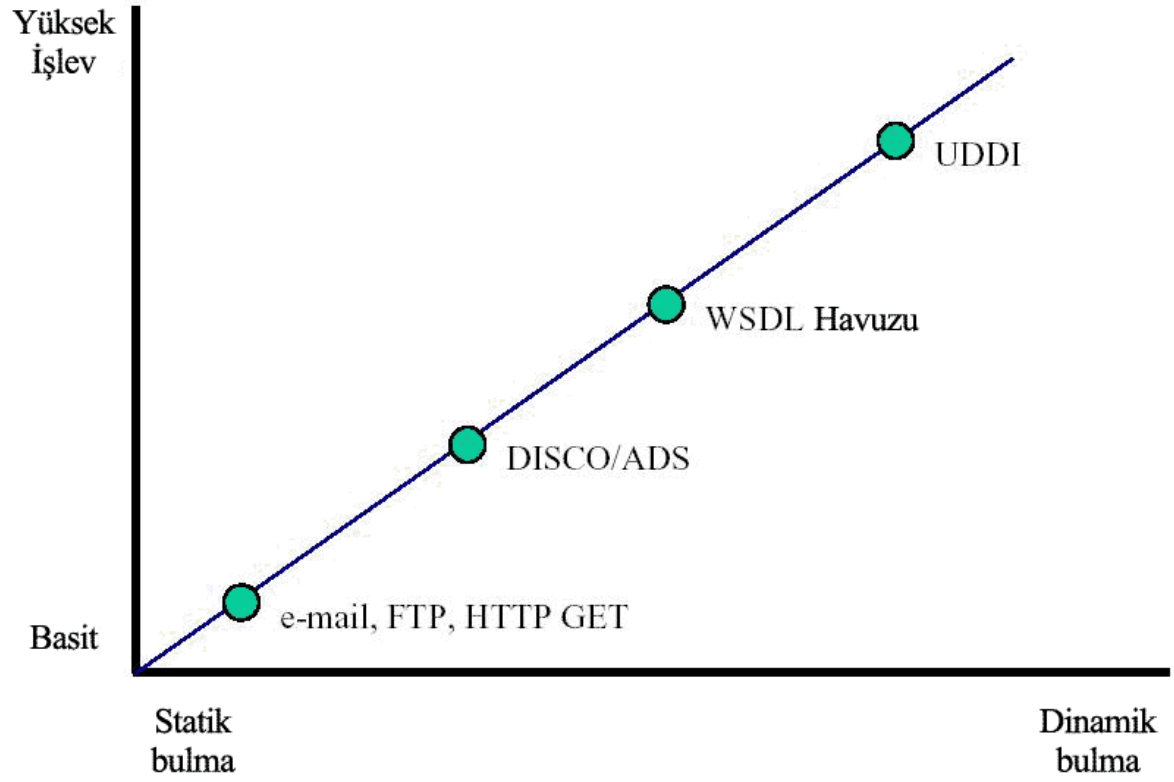
Bundan biraz daha az dinamik olan bir metot da DISCO ve ADS`nin kullanımıdır. DISCO ve ADS, verilmiş bir URL`den Web servis tanımlamalarının HTTP GET yoluyla alınmasına yarayan mekanizmalardır. Geliştirilmiş bir servis tanımlaması havuzu ilave arama kabiliyetleri olan bir önbellek gibi hizmet verebilir.

Bir kurumun içindeki makinelere yayılmış servis tanımlama havuzları için, bir servis sağlayıcısı yayınlama işlemi için bir özel UDDI düğümünü kullanabilir. UDDI tipleri, kendisinde yayınlanan Web servislerinin kapsama alanlarına bağlı olarak çeşitlenebilir.

- **Kurum içi uygulaması UDDI düğümü:** Bir şirketteki kurum içinde uygulamaların bütünleşmesinde kullanılacak bu tipte bir UDDI düğümü ile yayınlanmalıdır. Bu tip bir UDDI düğümünün kapsama alanı, tek bir uygulama, bölüm veya kurum olabilir. Bu UDDI düğümleri bir güvenlik duvarının arkasında konumlanır ve servis yayınlayıcılarının, servis kayıtçıları ve servis kayıtçılarının erişilebilirlikleri, kullanılabilirlikleri ve yayınlama gereksinimleri üzerine daha fazla kontrol haklarına sahip olmalarına izin verir.
- **Portal UDDI düğümü:** Bir şirket tarafından dıştaki ortakların kullanabilmesi için yayınlanan Web servisleri portal UDDI düğümünü kullanabilirler. Bir portal UDDI düğümü servis sağlayıcısının güvenlik duvarının dışında ve güvenlik duvarlarının arasında çalışabilir. Bu tip özel UDDI düğümleri sadece bir şirketin, dıştaki ortaklarının servis istemcilerine sağlamak istediği Web servislerinin tanımlamalarını içerir. Bu, şirketlerin, servis tanımlayıcıları, UDDI düğümüne yapılan erişimleri ve UDDI düğümlerinin servis kaliteleri üzerindeki kontrolü ellerinde tutmalarını sağlar.

- **Ortak katalogu UDDI düğümü:** Belirli bir şirket tarafından kullanılacak olan Web servisleri bir ortak katalogu UDDI düğümü ile yayınlanabilir. Ortak katalogu UDDI düğümü güvenlik duvarının arkasında konumlandırılır. Bu tip özel UDDI düğümleri, sadece yasal iş ortaklarına ait, onaylanmış, test edilmiş ve geçerli Web servisi tanımlamalarını içerir.

- **E-Pazar UDDI düğümü:** Servis sağlayıcının, servis istemcisinin kullandığı diğer servis sağlayıcıların servisleriyle rekabet etmek istediğim durumlarda Web servisi tanımlayıcısı bir E-Pazar UDDI düğümünde yayınlanmalıdır. E-Pazar UDDI düğümleri, bir endüstri standartları organizasyonunda, ya da bir konsorsiyumunda sunulurlar ve belirli bir endüstriden şirketlerin servis tanımlayıcılarını içerirler. Bu servislerin belirli standartları sağlaması gerekir (Örneğin: aranabilir meta-veri, ara yüzler veya veri yapıları).



