PAMUKKALE ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
PAMUKKALE UNIVERSITY ENGINEERING COLLEGE
MÜHENDİSLİK BİLİMLERİ DERGİSİ
JOURNAL OF ENGINEERING SCIENCES
YIL : 2003
CİLT : 9
SAYI : 1
SAYFA : 47-54

YENİDEN KULLANILABİLİR YAZILIM BİLEŞENLERİNE WEB ÜZERİNDEN ERİŞİM İÇİN CORBA TEMELLİ BİR MİMARİ

R. Cenk ERDUR, Mustafa TÜRKSEVER
Ege Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 35100-Bornova/İzmir

Geliş Tarihi : 16.04.2001

ÖZET

Bilgisayar ağları teknolojilerinin gelişimi ve İnternet'in sürekli büyümesi sonucunda, İnternet, yakın bir gelecekte yeniden kullanıma dayalı yazılım geliştiren kişi veya kuruluşların ortak yazılım bileşeni deposu durumuna gelecektir. Bu yazılım deposunda bulunan yeniden kullanılabilir kodlar, analizler, tasarımlar, tasarım desenleri (design patterns) gibi bileşenlerin yeni yazılım geliştirme süreçlerinde de kullanılabilmesi için, söz konusu bileşenlerin İnternet üzerinden aranabilmesini sağlayan ortamlara gerek duyulmaktadır. Bu ortamlar; kullanıcıların sorgu girebilmelerini sağlayan arayüzler, bileşen kütüphanelerine erişimi sağlayan bağlayıcı (wrapper) programlar, kullanıcı isteklerini ilgili bileşen kütüphanelerine ileten koordinatör programlar gibi temel elemanlardan oluşmaktadır. Bu çalışmada ilk olarak bu tür bir ortam için CORBA temelli bir mimari ortaya konmaktadır. Daha sonra, aynı ortam için Java 2 platformu teknolojileri kullanımına dayanan alternatif bir mimari verilmekte ve önerilen CORBA temelli mimari ile karşılaştırılmaktadır.

Anahtar Kelimeler: Yeniden kullanım, Bileşen üst-bilgisi, Bileşen sınıflandırma, CORBA, Java 2 platformu

A CORBA BASED ARCHITECTURE FOR ACCESSING REUSABLE SOFTWARE COMPONENTS ON THE WEB.

ABSTRACT

In a very near future, as a result of the continious growth of Internet and advances in networking technologies, Internet will become the common software repository for people and organizations who employ component based reuse approach in their software development life cycles. In order to use the reusable components such as source codes, analysis, designs, design patterns during new software development processes, environments that support the identification of the components over Internet are needed. Basic elements of such an environment are the coordinator programs which deliver user requests to appropriate component libraries, user interfaces for querying, and programs that wrap the component libraries. First, a CORBA based architecture is proposed for such an environment. Then, an alternative architecture that is based on the Java 2 platform technologies is given for the same environment. Finally, the two architectures are compared.

Key Words: Software reuse, Component meta-knowledge, Component classification, CORBA, Java 2 platform

1. GİRİŞ

Bileşen tabanlı yeniden kullanım (BTYK), önceki yazılım geliştirmelerde üretilen ve kaynak kod, analiz, tasarım desenleri (design patterns), sınıf çerçeveleri (class frameworks), sınama senaryoları, belgelemeler gibi türlerde olabilen bileşenlerin yeni yazılımların üretilmesi sürecinde de kullanılmasıdır. BTYK, yazılım geliştirmenin maliyetini azaltmak, yazılım kalitesini arttırmak, yazılım geliştirme sürecini hızlandırmak ve kolay denetlenebilir bir

biçime dönüştürmek gibi avantajlar sağlamaktadır (Griss, 1993). BTYK, bu özellikleri ile yazılım mühendisliğinin en yoğun çalışılan konularından birisi olma durumunu sürdürmektedir.

Günümüzde İnternet kullanımındaki artış, gelişen bilgisayar ağı teknolojileri sonucunda İnternet'in sürekli büyümesi, Enterprise Java Beans (Roman et al., 2002) bileşenleri gibi platform bağımsız veya Active-X (Anon., 1997) bileşenleri gibi sık kullanılan yazılım bileşenleri yazma olanağı getiren teknolojilerin yaygınlaşması sonucunda İnternet üzerinde BTYK için gerekli altyapı sağlanmış durumdadır. Akademik, özel ve resmi kuruluşların ürettikleri yazılım bileşenlerini İnternet üzerinden kullanıma sunmaya veya pazarlamaya başlamaları ile İnternet tüm yeniden kullanıma dayalı yazılım geliştiren kişilerin ortak yazılım bileşeni deposu durumuna gelecektir. Bunun sonucu olarak da İnternet üzerinde BTYK'ı etkin olarak sağlamak için bileşen kütüphaneleri, yeniden kullanıcılar, bileşen kütüphanelerine erişimi sağlayan bağlayıcı (wrapper) programlar, yeniden kullanıcıların sorgularını girebilmelerini sağlayan arayüzler, yeniden kullanıcılardan gelen istekleri ilgili bileşen kütüphanelerine iletmekten sorumlu koordinatör programlar gibi temel elemanlardan oluşan bir yeniden kullanım ortamına gereksinim duyulmaktadır.

