EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

SERVİS ODAKLI MİMARİ

Serkan ÜSTÜNDAĞ

Bilgisayar Mühendisliği Anabilim Dalı

Bilim Dalı Kodu : 619.01.00
Sunuş Tarihi: 18 Aralık 2006

Tez Danışmanı: Prof. Dr. Oğuz DİKENELLİ

Bornova-İZMİR

II

Serkan ÜSTÜNDAĞ tarafından yüksek lisans tezi olarak sunulan "Servis Odaklı Mimari" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 18 Aralık 2006 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza:**
Jüri Başkanı	**:**	.....
Raportör Üye	**:**	.....
Üye	**:**	.....

IV

ÖZET
SERVİS ODAKLI MİMARİ

ÜSTÜNDAĞ, Serkan
Yüksek Lisans Tezi, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Oğuz DİKENELLİ
18 Aralık 2006, 119 sayfa

Son yıllarda; nesneye yönelik dillerin kullanımının yaygınlaşması ve dağıtık programlama uygulamalarının XML ve web servisleri teknolojileri ile kolaylaşması sonucunda, fonksiyonların servisler halinde sunulmasını benimseyen servis odaklı mimari, uygulama mimarileri arasında hızla önem kazanmaya başlamıştır.

Özellikle kurumsal uygulamalar arasında varolan entegrasyon çözümlerinin maliyetleri, uygulamaların servis odaklı mimari prensiplerine göre geliştirilmesi durumunda önemli bir oranda azalmaktadır. Kurumlar servis odaklı mimari geliştirmeyi bilgi işlem stratejisi olarak belirlemeye başlamıştır. Servis Odaklı Mimarinin prensiplerinin iyi bilinmesi ve servislerin koordinasyonunun etkin bir şekilde nasıl sağlanacağının öğrenilmesi özellikle uygulama entegrasyonlarının kalitesini ve güvenilirliğini arttıracaktır.

Bu çalışmanın amacı, herhangi bir üretici firmaya bağlı kalmadan, iki uygulama arasındaki bir entegrasyon işleminin standartlara dayalı bir şekilde gerçekleştirilebileceğini ortaya koymaktır.

Anahtar Sözcükler: Servis, Web Servisleri, Entegrasyon, BPEL4WS, WSDL, Kurumsal Uygulama Entegrasyonları

ABSTRACT
SERVICE ORIENTED ARCHITECTURE

ÜSTÜNDAĞ, Serkan

MSc in Computer Engineering
Supervisor : Prof. Dr. Oğuz DİKENELLİ
18 December 2006, 119 pages

In the recent years, object oriented programming languages are getting to be used in common and the applications of distrubuted programming becomes easy to implement with the help of XML and web services technologies. As a results of this, service oriented architecture, which is based on presenting the functions within services, becomes popular in application architectures.

The usage of service oriented architecture principles descreases especially the cost of organizational integration solutions implementation. So, the organizations start to choose service oriented architecture as a strategy of information technologies. When the service oriented architecture principles and the way of effective service coordination are known well, the quality of integration process increases and this makes the process more reliable.

The aim of this work is to show that, regardless from the producer or organization, it is possible to do integration process between different applications in a standardized way.

Keywords : Service, Web Services, Integration, BPEL4WS, WSDL, Enterprise Application Integration

TEŞEKKÜR

Bu çalışma süresince danışmanlığımı yapan ve bana sürekli destek olan sayın danışmanım Prof. Dr. Oğuz DİKENELLİ ' ye, katkılarını hiç esirgemeyen annem ve babama, her zaman yanımda olan sevgili arkadaşım Sedef KARA' ya ve destek ve anlayışları için diğer tüm dostlarıma teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

1. GİRİŞ

Servis Odaklı Mimari bir bilgi işlem stratejisi olup, bu stratejinin amacı kurumsal yazılımlar içindeki farklı fonksiyonları karşılıklı çalışabilecek, standartlara dayanan, iş ihtiyaçlarını karşılamak üzere kolaylıkla tekrar kullanılabilen ve birleştirilebilen servisler haline sokmaktır.

Servisler, aslında mevcutta var olan fonksiyonların yada yeni oluşturulan fonksiyonların, belli prensipler göz önünde bulundurularak servis halinde sunulmasıdır. Bu kısa sürede gerçekleştirilebilecek bir hedef olamayacağından bir kurum ancak bunu stratejik olarak tasarlayıp, bu stratejiye ulaştıracak olan yol haritasını takip ederek zamanla hayata geçirebilecektir.