Şekil 2.6- Servis keşif düzlemi
(Kreger (2001))

Şekil 2.6`da yayınlama ve keşif için en basit ve statik teknolojilerden, en dinamik ve daha karmaşık teknolojilere bir düzlem gösteriliyor.

Servis Keşfi

Web servislerinin keşfi servis tanımlarının ele geçirilmesi ve tanımların kullanılması işlemlerinden oluşur. Elde etme işlemi çeşitli mekanizmalarla yapılabilir.

Servisin Tanımlamasının Elde Edilmesi

Web servis tanımlarının yayınlanmasında olduğu gibi, Web servis tanımlarının elde edilmesi de, servis tanımlamasının nasıl yayınlandığına ve Web servis uygulamasının ne kadar dinamik olduğuna bağlı olarak çeşitlilik gösterir. Servis istemcileri, Web servislerini, bir uygulamanın yaşam çevriminin iki fazı esnasında bulular: tasarım zamanı ve çalışma zamanı. Tasarım zamanında, servis istemcileri, Web servisi tanımlamalarını destekledikleri ara yüz tipiyle ararlar. Çalışma zamanında, servis istemcileri, Web servisi tanımlamalarını nasıl iletişim kurduklarına ya da servisin kalitesini temel alarak arar.

Direkt yayınlama yaklaşımında, servis istemcileri, servis tanımlamalarını, çalışma zamanında kullanmak üzere önbellekte saklarlar. Servis tanımlaması statik olarak, programın içinde temsil edilebilir, bir dosyada saklanabilir, ya da basit bir yerel servis tanımlaması havuzunda saklanabilir.

Servis istemcileri, servis tanımlamalarını, tasarım zamanında, ya da çalışma zamanında servis tanımlaması havuzlarından, basit bir servis kayıtçısından, ya da bir UDDI düğümünden alabilirler. Arama mekanizması, ara yüz tipine (bir WSDL şablonunu temel alarak), çalıştırma bilgisine (protokoller), özelliklere (servis kalitesi parametreleri gibi) gibi çeşitli özelliklere göre sorgulama yapabilmeyi mümkün kılmalıdır.

Servis Tanımlamasının Kullanılması

Bir servis istemcisi, bir servis tanımlamasını elde ettikten sonra, onu, servisi çağırmak için işlemeye ihtiyaç duyarlar. Servis istemcisi, servis tanımlamasını SOAP istekleri oluşturmak için, ya da programlama diline bağlı Web servisi vekillerini oluşturmak için kullanır. Bu oluşturma bir Web servisi çağrısı oluşturmak için tasarım zamanında veya çalışma zamanında yapılabilir. WSDL dokümanlarından, programlama dili çağrıları oluşturmak için çeşitli araçlar kullanılabilir. Bu çağrı işlemleri uygulama

programına bir API sunar ve XML mesajlaşmasının detaylarını uygulama programından izole eder.

Servis Akış Katmanı

WSFL (Web Service Flow Language) yığıtın en üst noktasındaki akış katmanı için bir standarttır. Bu katman yığıttaki diğer katmanlardan faklıdır; iş süreci modellemesini ve iş akışlarını tarif eder. WSFL Web servislerinin iş akışında nasıl etkileşimde bulunduğunu ve servisten servise iletişimleri nasıl gerçekleştirdiğini tarif eder.

Gerçek E-İş İçin Web Servisleri

SOAP ve HTTP birlikte işleyebilir XML mesajlaşması için yeterli bir altyapıdır ve WSDL servis istemcisi ve servis sağlayıcısı arasında hangi mesajların gidip gelmesine karar vermekte yeterlidir fakat gerçek e-iş`in tüm gereksinimlerini karşılayabilmek için bundan fazlası gereklidir. E-İş`i tam olarak desteklemek için, Web servisi yığıtındaki her bir katmana, güvenlik, güvenilir mesajlaşma, servis kalitesi ve yönetim için uzantılar gereklidir.

Güvenlik

Bir Web servisi güvenlik katmanı gerçekten gerekli midir? Endüstri, halihazırda mevcut ve yaygın kabul görmüş, mesaj tabanlı mimariler için SSL gibi iletim katmanı güvenlik mekanizması ve IPSec gibi bir Internet Protokolü Güvenliği mekanizmasına sahip iken bir diğerini eklemeye neden ihtiyaç vardır? Bu soruları cevaplamak için, gereksinimleri inceleyeceğiz ve çeşitli mevcut iletim katmanı güvenlik mekanizmalarının tek başlarına (bir Web servisi modeli için) yeterli güvenliği sağlamadıkları senaryoları araştıracağız.

Genel olarak, Web servisleri güvenlik katmanının sağlaması gereken dört temel güvenlik gereksinimi vardır:

- Gizlilik, bilginin, kimliği doğrulanmamış kişiler veya süreçlere kapalı olması özelliğidir.

- Kimlik doğrulama, erişim haklarını temel alan erişimleri ve bir göndericinin gönderdiği mesajı göndermeye yetkili olduğunun garantisini sağlayan bir otorite sağlar.

- Veri bütünlüğü, verinin fark edilemez bir şekilde değiştirilmemesini veya yetkilendirilmemiş bir şekilde yok edilmemesini sağlayan özelliktir.

- Kaynağın ispatı, bir mesajın, ya da verinin kaynağının tanımlanmasını sağlayan özelliktir. Mesajın, gereği gibi tanımlanmış bir gönderici tarafından gönderildiğini ve daha bir önceki adımda gönderilmiş olması gereken mesaj olmadığını garantileyen özelliktir.

XML mesajlaşmasını temel alan dinamik Web servisleri dünyasındaki farklı erişim şekillerini yönetme gereksinimi, güven ve risk değerlendirme arasındaki ilişkinin yeniden gözden geçirilmesini gerektiriyor. Tek bir varlığı temel alan mevcut kontrol mekanizmaları, belirli bir işlemi gerçekleştirmek için, tek bir varlığın güvenilir bir otorite tarafından kendisine sağlanmış kimlik doğrulama altında hareket etmesini sağlayan rol tabanlı bir alan ilişkisine dayanır. Web servisleri mimarisi, bilgi isteyen servis istemcilerini ve bilgi sağlayan servis sağlayıcılarını ve bazı durumlarda bilgi hakkında bilgi sağlayan aracı düğümlerini tanımlar. Aracı düğümler sahip olduğu verilere erişim için birçok istek alır ve kimin neyi istediğine ve isteyen birinin istemekte olduğu veriye erişim hakkının olup olmadığına karar verebilmek ister. Web servisi ortamında altyapılar ve ilişkiler hızlı bir şekilde değişir, bunları yöneten kuralların, erişimi engelleme ve erişime izin verme konularında esnek olması gerekir.