Bu çalışmada, ilk olarak bu tür bir ortamın CORBA (Common Object Request Broker Architecture – Ortak Nesne İstek Sunum Mimarisi) kullanarak tasarımı anlatılmaktadır. Daha sonra, aynı ortam için Java 2 platformu teknolojileri kullanılarak alternatif bir mimari önerilmekte ve CORBA temelli mimari ile karşılaştırılmaktadır.

Nesneye dayalı dağıtık sistem mimarisi standardı olan CORBA bundan sonraki bölümde kısaca tanıtılacaktır. Çalışmanın üçüncü bölümünde, CORBA temelli yeniden kullanım ortamı anlatılmaktadır. Java 2 platformuna dayanan mimari dördüncü bölümde tartışılmaktadır. Beşinci bölüm, önerilen her iki mimarinin karşılaştırılmasını içermektedir. Sonuçlar ve tartışma altıncı bölümde, kaynaklar yedinci bölümde yer almaktadır.

2. CORBA STANDARDI

CORBA , OMG (Object Management Group-Nesne Yönetim Grubu) tarafından geliştirilmiş dağıtık nesne mimarisi standardıdır (Orfali et al., 1997; Orfali and Harkey, 1998). OMG, bilgisayar endüstrisinde yer alan 800'den fazla kuruluş tarafından desteklenen bir çalışma grubudur. Microsoft sözkonusu destekçi kuruluşlar arasında bulunmamaktadır. Bunun nedeni Microsoft'un DCOM (Distributed Component Object Model-Dağıtık Bileşen Nesne Modeli) adındaki kendine özgü dağıtık nesne mimarisini geliştirmiş olmasıdır.

CORBA nesneleri bilgisayar ağı üzerinde herhangi bir yerde bulunabilmektedir. Uzak istemciler bu nesnelere metod çağrımları yolu ile erişebilmektedir. İstemcilerin, sunucu nesnelerin bilgisayar ağı üzerindeki yerini bilmesine gerek yoktur.

Sunucu nesnelerin hangi dille yazıldığı da istemciler açısından önemli değildir. Örneğin, bir sunucu nesne C++ sınıfları olarak veya uzun bir COBOL programı olarak gerçekleştirilmiş olabilir. İstemci açısından önemli olan sunucu nesnenin dışarıya sunduğu arayüzdür. Sunucu nesne arayüzünde, uzak istemcilere sunulabilecek servisler belirlenmektedir. Bu nedenle sunucu nesne arayüzü uzak istemciler ve sunucular arasında geçerli olan bir kontrata (anlaşma) benzetilebilmektedir. Bir nesnenin başka bir nesneden servis isteyebilmesi için, o nesnenin dışarıya sunduğu arayüzü bilmesi gerekmektedir.

IDL (Interface Definition Language-Arayüz Tanımlama Dili), CORBA nesnelerinin arayüzlerini tanımlamak için kullanılan dildir. IDL ile yazılan arayüzlerde bulunan metod prototiplerinin gerçekleştirimleri CORBA'nın desteklediği (CORBA ile bağlanabilen-binding) C, C++, Ada, Smalltalk, COBOL, Java dillerinde yapılabilmektedir. Diğer dillerin de desteklenmesi için çalışmalar sürmektedir. IDL, işletim sistemi ve programlama dilinden bağımsız arayüzler tanımlanmasını sağlayarak farklı dillerde yazılmış istemci ve sunucuların içten-işletilebilirliğine (interoperability) izin vermektedir.

IDL dili, C++ dilinin bir alt kümesi olarak tasarlanmıştır. Bu alt küme, kalıtım (inheritance), aykırı durum yönetimi (exception handling) gibi özellikleri içermektedir.

CORBA mimarisi dört temel bileşenden oluşmaktadır. Bunlar; ORB (Object Request Broker), CORBA servisleri, CORBA hizmetleri (facilities) ve uygulama nesneleridir. ORB; nesnelerin birbirlerinin yerini, gerçekleştirim dillerini, alt düzey iletişim mekanizmalarını bilmeden saydam (transparent) olarak birbirlerinden istekte bulunabilmelerini ve yanıtlar alabilmelerini sağlamaktadır. CORBA servisleri, IDL ile yazılmış arayüzler biçiminde paketlenmiş sistem düzeyinde servisler topluluğudur. Kalıcılık, isimlendirme, eş-zaman kontrol bu tür servislere ilişkin örneklerdir. CORBA hizmetleri (CORBA facilities), uygulama nesnelerinin doğrudan kullanabileceği ve IDL ile

tanımlanmış olan uygulama düzeyi çerçeveler (frameworks) topluluğudur. Gezgin etmenler (mobile agents), güvenlik duvarları (firewalls), uluslar arası yapma (internationalization) CORBA hizmetleri arasında sayılabilmektedir. Son bileşen olan uygulama nesneleri kullanıcıların geliştirdiği uygulamaya ilişkin nesnelerdir.