Günümüzde, varolan kurumsal uygulamalar ile yeni geliştirilen veya satın alınan uygulamaların birlikte çalışabilirliğini sağlamak organizasyonların en fazla yatırım yaptığı alanlardan biridir. Bu konuda farklı birçok firmanın geliştirdiği kurumsal uygulama entegrasyon araçları bulunmaktadır.

Servis Odaklı Mimari prensiplerine uyularak geliştirilen uygulamaların varolan diğer uygulamalar ve yeni geliştirilecek olan sistemlerle birlikte çalışabilirliğini sağlamak daha az efor gerektiren bir çalışma gerektirmektedir.

Bu çalışmada Servis Odaklı Mimari prensiplerine uygun olarak, BİMAR Bilgi İşlem Hizmetleri tarafından geliştirilmekte olan Yeni Nesil Acentelik (YNA) uygulamasının temel gereksinimlerinden yola çıkılarak, öncelikle Servis Odaklı Mimari prensiplerinden ve YNA sisteminin

uygulama çatısı olan ROTA' nın bu prensipleri nasıl sağladığından bahsedilecektir.

Sonraki kısımda ise günümüzde kullanılmakta olan kurumsal uygulama entegrasyon mimarileri anlatılacak, YNA sisteminde varolan entegrasyon çözümünden kısaca bahsedilecektir.

Son kısımda ise standartlara dayanan bir entegrasyon yapısı için BPEL4WS kullanımına örnek verilecek ve YNA sisteminde varolan bir entegrasyon, BPEL4WS kullanılarak yeniden geliştirilecektir. Standartlara dayanan entegrasyon çözümlerinde web servisleri önemli bir yere sahip olduğundan bu kısımda web servisleri hakkında kısa bilgiler verilecek, web servisleri ile süreç bütünlüğünü sağlamak için kullanılan standartlar hakkında anlatım yapılacaktır.

2. LİTERATÜR ÇALIŞMASI

Yapılan literatür taraması sonucunda Servis Odaklı Mimari özellikleri, İş Süreçleri Yönetimi ve Hareket problemi, Web Servisleri, İkinci Nesil Web Servislerinin iş süreçleri için getirdiği çözümler ve BPEL4WS, Kurumsal Entegrasyon Mimarileri gibi kavramlar hakkında detaylı bilgilere erişme olanağı bulunmuştur. Araştırma aşamasının ilk döneminde temel kavramlar üzerinde durulmuş, ilerleyen bölümlerde standartlara dayalı entegrasyon konusunda kullanılacak olan web servisleri genel mimarisi ile ikinci nesil web servislerinin özellikleri ve BPEL4WS konularında araştırmalar yapılmıştır.

2.1. Yeni Nesil Acentelik Sistemi

Yeni Nesil Acentelik (YNA), Arkas Holding'e bağlı acentelik şirketlerinin yazılım gereksinimlerini karşılamak için geliştirilmekte olan bir yazılım paketidir.

Bu paket içerisinde farklı işlevleri gerçekleştiren ve birbiri ile iletişim halinde olan büyük modüller bulunmaktadır. YNA sisteminin alt sistemleri tamamlandıkça kullanıma açılmaktadır. Bu yüzden tamamlanan alt sistemlerin, şu anda kullanılmakta olan AS400, SAP, CATLOGIC sistemleri ile ortak olarak çalışması gerekmektedir.

2.1.1. Temel Gereksinimler

YNA yazılımının karşılaması beklenen en temel gereksinimler şunlardır;