XML bu tür servisler için ortak bir ara yüz sağlayabiliyorken, yukarıda bahsedilen güvenlik mekanizmaları konusunda herhangi bir faydası yoktur.

SOAP zarfı XML ile tanımlanmıştır ve hareket tanımlayıcısı (transaction ID), mesaj yönlendirme bilgisi ve mesaj güvenliği gibi geniş bir çeşitlilikte meta-bilginin mesaja eklenmesini mümkün kılar. SOAP zarfı iki kısımdan oluşur: Başlık (header) ve gövde (body). Başlık, SOAP mesajına özellikler eklemek için kullanılan mekanizmadır. SOAP başlığının bütün alt öğelerine başlık girdisi denir. Gövde, uygulama verisi için bir kap görevi üstlenir. Böylece SOAP iletişim katmanı ile uygulama katmanı arasında yeni bir katman gibi görülebilir. Bu katman mesaja ait meta bilginin taşınması için uygun bir katmandır.

SOAP başlığı, SOAP mesajlarını birçok yönde kullanmak üzere genişletilebilir olmasını sağlamak için geliştirilebilir bir yapı sunar. SOAP başlığı mesajlara güvenlik özelliklerinin eklenmesi için en uygun yerdir fakat SOAP tanımlaması tek başına bu şekilde kullanılabilecek SOAP başlıkları tanımlamaz.

Şimdi iletişim katmanındaki çeşitli güvenlik mekanizmalarının Web servisleri modelinde neden yeterli olmadıklarını ve Web servisi güvenlik katmanına neden ihtiyaç duyulduğunu daha yakından inceleyelim ve bu katmanının nasıl bir yapıda olduğunu gözden geçirelim.

SSL ve IPSec gibi güvenli iletişim katmanları, iletim esnasında mesajın bütünlüğü ve gizliliğini sağlayabilir fakat bunu sadece noktadan noktaya (point to point) iletişim için sağlarlar. Bununla birlikte SOAP mesajları aracılık eden noktalarca (intermediary) alındığında ve işlendiğinde, eğer bütün aracı noktalar arasında bir güven ilişkisi yoksa güvenli uçtan uca (end to end) iletişim mümkün olmaz. İletişim bağlantılarından biri güvenli değil ise de uçtan uca güvenlikten bahsedilemez. Web servisleri topolojisine baktığımızda, SSL ve IPSec, SOAP mesajlarının uçtan uca güvenlikleri için yeterli değildir.

Uçtan uca güvenliği sağlamanın tek yolu, uygulama veya özel yazılım (middleware) katmanında güvenliğin sağlanmasıdır. İletişim kuran noktalar arasında, mesajın şifresiz olarak bulundurulduğu bir başka nokta var ise, bu nokta muhtemel bir saldırı noktasıdır. Fakat, yeni güvenlik açıkları oluşturmadan ve riski arttırmadan, kriptografik işlevselliğin mevcut bir ya da yeni bir uygulamaya eklenmesi kolay ve istenen bir yoldur.

SOAP Orta noktalarının kullanımlarından biri de, genellikle farklı iletişim protokollerini kullanarak, mesajların farklı ağlara gönderilmesidir. Bütün iletişim bağlantıları güvenli olsa da ve orta noktalar güvenilir olsa da, mesajın kaynağının doğrulanmışlık bilgisi gibi güvenlik bilgileri, mesajın iletileceği yoldaki iletişim protokollerinin güvenlik birimlerine iletilmelidir. Bu işlem karmaşıktır ve mesajın bütünlüğü konusunda şüphelere sebep olabilir.

İletişim katmanındaki güvenlik, veriyi iletişim noktaları arasında gezerken korur. Orta noktalarda tutulan veri ile ilgili bir güvenlik sağlamaz. Bir iletimden sonra veri alındıktan ve deşifre edildikten sonra, iletişim katmanı güvenliği, verinin kimliği doğrulanmamış erişimlere karşı korunması konusunda çok fazla yardımı olmaz. Mesajların saklandığı ve iletildiği durumlarda, mesaj katmanında güvenlik gereklidir.

Görüldüğü gibi SSL ve IPSec gibi güvenlik özellikleri, Web servisi geliştirme yaklaşımında ve bazı kullanım senaryolarında yeterli değildir. Yapılması gereken aşağıdaki bileşenlerden meydana gelen kavramsal bir Web servisi güvenlik katmanının yaratılmasıdır:

1- Ağ güvenliği için:

   a. Gizliliği ve bütünlüğü sağlayan SSL ve HTTPS gibi güvenlik mekanizmalarının desteklenmesi

2- XML Mesajları için:

   a. Bağlantı içerisinde orta nokta yok ise, gönderici kullanıcı kimliği ve parolanın gizliliği konusunda SSL ve HTTPS`e güvenebilir.

   b. W3C tarafından XML Sayısal İmza desteği bir standart haline getiriliyor. Bu mekanizma, mesajın özetinin çıkarılmasını ve mesajın göndericisi tarafından, gönderenin özel anahtarıyla imzalanmasını sağlamak için, standart bir SOAP başlığı ve algoritmalar tanımlıyor. Bu sayede alıcı mesajın göndericisinin kimliğini doğrulayabilecektir.

Servis Kalitesi

Servis kalitesi dikey katmanı, kavramsal Web servisi yığıtındaki her bir katman için bilgiyle ilgili tanımlamalar sağlar. Örneğin, ağ katmanı için, çeşitlik kalite servisi seviyelerinde ağları kullanabiliyor olmayı ifade eder.