2. 1. Visibroker for Java 3.3

Şu anda pazarda bir çok firma tarafından üretilen çok sayıda Java dili destekli ORB bulunmaktadır. Visibroker for Java 3.3 1997 yılı sonlarında çıkmıştır ve OSAgent adı verilen hata-toleranslı (fault-tolerant) bir isimlendirme servisi içermektedir (Pedrick et al., 1998). Statik ve dinamik çağrımlar, arayüz deposu, IIOP protokolü kullanımı, "gatekeeper" adı verilen güvenlik duvarı, otomatik iş-parçası yönetimi gibi ileri özellikleri bulunmaktadır.

Bu çalışmada da yaygın kullanımı, barındırdığı ileri özellikleri ve yeterli dokümantasyonundan dolayı Inprise Visibroker for Java 3.3 kullanım için seçilmiştir.

Visibroker for Java 3.3, içerdiği CORBA 2.0 sürümü uyumlu ORB ve buna ek olarak 3 servis daha sağlar. Bu servisler isimlendirme servisi, olay servisi, "gatekeeper" servisidir. İsimlendirme servisi nesne gerçekleştirimlerine birden fazla mantıksal isim verilmesini sağlar. Bu isimler isim uzayında (namespace) saklanır. Olay servisi, nesneler arasındaki iletişimi yönetir. "Gatekeeper" servisi, bir Web sunucu üzerinde çalışır ve Web sunucu üzerinde bulunmayan nesnelere erişilebilmesini sağlar.

2. 1. 1. Visibroker for Java 3. 3. Paketi İçindekiler

Visibroker for Java 3.3. aracını başarılı bir biçimde yüklemek, çalıştırmak ve yönetmek için araç ile birlikte gelen öğeleri bilmekte yarar olacaktır. Bu öğeler aşağıda tanıtılmaktadır.

Çalışma Zamanı (Run Time) Paketi: Bu paket aşağıda belirtilen servislerin çalışması için gerekli ortamı sağlamaktadır.

SmartAgent: OsAgent adı ile de bilinen bu servis, uygulamaların kullanmak istedikleri nesneleri bulmasında kullanılır. Bu servis en başta bir kez başlatılması gereken bir süreçtir.

Lokasyon Servisi: Bu servis, Visibroker uygulamalarının bir nesnenin tüm örneklerini (instance) bulmalarını sağlamaktadır. SmartAgent ile birlikte yük dengelemede yardımcı olur.

Nesne Etkinleştirme Hayalet Programı (Object Activation Daemon-OAD): OAD, nesnelerin bir istemci uygulama tarafından gereksinim duyulduğunda otomatik olarak etkinleştirilmelerini sağlar. Böylece sunucular üzerindeki iş yükünde önemli bir azalma kaydedilir. Çünkü nesneler sürekli çalışmak yerine sadece istek geldiğinde etkinleştirilirler.

Arayüz Deposu (Interface Repository): Arayüz deposu, nesne türleri ile ilgili üstbilgi tutan çevrimiçi bir veri tabanıdır. Üstbilgi, modül bilgileri, arayüz bilgileri, operasyon bilgileri gibi bilgilerden oluşmaktadır. Dinamik arayüz kullanan Visibroker uygulamaları arayüz deposu kullanmaya gereksinim duymaktadır.

2. 1. 2. Visibroker for Java 3. 3. Çalıştırmak İçin Gerekli Olan Diğer Araçlar

Aşağıda Visibroker paketi içinde yer almayıp uygulama geliştirmek için gerekli olan ve bu nedenle bir biçimde elde edilmesi gereken öğeler listelenmiştir.

Java Uygulama Geliştirme Ortamı: Sun Microsystems tarafından tarafından sağlanan Java Development Kit (JDK) http://java.sun.com adresinden ücretsiz olarak elde edilebilir. Inprise JBuilder aracı da Java uygulama geliştirme ortamı olarak kullanılabilmektedir.

Java Çalışma Zamanı Ortamı (Java Run Time Environment-JRE): JRE, Java uygulama geliştirme ortamının bir parçası olup, Java uygulamalarını yorumlayan ve çalıştıran bir motor içermektedir. JRE, JDK1.5 sürümünden sonra JDK ile birlikte ücretsiz olarak elde edilebilmektedir.

Java Uyumlu Bir Web Tarayıcı: Netscape Navigator veya Microsoft Internet Explorer ürünleri kullanılabilmektedir. Netscape Navigator http://www.netscape.com adresinden elde edilebilmekte, Microsoft Internet Explorer ürünü ise Microsoft kuruluşu tarafından pazarlanmaktadır.

2. 1. 3. Visibroker for Java 3. 3 Yüklenmesi

Yükleme işlemi için sistem gereksinimleri aşağıda verilmektedir.