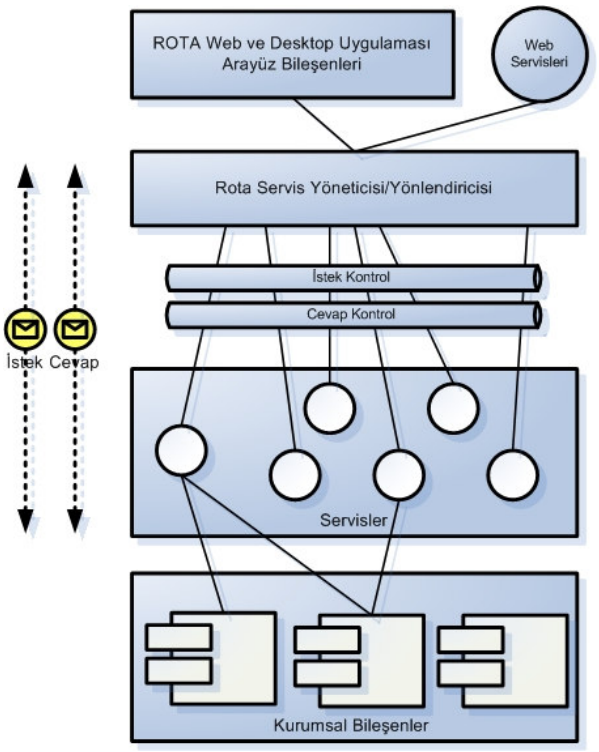
- Varolan kurumsal sistemler ile entegre olarak çalışabilmesi
- Müşterilerin sistemleri ile veri alışverişinde bulunabilmesi
- Kullanıcıların ve müşterilerin farklı arayüzlerden servis alabilmesi
- Değişikliklere kolay uyum sağlayabilen bir sistem olması
- Bakım maliyetinin çok yüksek olmaması

Tüm bu gereksinimlerin incelenmesi sonucunda bir sonraki bölümde genel prensipleri ve servis kavramı anlatılacak olan servis odaklı mimari yaklaşımının en uygun yaklaşım olduğuna karar verilmiş ve bu doğrultuda çalışmaya başlanmıştır. Bu kararın verilmesinde, özellikle sistemin varolan SAP ve AS400 kurumsal uygulamaları ile entegre bir şekilde çalışması gereksinimi önemli bir etken olmuştur.

2.1.2. ROTA Uygulama Çatısı

Rota uygulama çatısı katmanlı mimari prensiplerine uygun olarak önyüz, iş katmanı ve veri erişim ile ilgili temel sınıfları barındıran, katmanlar arasındaki iletişimin mesaj sınıfları ile sağlandığı, servis prensiplerine uygun olarak iş metotlarının, birbirine bağımlı olmadan, yeniden kullanılabilir ve web servisleri ile dış sistemlere açılabilen

servisler olarak geliştirilmesine olanak tanıyan bir uygulama çatısıdır. Şekil 2-1' de Rota uygulama çatısının genel mimarisi görülmektedir.



Şekil 2-1 Rota Uygulama Çatısı Katmanları

İş Katmanı (Servisler ve Kurumsal Bileşenler):

İş katmanında yer alan operasyonlar iş varlıkları adı verilen nesnelerde yer almaktadır. Servis olarak nitelendirilebilecek olan işler bu nesnelerde yer alan operasyonları kullanabilmektedir.

Rota uygulama çatısı içerisinde yer alan temel “Transaction” sınıfları kullanılarak yaratılan iş nesneleri, YNA içerisindeki servisleri temsil etmektedir. Daha önceden de belirtildiği gibi bu servisler web servisleri ile dış dünyaya açılmaktadır.

Servisler, işi gerçekleştiren metot çağrımlarında, bir sonraki kısımda anlatılacak olan istek ve cevap nesnelerini almaktadır. İstekteki verileri işleyerek oluşan sonuçları cevap nesnesine aktarmaktadır.

Bu servisler;

- Hangi istemci için iş yaptıklarını bilmezler, istemciler servislerin ne şekilde iş yaptığını bilmezler, böylelikle servislerin çalışma mantığını gizlemesi prensibi sağlanmış olmaktadır.
- Bir servis farklı bir servise direk olarak erişemez, böylelikle servis odaklı mimari prensiplerinden olan gevşek bağlılık sağlanmaktadır.
- Servisler dağıtık hareketleri desteklemez, bu sayede servis odaklı mimari prensiplerinden olan durumsuz servisler ilkesine uyulmakta ve bağımsız çalışabilen servisler oluşturulabilmektedir.
- Servisin operasyonu içerisinde yer alan bütün işlemler “ACID” özellikli bir hareket kapsamında yönetilmektedir, bu sayede servis odaklı mimari prensiplerinden olan durumsuz servisler ilkesine uyulmaktadır.

Tüm bu özellikler yeniden kullanılabilirliği arttırmakta ve servislerin bir süreç içerisinde birlikte kullanılabilmesini kolaylaştırmaktadır.