Ağ ve XML mesajlaşması katmanlarının servis kalitesini sağlamaları için aşağıdaki üç özelliği sağlamaları gerekir:

- Bir mesaj bir defa gönderilir, düğümler veya servis altyapısı bir mesajın yeniden iletilmesini istemez
- Servis istemcisi, bir isteği yapar ve isteğinin alındığına dair bir cevap alıncaya kadar isteğini tekrarlar. Servis sağlayıcının bir mesajı tekrarlı olarak işlemesi bir problem değildir. Bu druum, uygulamada, "her bir mesajın tekil bir ID`si vardır" anlamına gelebilir. Servis istemcisi, kendisinin karar verdiği aralıklarla cevaplanmamış isteklerini tekrar iletir. Servis sağlayıcı bir onay mesajı gönderir ve eğer isteği işleyemez ise, bir "mesaj işlenemedi istisnası" ("cannot process message exception") gönderir.
- Servis istemcisi bir cevap alması, kesinlikle onun isteğinin işlendiği anlamına gelir.

Servis ve Uygulama Yönetimi

Web servisleri kritik iş parçaları haline geldikçe, yönetimlerinin gereksinimi doğacaktır. Bu ifadedeki yönetim, bir üreticiden alınmış, ya da belirli bir amaç için geliştirilmiş, Web servisi alt yapısının, Web servislerinin, servis kayıtçılarının ve Web servisi uygulamalarının, varlığını, kullanılabilirliklerini ve sağlıklarını gözlemlemeye yarayan uygulamalardır.

Web servislerini, kavramsal Web servisi yığıtındaki tüm katmanlar için yönetmek mümkün olmalıdır.

Ağ katmanı için, mevcut ağ yönetimi ürünleri neredeyse tüm ağ alt yapılarını destekler. Kurum içindeki Web servisleri için kullanılan ağ altyapısını yönetmek için bu ürünler kullanılabilir. Kurumlar birbirleriyle iletişim kurduklarında, birbirlerine, Web servisi altyapılarının kullanılabilirliklerini raporlamalılardır. Web servisi altyapısının kullanılabilirliğini etkileyen ağın kullanılabilirliği de bu raporda belirtilmelidir.

XML mesajlaşması katmanında, protokol, kurum içindeki mevcut altyapı yönetim araçlarını kullanarak yönetilmelidir. Kurumlar arası iletişimdeyse, her bir düğümün kendi protokollerinin keşfi ve raporlamalarını sağlamaları gerekir. Örnek olarak, A düğümü XML mesajlaşması protokolünü destekliyorsa, B düğümünün kullanabileceği ve aktif mesajlaşmaları sorgulayabileceği bir ara yüz sağlamalıdır.

2.2.3. Web Servisleri ve CORBA Kıyaslaması

Web servislerinden önce geliştirilmiş yazılım bütünleştirme teknolojilerinin içinde en iyi çözüm olarak CORBA`yı gösterebiliriz. Çünkü CORBA`nın dışındaki iki teknoloji olan DCOM ve Java/RMI`nın sunduğu çözümler platformdan bağımsız olmayı sağlamıyor. Bu yüzden Web servisleri ile bu iki teknolojiyi kıyaslamak gerekmiyor.

CORBA ile Web servislerini kıyaslayabilmemiz için öncelikli olarak iki tekonojinin sundukları mimarileri göz önüne getirmemiz gerekiyor. Çizelge 2.1`de iki mimariye ait kıyaslama amaçlı basitleştirilmiş bir yığıt ikilisi görüyoruz.

CORBA yığıtı	Web Servisi Yığıtı
IDL	WSDL
ORB	SOAP

IIOP	HTTP
TCP/IP	TCP/IP

Çizelge 2.1- Corba ve Web Servisleri yığıtları

Çizelge 2.1`de iki teknolojiye ait mimaride hangi öğeye karşılık hangi öğenin var olduğunu görebiliyoruz.

Şimdi katmanlardaki farklılıkları ve bu farklılıkların hangi teknolojide hangi yönde iyi ya da kötü özelliklere sebep olduğunu incelemek için aşağıdan yukarıya doğru yığıtları inceleyelim:

- IIOP <-> HTTP:

Gördüğümüz gibi iki mimari uygulama katmanından itibaren farklılık göstermeye başlıyor. Web servisleri genel ve yaygın kullanımda olan HTTP protokolünü kullanırken CORBA`da iletişim için IIOP protokolü kullanılıyor. IIOP protokolü OMG`nin ilkel veri tiplerini (tamsayı, karakter, vs.), CORBA nesnelerini ve dizileri network üzerinde transferlerinin yapılabilmesi için geliştirdiği özel bir protokol iken, HTTP protokolü genel amaçlı veri transferi için geliştirilmiş bir protokoldür. CORBA`da IIOP üzerinden transfer edilen veri, çağrılan prosedürün parametre olarak kullandığı ya da dönüş değeri olarak döndürdüğü verilerin ikili olarak tasviri iken, Web servislerinde HTTP protokolü üzerinden gönderilen veri, aynı amaca hizmet eden fakat veri tipi olarak ikili veri yerine XML mesajları olan SOAP mesajlarıdır. Bu farklılık CORBA mimarisini temel alan çözümlerin Web servisleri çözümlerine nazaran daha yüksek performans sağlamalarına sebep olur. Çünkü sonuç olarak aynı manaya gelen veriler CORBA`da daha az boyutta olacaktır. Bu da ağa yüklenen verinin daha az olmasına ve veri transferinin daha hızlı olmasını sağlıyor. Fakat Web servislerinde isteğe bağlı olarak istemci ve sunucu arasındaki iletişim, sıkıştırılmış SOAP mesajlarıyla da yapılabilir ve böylece ağda transfer edilen verinin boyutu azaltılabilir. Mevcut SOAP gerçekleştirimleri bu özelliği sağlıyorlar. Bunun yanında HTTP protokolü Internet üzerinde yaygın bir protokoldür ve hemen hemen bütün sistemlerde desteği vardır. Örnek olarak, oluşturulacak olan sistemin bir güvenlik duvarının arkasında konumlanması istendiğinde HTTP için bunu yapmak mümkünken, IIOP için mümkün değildir, ya da standartların dışında çözümleri vardır. Bu durum Web servislerinin

açık Internet standartları üzerine kurulu bir teknoloji olması yönünde önemli bir özelliktir. Bu farklılıkla, Web servisleri yazılım bütünleşme konusunda tüm platformların desteğini sağlama kriterinde CORBA`dan daha üstün sayılır diyebiliriz.