* Windows 95/98 veya Windows NT 3.5.1 / 4.0

* JavaSoft JDK 1.1.5 veya daha üst sürümü. (Visibroker for Java 3.3. sürümü ile JDK 1.2.

sürümü arasında uyum sorunları görülmüştür. Bu sorunun yeni sürümlerde giderildiğinden söz edilmektedir. Bu çalışma çerçevesinde JDK 1.1.5. sürümü kullanılmıştır.)

*Visibroker for Java 3.3. CD 'si veya CD yoksa Inprise Web sitesinden indirmek için İnternet bağlantısı.

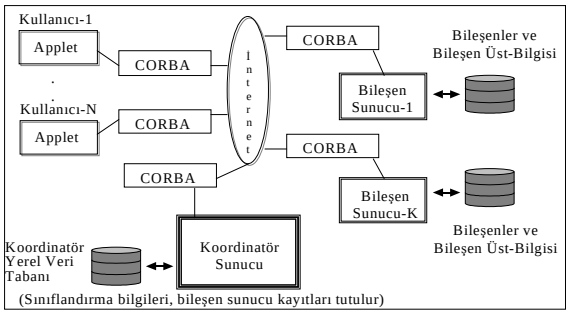
Kurulum aşamasında, gerek JDK aracı gerekse Visibroker aracı için gerekli sistem ayarları yapılmalı ve "autoexec.bat" dosyasında saklanmalıdır.

Visibroker for Java 3. 3. ve JDK paketleri yüklendikten sonra değinilmesi gereken bir nokta daha bulunmaktadır. Eğer geliştirilecek olan uygulama applet'lerden oluşuyor ise bu applet'ler dış dünyaya bir Web sunucu ile sunulmalıdır. Bu çalışmada Windows 95 ortamında Microsoft tarafından geliştirilen Personal Web Server ürünü kullanılmıştır.

3. CORBA TEMELLİ BTYK ORTAMININ TASARIMI

Şekil 1, önerilen mimarinin ayrıntılarını ortaya koymaktadır. Şekil 1'den de görüldüğü gibi. mimari üç katmandan oluşmaktadır. Bunlar, yeniden kullanıcılar, bileşen sunucular ve koordinatör'dür. Kullanıcılar ve bileşen sunucular doğrudan iletişim kurmazlar. Söz konusu iletişim koordinatör nesne yardımı ile gerçekleştirilmektedir. Böylece bileşen sunucular ve onların yapılanması ile ilgili ayrıntılar kullanıcılardan gizlenmiş olacaktır. Eğer kullanıcılar ve bileşen sunucuların doğrudan iletişim kurduğu bir yapı söz konusu olsa idi, bu kez kullanıcılar her bileşen sunucuya ayrı ayrı bağlanıp tarama yapmak zorunda kalacaktı. Koordinatörün varlığı kullanıcıları bu tür zorluklardan kurtarmaktadır. Kullanıma hazır duruma gelen bileşen sunucular kendilerini koordinatör nesneye kaydettirmek zorundadır. Böylece herhangi bir anda kullanıma açık olan tüm bileşen sunucular koordinatör nesne tarafından biliniyor olacaktır. Kullanıcılar koordinatör nesneye Web üzerinden bir Java Applet kullanarak erişmektedir.

Şekil 1'den de görüldüğü gibi sistemde değişik veri tabanları bulunmaktadır. Koordinatör sunucu; sınıflandırma için kullanılabilecek anahtar kelimelerin neler olduğu bilgisi ile bileşen sunuculara ilişikin kayıtları tutmaktadır. Bileşen sunucular; bileşenlere ilişkin üst-bilgi ile bileşenlerin kendilerini tutmaktadır.



Şekil 1. BTYG Ortamı için CORBA temelli bir mimari

3. 1. Bileşenlerin Sınıflandırılması

Kütüphanelerde bulunan bileşenler organize edilmez ise bileşenlere erişim zorlaşacaktır. Kütüphanelerdeki bileşenleri belli bir sınıflandırma tasarısına göre organize etmek, kütüphanelerin aranmasını ve kullanıcı istekleri ile kütüphanede bulunan bileşenlerin eşlenmesi işlemini kolaylaştıracaktır. Yeniden kullanım literatürü incelendiğinde, bileşenlerin sınıflandırılması konusunun üzerinde en çok çalışılan konulardan birisi olduğu görülmektedir.

Diaz tarafından önerilen boyutlu yaklaşım (faceted approach) (Diaz and Freeman, 1987) bu alanda yapılmış çalışmalarda en çok kullanılan ve önerilen sınıflandırma tasarısıdır. Bu çalışmada da bir anahtar kelime tabanlı sınıflandırma tasarısı olan Diaz'ın boyutlu yaklaşımı kullanılmıştır.

Boyutlu yaklaşım bilginin farklı kategorilere ayrılması prensibine dayanır. Böylece bu yaklaşıma göre sınıflanmış bir bileşenin farklı özellikleri hakkında bilgi edinmek olasıdır. Boyutlu yaklaşımda her farklı özellik bir boyut (facet) ile ifade edilir. Her boyut sonlu sayıda terim içerir. Bu terimler hem yeniden kullanıcılar, hem de bileşenleri üretenler için ortak bir sözlük oluşturur. Örneğin, kaynak kod içeren bileşenlerin sınıflandırılması söz konusu olduğunda "Programlama Dili" isimli bir boyut bileşenin yazıldığı dil özelliğini ifade etmek için kullanılabilir. Bu boyutun terimleri de, Java, C, C++, Pascal, Smalltalk gibi olacaktır. Bir bileşeni sınıflandırmak için, o bileşene ilişkin tanımlı boyutların her birisinden birer terim seçerek bir tanımlayıcı oluşturmak gerekmektedir.

Somut bir örnek olarak, kaynak kod türündeki bileşenleri sınıflamak için gerekli olan boyutlar ve bu boyutların içerikleri Tablo 1'de görülmektedir. Analiz, tasarım, tasarım desenleri, belgeleme gibi diğer bileşen türleri için de boyutlar ve terimler belirlenmiş olup, ayrıntılar diğer bir çalışmamızda verilmektedir (Erdur ve Türksever, 1999).

Örneğin, Tablo 1'den yararlanarak, Unix ortamında bir yığıta push işlemi yapan Java metodu bileşeni (Yığıt + Push + Metod + Java + Unix) anahtar kelimelerinin birleşmesinden elde edilen tanımlayıcı ile sınıflandırılmaktadır. Bu tanımlayıcı, ileride söz konusu bileşene erişim için anahtar saha görevi yapmaktadır.

Tablo 1. Kaynak Kod Türü Bileşenleri Sınıflandırmak İçin Gerekli Boyutlar ve Terimler

Kaynak Kod Bileşenleri Sınıflandırmak İçin Boyut ve Terimler				
Soyut-Lama	İşlem	Alt-Tür	Dil	Ortam
yığıt kuyruk bağlaçlı-liste : : :	push pop ekle sil sona-ekle :	sınıf yordam metod : : :	Java C++ C Smalltalk : :	Unix Windows 95/98 OS/2 : : :

3. 2. Koordinatör ve Bileşen Sunuculardaki Veri Tabanı Bilgileri

Koordinatör sunucuda, sistemdeki kütüphanelere ilişkin alan (domain) bilgileri, bileşen türü bilgileri, ilgili alan ve türlere ilişkin boyutlar ve içerikleri tutulmaktadır. Aşağıda yer alan Tablo 2 ve Tablo 3, koordinatör sunucudaki veri tabanı Tablolarından örnekler sunmaktadır.

Tablo 2. Koordinatör Sunucu Alan Bilgileri Tablosu

TBLDOMAIN	
Alan No	Alan Adı
1	Veri Yapıları
2	Ağ Programlama
3	Genel Algoritmalar
4	Bankacılık

Sistemde şu anda analiz, kaynak kod, tasarım ve belgeleme türü bileşenler desteklenmektedir. Diğer türler de ilgili Tabloya ekleme yapılarak desteklenir biçime getirilebilecektir. Türlere ilişkin bir Tablo olan Tablo-3 aşağıda görülmektedir.

Tablo 3. Koordinatör Sunucu Tür Tablosu

TBLTYPE
Tür Adı
Analiz
Kaynak Kod
Tasarım
Belgeleme

Boyutların ve terimlerin saklandığı Tablolar ise Tablo 4 ve Tablo 5'te görülmektedir.

Tablo 4. Boyutlara İlişkin Tablo

TBLFACETS			
Boyut No	Alan No	Tür	Boyut Adı
1	1	KOD	Soyutlama
2	1	KOD	İşlem
3	1	KOD	Tür
4	1	KOD	Dil
5	1	KOD	Ortam

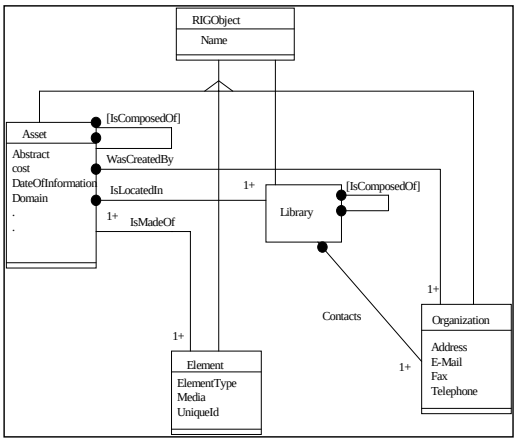
Tablo 5. Terimlere İlişkin Bir Tablo Örneği

TBLTERMS	
Boyut No	Terim
1	yığıt
1	kuyruk
1	bağlaçlı-liste
1	ağaç
1	çizge

Bileşen sunucularda, bileşenlere ilişkin tekil numaralar, diskte bulunduğu fiziksel yol ve bileşen üst bilgileri gibi bilgilerin tutulduğu Tablolar yer almaktadır. Bileşen üst bilgisi, bileşenleri daha iyi anlamak için bileşen hakkında ek bilgilerden oluşmaktadır.