- ORB <-> SOAP:

Bu katmanlarda yapılan işlem sunucu ve istemci düğümlerinde istek ya da cevap birimlerinin istemci ya da sunucuya gönderilmesi için, bir alt katman olan ağ katmanın kullanılmasıdır. ORB`da bu işlem için, servisin vekilliğini yapan gövde de iskelet nesneler kullanılır. SOAP`da ise istemci ve sunucu tarafında konumlandırılmış SOAP işlemcilerinin oluşturduğu istek veya cevap mesajları kullanılır. Bu mesajlar XML formatındadır. Gövde iskelet gibi kavramların SOAP`da (dolayısıyla Web servislerinde) olmaması, uygulamada birçok kolaylık getirir ve bunun yanında iki taraf arasındaki bağımsızlığı arttıran bir özelliktir. SOAP`da, istemci uygulamada "X sunucusundaki Y servisini çalıştırıp sonucunu geri getir" diyebiliyorken, ORB`daysa vekil bir sınıf kullanılır. Bu, istemcide sunucuda oluşturulan yapının işlevsel olmasa da, yapısal bir kopyasının oluşturulması demektir. Bu durum, sunucuyla istemci arasında belirli bir derecede bağımlılık oluşturur.

- IDL <-> WSDL:

Servislerin, istemciler tarafından kullanılması için**,** ara yüzlerinin, tanımlandıkları bu katmanda CORBA için IDL kullanılırken Web servisi için WSDL kullanılır. IDL OMG tarafından CORBA teknolojisinde kullanmak üzere geliştirilmiş bir format iken WSDL bir XML dokümanıdır. IIOP<->HTTP kıyaslamasında değindiğimiz gibi bütünleştirme için mevcut bir varlığın kullanılması, teknolojiye, desteklediği sistemlerin artması özelliğini sağlar ve sunucu ile istemci arasındaki bağımlılığı düşürür.

Sonuç olarak kısım 1.2.4`te de değindiğimiz gibi Web servislerinden önceki teknolojilerde eksikliği duyulan özellikler, teknolojilerin açık Internet standartlarından uzaklaşmış olmaları, istemci ve sunucular arasında tam bağımsızlığı sağlayamıyor olmaları ve uygulamalarındaki zorluklardı. Web servisleri, SOA mimarisinin bir gerçekleştirimi vazifesini görerek bu eksikliklere cevap sağlar.

3. YAZILIM BÜTÜNLEŞTİRMESİ ÇÖZÜMLERİNİN E-POSTA İSTEMCİSİ ALANINDA UYGULANMASI

Bu bölümde çalışmanın uygulama kısmında geliştirilen yazılımlar anlatılacaktır.

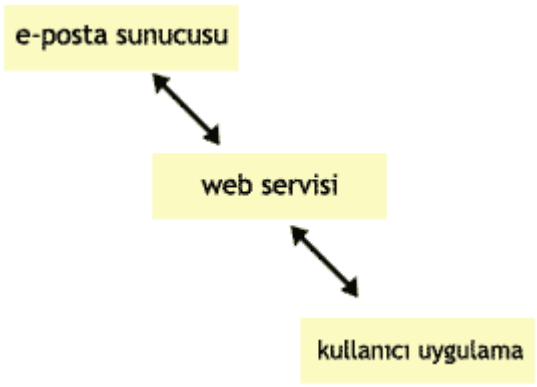
3.1 Uygulama Alanının Tanımlanması

Uygulama kısmında bir web servisi ve bu web servisini kullanan bir web uygulaması geliştirilmiştir.

Geliştirilen web servisi bütün e-posta istemcilerinin yaptığı ortak ve en temel işlevleri gerçekleştiren 3 servis içeriyor:

- E-posta sunucusuna bağlanma (kimlik doğrulama işlemi),
- Gelen kutusundaki e-postaları alma,
- E-posta gönderme

Bu web servisini kullanan uygulama programı olarak bir web uygulaması geliştirilmiştir. Uygulama programı olarak masaüstü, konsol veya başka bir ortamda çalışan bir uygulama geliştirilebilirdi. Bu konudaki tek kısıtlama istemci uygulamanın web servisleri teknolojisini destekliyor olmasıdır ve günümüzde birçok yazılım geliştirme ortamı bu kriteri sağlıyor. Şekil 3.1`de bir e-posta sunucusu, geliştirilen web servisi ve web servisini kullanan uygulama arasındaki iletişimi görebilirsiniz.



Şekil 3.1- Uygulamanın genel görünüşü

Uygulama alanı seçiminde, e-posta istemci geliştirilmesi problemine standart bir yaklaşım getirilebileceği fikri önemli bir rol oynamıştır. Üç katmanlı veritabanı uygulaması geliştirme yaklşamında en alt katmanda verilerin tutulduğu bir veritabanı, bir üst katmanda bu veritabanıyla direkt olarak etkileşimde bulunan (çeşitli kriterlerdeki verilerin alınmasını, çeşitli kriterlerdeki verilerin güncellenmesi ve veritabanına yeni verilerin eklenmesini sağlayarak) fonksiyonları içeren bir kütüphane ve en üst katmanda da ikinci katmandaki kütüphaneleri kullanan ve kullanıcı ile direkt olarak etkleşimde bulunan çeşitli paltformlarda çalışan uygulamar vardır. Geliştirilen uygulamada bu yaklaşımdan esinlenilmiştir. En alt katmandaki veritabanına karşılık olarak e-posta sunucusu, ikinci katmandaki kütüphanelere karşılık olarak geliştirilen web servisi ve son katmandaki uygulama programlarına karşılık olarak geliştirilen web uygulaması düşünülebilir.

Bu yapı bir e-posta sunucusuyla haberleşebilen bir yazılım ile bu yazılım ile kullanıcı arasındaki iletişimi sağlayan ugyulama yazılımlarının bütünleştirilmesi problemine çözüm getirmek amacıyla oluşturulmuştur.