Bileşenlere ilişkin ek açıklayıcı bilgiler (bileşenin sürümü, yazılma tarihi, yazarı, yazıldığı kuruluş, kuruluş adres bilgileri vb.) olan bileşen üst-bilgisinin (meta-knowledge) yeniden kullanıma dayalı yazılım geliştirmede önemli bir yeri bulunmaktadır. Bunun nedeni, bileşen üst-bilgisinin yeniden kullanıcılara bileşen seçiminde büyük kolaylık sağlayabilmesidir. Bu çalışmada bileşen üst-bilgisi RIG-BIDM (Reuse Library Interoperability Group-Basic Interoperability Data Model) temel alınarak tanımlanmıştır. RIG, yeniden kullanılabilir bileşen kütüphanelerinin içten-işletilebilirliğini (interoperability) sağlamaya yönelik standartlar geliştirilmesi amacıyla 1991 yılında kurulmuş bir çalışma grubudur (Browne and Moore, 1997).

BIDM ise yeniden kullanılabilir bileşen kütüphanelerinin içten-işletilebilirliğinin sağlanması için bu kütüphanelerin birbiri ile değişeceği bilgileri tanımlayan bir veri modelidir. BIDM ile, kullanıcılar tek bir arayüz kullanarak farklı bileşen kütüphanelerindeki bileşenlere erişme olanağına kavuşmaktadır. BIDM veri modeli, belli bir sıradüzende olan sınıflardan oluşmaktadır. Bu sınıflara ilişkin nitelikler de (attributes) belirlenmiştir. BIDM'in, James Rumbaugh'ın nesneye yönelik modelleme ve tasarım metodolojisinin grafiksel notasyonunda gösterimi Şekil 2'de verilmektedir (Browne and Moore, 1997).



Şekil 2. BIDM veri modeli (Browne and Moore, 1997'den)

Bu çalışmada bileşen üst-bilgileri yukarıda söz edilen BIDM veri modelinin bir alt kümesi kullanılarak tanımlanmıştır.

Bileşen sunucularda bulunan Tablo örnekleri aşağıda yer alan Tablo 6 ve Tablo 7'de verilmektedir. Tablo 6, bileşen sunucunun diskinde saklanan bileşenlerin hangi dizinlerde olduğuna ilişkin olarak fiziksel lokasyon bilgisi tutmaktadır. Bir sunucuda bulunan bileşenler yazılım yaşam döngüsünün her aşamasına ilişkin ürünler olabilmektedir. Bu nedenle, bir bileşen genellikle birden fazla bileşenle bağlantılı olabilmektedir. Örneğin, bir analiz bileşenine karşılık gelen tasarım ve kaynak kod bileşenleri, söz konusu analiz bileşeninin bağlantılı olduğu bileşenlerdir.

Tablo 6. Bileşenlerin Diskte Ortamındaki Adreslerine İlişkin Tablo Örneği

TBLFILES	
Bileşen No	Yol
0-1111112	c:\yığıt.java
0-2222234	c:\proje\agaç.cpp

Bir bileşen sunucuda birbiri ile ilişkili bileşenler de olabilecektir. Bunun için Tablo 7 yapısındaki Tablo düzenlenmiştir.

Tablo 7. Birbiri ile Bağlantılı Bileşenleri Tutan Tablo

TBLRELATED	
Bileşen No	Bileşen No
0-1122334	0-98887765
0-2223345	0-67888909
0-4332223	0-88876555

3. 3. Gerçekleştirim

Bileşen sunucular, kullanıcı arayüzleri ve koordinatör sunucunun Visibroker ile gerçekleştirimi belirli adımlardan oluşmaktadır. Aşağıda bu adımlar ve her adım sonucunda hangi Java sınıflarının oluştuğu açıklanmaktadır.

Tüm bu adımlardan önce, Visibroker'ın kurulumundan sonra C:\Inprise\vbroker\lib dizininde yer alan *.jar dosyaları projenin gerçekleştirileceği dizine açılmalıdır. Aksi durumda IDL kodları derlenemeyecektir.

Adım-1:

IDL tanımlamaları yapılmalıdır. IDL tanımlamaları, MultiCoordinator adı verilen bir modül içerisindedir.

Adım-2:

IDL kodları derlenir. Visibroker için "idl2java" ismindeki bir derleyici kullanılır. Derleme sonucunda 3 çeşit çıktı oluşur. Bunlar, istemci koçanları (client stubs), sunucu iskeletleri (server skeletons) ve dil bağımlı örnek bir sınıftır.

Adım-3:

Bu adımda daha önceden IDL tanımlamaları yapılan sunucunun gerçekleştirimi yapılır. Bu gerçekleştirim yapılırken, kolaylık sağlaması amacı ile adım-2'de söz edilen dil bağımlı örnek sınıf kullanılarak daha da geliştirilebilir.

Adım-4:

Sunucuların etkinleştirilmesi başka bir ana program içinden gerçekleşmelidir. Bu adımda bu ana program yazılır.