3.2 Uygulama Alanının Bütünleşik Yazılım Mimarisi ve Bileşenleri

Bu kısımda uygulama alanında geliştirilen web servisi ve bu web servisini kullanan uygulamanın yapıları anlatılacaktır.

3.2.1. E-Posta İstemcisi Web Servisi

E-Posta sunucusu ile haberleşen web servisi IBM WSAD yazılım geliştirme aracı ile J2EE teknolojisi kullanılarak geliştirilmiştir. Geliştirilen web servisi JavaMail API`sini kullanarak e-posta sunucusuyla haberleşir.

Web servisinde kullanılacak olan işlevleri barındıran nesne için MailClient sınıfı geliştirilmiştir. Bu sınıfın yapısını incelemeden önce, bu sınıfın metotlarında dönüş değeri ve giriş parametresi olarak kullanılan veri yapılarını anlatmak gerekiyor:

LoginReturn: Bu sınıf e-posta sunucusuyla kimlik doğrulama işleminin sonucunu istemciye bildiren iki ilkel veri tipi içerir; bir tam sayı değeri olan returnCode, işlem başarılı ise 0, başarısız ise 1 değerini alır, returnMessage ise bir karakter katarıdır işlem başarılı ise “OK”, başarısız ise e-posta sunucusundan gelen hata mesajını içerir.

SMTPSendMailInput: Bu sınıf e-posta gönderme işlevini gerçekleştiren metodun giriş parametresini tanımlar. İçerdiği veriler:

- smtpHost: E-posta gönderme işlemini gerçekleştirecek olan e-posta suncusunun adresi.
- userName: E-posta gönderme işlemi kimlik doğrulama gerektiriyorsa bu işlemde kullanılacak kullanıcı adı.
- passWord: E-posta gönderme işlemi kimlik doğrulama gerektiriyorsa bu işlemde kullanılacak parola.
- from: E-potanın göndericisinin e-posta adresi.
- to: E-postanın asıl alıcısının e-posta adresi.
- cc: E-postanın kopyasının alıcısının e-posta adresi.
- bcc: E-postanın gizli olan kopya alıcısının e-posta adtresi.
- subject: E-postanın konusu.
- body: E-postanın içeriği.

MyMail: Bu sınıf web servisindeki, sunucudan e-posta alma işlevini gerçekleştiren metodun dönüş değeridir. E-posta alma işlevinde bu sınıfa ait bir nesne dizisi döndürülür. Bu sınıfın içindeki veriler şunlardır:

- from: E-postanın göndericisinin e-posta adresi.
- subject: E-postanın konusu.
- date: E-postanın gönderilme tarihi.
- body: E-postanın içeriği.

POPReceiveMailInput: Bu sınıf e-posta alma işlevini gerçekleştiren metodun giriş parametresini tanımlar. İçerdiği veriler:

- popServer: E-posta alma işleminin gerçekleştirileceği e-posta sunucusunun adresi.
- userName: E-posta alma işleminde kimlik doğrulamada kullanılacak kullanıcı adı.
- passWord: E-posta alma işleminde kimlik doğrulamada kullanılacak parola.

MailClient sınıfı bir e-posta istemcisinin sunucuyla etkileşimde gerçekleştirdikleri üç ana işlevi en temel seviyede gerçekleştiren metotlar içerir. Bu metotlar yukarıda

anlatılan veri yapılarını ve diğer ilkel veri tiplerini parametre olarak alır ve/veya dönüş değeri olarak döndürür. Bu metotlar şunlardır:

SMTPLogin: E-posta sunucusuyla kimlik doğrulama işlevini gerçekleştirir. smtpHost, userName ve passWord karakter katarlarını parametre olarak alır ve userName kullanıcısının passWord parametresiyle smtpHost sunucusuna bağlanabilip bağlanamayacağını kontrol eder. Kontrol olumlu sonuç verirse döndürğü LoginReturn nesnesinin örneğine ait returnCode değişkeni 0, returnMessage değişkeni "OK" değerini alır. Kontrol olumsuz sonuç verirse döndürğü LoginReturn nesnesinin örneğine ait returnCode değişkeni 1, returnMessage değişkeni e-posta sunucusundan gelen hata mesajıdır.

SMTPSendMail: Parametre olarak aldığı SMTPSendMailInput sınıfına ait ssmi nesnesinin üye verilerindeki değerleri kullanarak e-posta sunucusuna bir e-posta gönderme isteğinde bulunur. İşlem başarılı bir şekilde gerçekleşmişse 0 bir istisna oluşmuşsa 1 değeri döndürülür.

POPReceiveMail: Parametre olarak aldığı POPReceiveMailInput sınıfına ait prmi nesnesinin üye verilerindeki değerleri kullanarak e-posta sunucusundan ilgili hesaba ait e-postaları bir MyMail sınıfı dizi içinde döndürür.

3.2.2. E-Posta İstemcisi Web Servisi Kullanıcı Uygulaması

3.2.1`de anlatılan web servisinin üç metodunu kullanıcının kullanımına sunulması amacıyla geliştirilen uygulama PHP dili ile yazılmış bir web uygulamasıdır. PHP 5.0 sürümünde öntanımlı olarak sisteme dahil olan php_soap kütüphanesi bir wsdsl dosyasında tanımlanan fonksiyonları, SOAP altyapısını kullanarak ilgili web servisine bağlanıp, web servisinin sunduğu fonksiyonları çalıştırabilir.