Adım-5:
İstemci kodlar yazılmalıdır. İstemci kodlar, hem yeniden kullanıcıların sistemi kullanmasını sağlayan applet'ler hem de bileşen sunuculardaki veri tabanlarına JDBC (Java Database Connectivity) ile erişimi sağlayan Java uygulamalarından oluşmaktadır.

Adım-6:

Tüm sunucu ve istemci kodlar derlenir ve işletilmeye hazır duruma gelir.

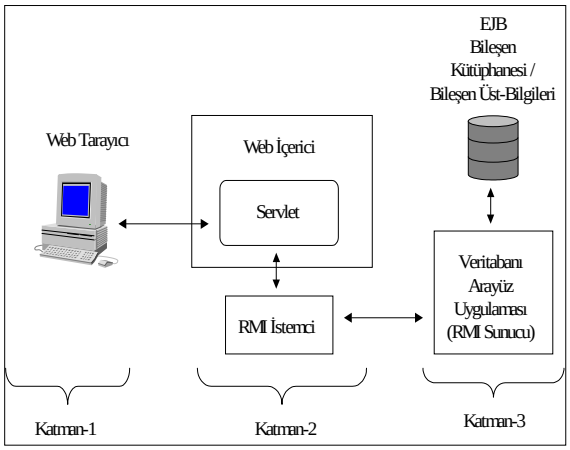
Adım-7:

Programlar çalıştırılır. Bunun için ilk önce osagent (smartagent) servisi çalıştırılmalıdır. Daha sonra koordinatör sunucu çalıştırılır. Koordinatör sunucu çalıştıktan sonra gatekeeper servisi çalıştırılır. Daha sonra bileşen sunucular çalıştırılır. Artık İnternet üzerinde herhangi bir tarayıcıdan (browser) Web

sunucumuza ulaşıp kullanıcı applet'leri çalıştırılarak sisteme erişilebilir.

4. BTYK ORTAMININ JAVA 2 PLATFORMUNDA GERÇEKLEŞTİRİMİ

BTYK ortamının günümüzün en popüler Web uygulaması geliştirme ortamlarından birisi olan Java 2 platformunda gerçekleştirilmesine yönelik bir mimari Şekil 3'te verilmiştir. Bu mimariye ilişkin uygulamanın kodlanması özellikle EJB bileşen kütüphaneleri aranmasına yönelik olarak yapılmıştır. Bununla birlikte, aynı mimari her türlü yazılım bileşeni içeren kütüphaneler için de aynı şekilde kullanılabilecektir.



Şekil 3. BTYK ortamının Java 2 platformunda gerçekleştirimi

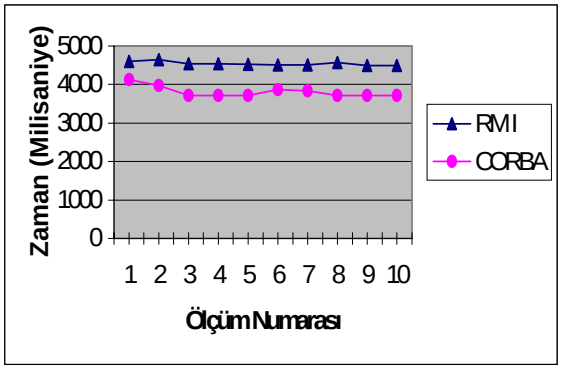
Şekil 3'ten de görüldüğü gibi, geliştirilen uygulama üç katmanlı tipik bir Web uygulamasıdır. Kullanıcının Web tarayıcıdan HTML formu yardımıyla girdiği anahtar kelimeler ile sorgu oluşturulup, sorgu sunucu tarafta Web içerici içindeki bir servlet'e gönderilmektedir. Bu servlet, aynı zamanda Java RMI (Remote Method Invocation) istemcisi rolündedir. Java RMI istemcisi rolündeki bu servlet daha sonra sorguyu uzak veritabanı önünde arayüz görevi gören ve Java RMI sunucusu rolündeki Java uygulamasına geçirmektedir. Bu Java uygulaması veritabanındaki üst-bilgileri sorgulamakta ve sonuçlar istemci rolündeki servlet'e gönderilmektedir. Servlet de gelen yanıtları Web tarayıcıya aktararak kullanıcının görmesini sağlamaktadır.

Bu uygulamada kullanıcı arayüzleri çok karmaşık olmadığı için JSP teknolojisi kullanılmamıştır. Kullanıcıya gösterilecek olan bilgiler, servlet içerisinde HTML biçeminde hazırlanarak Web tarayıcıya aktarılmıştır.

Geliştirilen uygulama, Jakarta Tomcat sunucusu kullanarak Web üzerinden erişilebilir hale getirilmiştir. Bu uygulamaya erişmek için http: //155.223.24.43/Muh06EJBLibrary.html adresi kullanılabilecektir. Ancak, bu uygulamaya atanmış özel bir sunucu bilgisayar olmadığı için, bağlantıyı sınamadan önce sistemin hazırlanması amacıyla makale yazarları ile iletişime geçilmelidir.