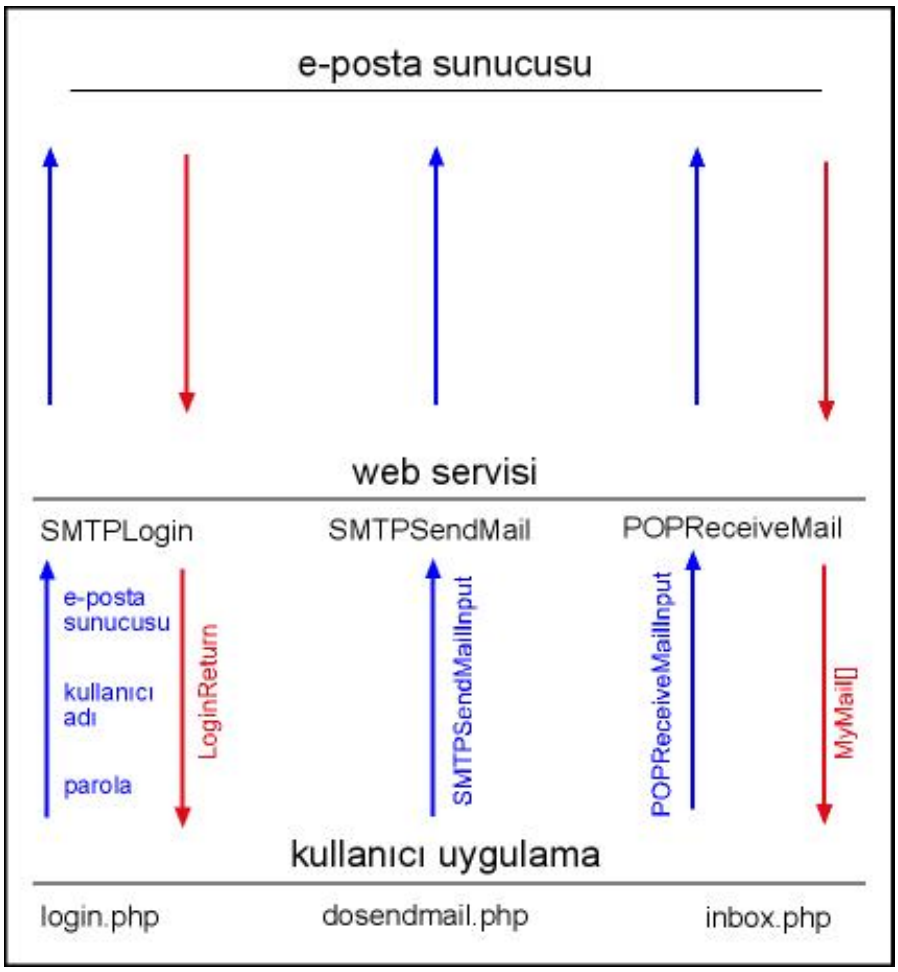
Uygulamanın web servisi ile iletişim kuran kısımlarını şu şekilde anlatabiliriz:

login.php: Bu bileşen kullanıcının bir web formu aracalığıyla gönderdiği SMTP ve POP sunucu adreslerini, kullanıcı adını ve parolasını MailClient.wsdl dosyasında tanımlanan web servisindeki SMTPLogin fonksiyonuna gönderir. Dönüş değeri olumlu ise kullanıcının giriş yaparken verdiği bilgileri birer oturum değişkeni olarak kaydeder, başarısız ise bir hata mesajı ile birlikte giriş web formunu ekrana basar.

dosendmail.php: Bu bileşen kullanıcının bir web formu aracalığıyla gönderdiği bir e-posta ya ait alıcı, diğer alıcı, gizli diğer alıcı, konu ve mesaj içeriği verilerini MailClient.wsdl dosyasında tanımlanan web servisindeki SMTPSendMail fonksiyonuna gönderen yerel sendMail fonksiyonuna göndererek e-postanın gönderilmesini sağlar.

inbox.php: Bu bileşen oturum değişkeni olarak kaydedilmiş, POP sunucu adresi, kullanıcı adı ve parolası verilerini MailClient.wsdl dosyasında tanımlanan web servisindeki POPReceiveMail fonksiyonuna gönderen yerel fetchMail fonksiyonuna göndererek web servisi ile haberleşir ve kullanıcının gelen kutusundaki e-postaları MyMail sınıfına ait bir dizi olarak elde eder ve kullanıcıya listeler.

Şekil 3.2`de bu yapı gösterilmiştir.



Şekil 3.2- Uygulamanın detaylı görünüşü

3.3 Platform ve Teknoloji Seçimleri

Bu kısımda uygulama kısmında geliştirilen uygulamaların kullandıkları teknolojileri ve gereksinimleri anlatacağız.

Web Servisi: Web servisi uygulaması IBM WSAD yazılım geliştirme aracıyla geliştirilmiştir. Web servisi bir J2EE uygulaması olduğu için web servisinin çalıştırılacağı makinada bir J2EE sanal makinası, bir uygulama sunucusu ve web servislerinin çalıştırılabilmesi için bir SOAP gerçekleştirimi gereklidir.

Kullanıcı Uygulama: Web servisini kullanan uygulama PHP dili ile geliştirilmiştir ve PHP`nin 5.0 sürümüyle dağıtılan php_soap kütüphanesini kullanır. Ayrıca bir web uygulaması olduğu için kullanıcı uygulamanın çalıştırılacağı makinada PHP desteği olan bir web sunucusunun (Örneğin: Apache) çalışıyor olması gerekiyor.

SONUÇ

Buraya kadar yazılım bütünleştirmesinin tanımını yaptık, bilgisayar dünyasında bu kavrama neden ihtiyaç duyulduğundan bahsettik, yazılım bütünleştirmesinin kullanıldığı alanlarda yazılım bütünleştirmesi kavramı yok iken nelerin nasıl yapıldığını ve bu yöntemlerdeki sorunları anlattık. Daha sonra bu sorunlara standart yaklaşımlar getiren çözümleri yani bilgisayar bilimlerinin bu sorunlara verdiği ilk tepkileri (CORBA, DCOM ve JAVA RMI teknolojileri) detaylı bir şekilde inceledik. Ardından bu çözümlerin uygulanmasında karşılaşılan problemlere ve eksik noktalarına değindik. Bu olumsuzluklara SOA mimarisinin ve Web servislerinin nasıl cevap olabildiklerini anlatmak amacıyla W3C tarafından geliştirilen Web servisleri mimarisini inceledik. Bu mimariye sonradan eklenen gerçek e-iş gereksinimlerinin doğurduğu ihtiyaca binaen yapılan eklemeleri inceledik. Son olarak da, çalışmanın asıl konusu olan web servislerini kullanan örnek bir uygulama geliştirdik ve bu uygulamayı anlattık.