5. CORBA VE JAVA 2 TEMELLİ MİMARİLERİN KARŞILAŞTIRILMASI

CORBA ve Java 2 platformu temelli mimarilerde hız karşılaştırılması daha önceki bir çalışmamızda yapılmıştır (Türksever ve Erdur, 2001). Söz edilen çalışmada, hem CORBA kullanarak hem de Java 2 platformunda Java RMI teknolojisi kullanılarak dosya sunucu nesneler oluşturulmuş ve hız ölçümü için yaklaşık 49000KB uzunluğunda bir dosyanın transferi için on kez zaman ölçümü yapılmıştır. Elde edilen sonuçlar Şekil 4'te görülmektedir.



Şekil 4. Java RMI ve CORBA ile dosya sunucudan dosya transfer zamanları (Türksever ve Erdur, 2001)

Şekil 4'teki grafik incelendiğinde tipik bir dosya transferi işlemi için CORBA ortalaması 3808.2 milisaniye iken, Java RMI ortalamasının 4544.6 milisaniye olduğu görülmektedir. Aradaki yaklaşık 700 milisaniyelik fark çok büyük bir fark değildir. O nedenle, CORBA veya Java 2 ortamları arasında seçim yapmak için diğer kriterlerin incelenmesi daha fazla önem arz etmektedir.

CORBA ortamlarındaki dezavantaj, sunucu nesnelere ilişkin arayüzlerin tamamen ayrı bir arayüz tanımlama dili (interface definition language – idl) ile yazılması gereğidir. Bu da geliştiriciler için fazladan bir eğitim gerektirebilmektedir. Bunun

aksine, Java 2 ortamındaki her türlü kodlama Java dilinin sunduğu sınıf ve arayüz yapıları ile gerçekleştirilebilmektedir. Saf Java yaklaşımı olarak adlandırılan bu yaklaşım bir avantaj teşkil etmektedir. Ancak, farklı programlama dillerinde yazılmış istemci ve sunucuların bağlantısı sağlamak istendiğinde CORBA daha avantajlıdır. Örneğin, Smalltalk dilinde gerçekleştirilmiş bir istemci, C++ dilinde gerçekleştirilmiş olan bir sunucu nesneden servis alabilmektedir. Bu özellik, IDL gibi ayrı bir arayüz tanımlama dili olması nedeni ile sağlanmaktadır. IDL ile yazılmış tanımlar IDLtoSmalltalk, IDLtoC++, vb. derleyicilerden geçirilerek istemci "stub" ve sunucu "skeleton" oluşturulmaktadır. Bu "stub" ve "skeleton" kodlar daha sonra istemci ve sunucu arasındaki bağlantıyı sağlamaktadır.

6. SONUÇLAR VE TARTIŞMA

Bu çalışmada, İnternet üzerinde yeniden kullanılabilir yazılım bileşenleri aramaya yönelik CORBA temelli ve Java 2 platformu temelli iki mimari ortaya konmuş ve karşılaştırılmıştır. İki mimari arasında hız açısından belirgin bir fark olmadığı belirlenmiştir. Tüm programlamanın Java dili yapılmak istendiği durumlarda Java 2 platformu avantaj sağlamaktadır. Bunun dışında günümüzdeki genel eğilim, taşınabilir Web uygulamaları geliştirmede Java 2 platformunun kullanılması yönündedir . Bu nedenle, Java 2 platformu temelli uygulamaların sayısı CORBA'ya göre giderek artmaktadır.

7. KAYNAKLAR

Anonymous 1997. Microsoft Visual Basic Component Tools Guide. Microsoft Corporation, 796p.

Browne, S. and Moore, J. 1997. "Reuse Library Interoperability and the World Wide Web, in symposium on Software Reusability", ACM Press, USA.

Diaz, R. and Freeman, P. 1987. Classfication of Software For Reusability, IEEE Software, January 1987.

Erdur, R. C. and Türksever M. 1999. Design and Implementation of a Java-based Reuse Support Tool, in the proceedings of ISCIS-XIV, 335-343, Ege University, 1999.

Griss, M. L. 1993. Software Reuse from Library to Factory, IBM Systems Journal, 32(4):548-566, 1993.

Orfali, R., Harkey, D., and Edwards, J. 1997. Instant CORBA, John Wiley & Sons, USA.

Orfali, R. and Harkey, D. 1998. Client/Server Programming With Java & CORBA, John Wiley & Sons, USA.

Pedrick, D., Weedon, J., Goldberg, J., and Bleifield E. 1998. Programming with Visibroker, Wiley Computer Publishing, USA.

Roman, E., Ambler, S., and Jewell, T. 2002. Mastering Enterprise Java Beans, John Wiley & Sons, Inc. USA.640p.

Türksever, M. ve Erdur, R. C. 2001. Farklı Bakış Açılarından Java RMI ve CORBA'nın Karşılaştırılması. Pamukkale Üniversitesi, Mühendislik Bilimleri Dergisi, 7 (1), 63-70.