Günümüzde Web servislerinin yazlım bütünleştirmesi konusunda tam anlamıyla bir standart olarak kabul edilmesi için gerekli olan çalışmalar devam ediyor. Bu çalışmanın asıl hedefi yukarıda bahsettiğimiz, Web servisleri mimarisine sonradan eklenen, Web servislerinin gerçek e-iş`te kullanılmasını sağlamak için eklenen dikey katmanları da, yatay katmanlar kadar standartlaştırmaktır.

Yukarıda bahsettiğimiz çalışmaların amacı, Web servislerinin, diğer Internet teknolojileri (WWW, HTTP, FTP vs.) gibi bir Internet standardı haline gelebilmesini sağlamaktır. Web servisleri dikey katmanların tam olarak standartlaştırılamamış olmasından dolayı henüz bu noktaya gelememiş bir teknolojidir fakat bu noktaya gelme yolunda en büyük aday konumundadır.

ÖZET

Tezimde genel olarak yazılım bütünleştirmeyi, yazılım bütünleştirmeyle ilgili olarak geliştirilmiş CORBA, COM/DOM ve JAVA RMI gibi önemli teknolojileri inceledim. Bunun ardından, SOA mimarisinin geliştirilme nedenlerini ve tanımını ve bir SOA gerçekleştirimi olan web servisleri mimarisini ve bileşenlerini inceledim. Çalışmanın sonucu olarak web servislerini kullanan bir uygulama projesi geliştirdim.

SUMMARY

In my thesis, I examined software integration in a general manner and major techonologies like CORBA, JAVA RMI and COM/DCOM related to software integration. After this, I examined the development goals of SOA architecture, definition of it and web services as an implementation example of SOA, the architecture and components of web services. As the result of the study I developed an application project utilizing web services.

KAYNAKÇA

Roy, M & Ewald A. (March 18, 1997). Inside DCOM. Retrieved June 17, 2006, from http://www.dbmsmag.com/9704d13.html.

Seshadri, G. (February 2000). Remote Method Invocation. Retrieved June 15, 2006, from http://java.sun.com/developer/onlineTraining/rmi/RMI.html.

Gisolfi, D. (July 01, 2001). Web Services Architect. Retrieved June 14, 2006, from http://www-128.ibm.com/developerworks/webservices/library/ws-arc3/.

Kreger, H. (May 2001). Web Services Conceptual Architecture. Retrieved June 13, 2006, from http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

He, H. (September, 30 2003). What Is Service Oriented Architecture. Retrieved June 10, 2006, from http://www.xml.com/pub/a/ws/2003/09/30/soa.html.

Rao, P. (November, 13 1998). JAVA RMI, CORBA or COM. Retrieved June 10, 2006, from https://www.usenix.org/publications/java/usingjava13.html

Haas, H & Brown A. (February 03, 2004). Web Services Gloassary. Retrieved June 17, 2006, from http://dev.w3.org/cvsweb/%7Echeckout%7E/2002/ws/arch/glossary/wsa-glossary.html.

Stearns, B. (January, 23 2002). Enterprise Application Integration. Retrieved June 15, 2006, from http://java.sun.com/developer/Books/j2ee/connectorch01.pdf

Enterprise Application Integration (n.d.). Retrieved June 15, 2006, from http://en.wikipedia.org/wiki/Enterprise_Application_Integration.

CORBA FAQ (n.d.). Retrieved June 15, 2006, from http://www.omg.org/gettingstarted/corbafaq.htm.

The CORBA Architecture (n.d.). Retrieved June 15, 2006, from http://java.sun.com/docs/books/tutorial/idl/intro/corba.html.

Java Remote Method Invocation (n.d.). Retrieved June 15, 2006, from http://en.wikipedia.org/wiki/Java_remote_method_invocation.

Technical Glossary (n.d.). Retrieved June 15, 2006, from http://www.informatica.com/solutions/resource_center/glossary/default.htm.

EAI – A Definition From Whatis.com (n.d.). Retrieved June 15, 2006, from http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci213523,00.html.

TEŞEKKÜR

Bu tezin tamamlanmasında büyük katkıları olan danışman hocam Doç.Dr. M. Ali Salahlı'ya ve maddi ve manevi desteklerini hiçbir zaman benden esirgemeyen aileme teşekkür ederim.

ÖZGEÇMİŞ

Adı Soyadı: Faruk Eskicioğlu
Doğum Yeri ve Yılı: DENİZLİ / 22.05.1979
Adres: D. Hüseyin Paşa Cd. Onur Sitesi C Blok Daire: 44 Bahçelievler/İSTANBUL
E-Posta: farukesk@comu.edu.tr

Eğitim Durumu

2003-2006: Yüksek Lisans / Çanakkale Onsekiz Mart Üniversitesi
1998-2002: Lisans / Çanakkale Onsekiz Mart Ünivesitesi, Bilgisayar Mühendisliği
1993-1997: Lise / Denizli Lisesi / DENİZLİ
1993-1996: Lise / Gönen Anadolu Öğretmen Lisesi / Gönen / ISPARTA
1990-1993: Ortaokul / Pamukkale Ortaokulu/ DENİZLİ
1988-1990: İlkokul / Gazi Mustafa Kemal İlkokulu / DENİZLİ
1985-1988: İlkokul / Cumhuriyet İlkokulu / DENİZLİ

Staj ve Kurslar

- [Staj-1] Türk Telekom A.Ş. Bilgi İşem Müdürlüğü, 2000 – DENİZLİ.
- [Staj-2] Çanakkale Onsekiz Mart Üniversitesi Bilgi İşlem Daire Başkanlığı, 2001 - ÇANAKKALE.

Mesleki Deneyim

- 2004-: Telsim Telekomünikasyon Hizmetleri A.Ş.
- 2002-2003: Ataper Bilgisayar LTD.

Bilimsel Kuruluşlara Üyelikler

- Linux Kullanıcıları Derneği – http://www.lkd.org.tr

Çalışma ve İlgi Alanları

- Yazılım Mühendisliği.
- Tasarım Şablonları (Design Patterns).
- Nesneye Yönelik Programlama (Java, C++).
- 3-Katmanlı Uygulama Geliştirme (J2EE).
- Uygulama Sunucuları.
- QT.
- ORACLE – PL/SQL
- Veri Madenciliği
- Yönetim Bilgi Sistemleri