İletişim Katmanı:

Katmanlar arasındaki iletişim istek ve cevap nesneleri aracılığı ile sağlanmaktadır. Önyüzden girilen bilgiler (web arayüzü, mobil cihaz arayüzü veya bir web servisi vb.) istek nesneleri aracılığı ile hizmet verecek olan uygun servise aktarılmakta, servisin işletilmesi sonucunda oluşan bilgiler cevap nesnesi aracılığı ile istekte bulunan varlığa geri dönmektedir.

Rota uygulama çatısında yer alan temel mesaj sınıflarını ata olarak kullanan, uygulama servisine özel mesaj sınıfları üretilmektedir.

Mesajlar;

- Tek bir komut ile XML' e dönüştürülebilmektedir,
- Serileştirilebilir sınıflardır,
- Uygulamaya özel temel sınıf yaratılmasına olanak sunmaktadırlar,
- Başlık ve gövde olmak üzere iki kısımdan oluşmaktadırlar.

Servis Yönetimi ve Yönlendirme Katmanı:

Rota uygulama çatısı, farklı arayüzlerden (grafiksel arayüz, web servisleri vb.) istek mesajı gelmesine olanak sağlaması açısından bir aktarım katmanı içermektedir. Bu aktarım katmanının temel görevleri

istek ve cevap nesnelerini yaratmak, çalıştırılması gereken servisi bulmak ve bu servisi tetiklemektir.

Ayrıca bu katman istek ve mesaj sınıfları ile ilgili yapılacak olan ortak işlemleri gerçekleştiren “Pipeline” adı verilen yapıları tetiklemekle görevlidir. “Pipeline” varlıkları günlükleme, yetki kontrolü gibi işlemleri yapan daha küçük yapıları içermektedir. Bu küçük yapılar farklı amaçlar için oluşturulabilecek olan “Pipeline” larda yeniden kullanılabilmektedirler.

2.2. Servis Odaklı Mimari

Servis Odaklı Mimari bir bilgi işlem stratejisi olup, bu stratejinin amacı kurumsal yazılımlar içindeki farklı fonksiyonları karşılıklı çalışabilecek, standartlara dayanan, iş ihtiyaçlarını karşılamak üzere kolaylıkla tekrar kullanılabilen ve birleştirilebilen servisler haline sokmaktır.

Servisler, aslında mevcutta var olan fonksiyonların yada yeni oluşturulan fonksiyonların, belli prensipler göz önünde bulundurularak servis halinde sunulması ile oluşturulmaktadır. Bu kısa sürede gerçekleştirilebilecek bir hedef olamayacağından bir kurum ancak bunu stratejik olarak tasarlayıp, bu stratejiye ulaştıracak olan yol haritasını takip ederek zamanla hayata geçirebilecektir (Hüdayioğlu, 2006).

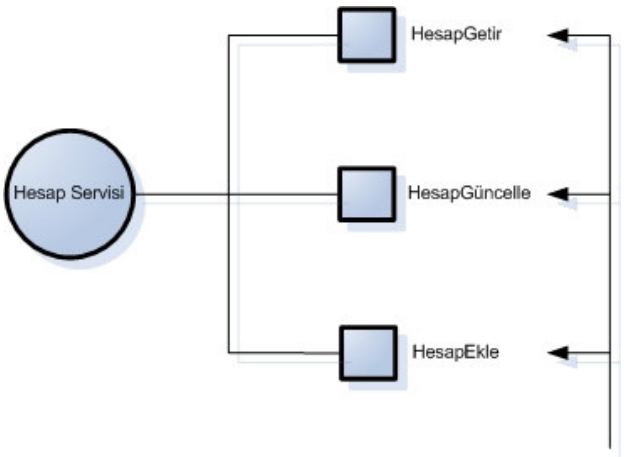
2.2.1. Servis Odaklı Mimarinin Genel Özellikleri

Servis odaklı mimarinin genel özellikleri şu şekilde listelenebilir:

- Yeniden Kullanılabilir Servisler

Servis odaklı mimari tüm servislerin yeniden kullanılır olmasını öngörmektedir. Böylelikle kurum kaynakları en verimli şekilde kullanılabilmektedir. Tasarım standartları uygulandığı taktirde tüm servisler yeniden kullanılabilir olacaktır.

Servisler, olması gereken büyüklükte parçalar olarak tanımlanmalıdır. İdeali bulmak için, duruma göre değerlendirme yapmanın yanında erkenden yöntem belirlenmesi tavsiye edilmektedir.



Şekil 2-2 Servis yeniden kullanılabilir operasyonlar sunmakta

Mesajlaşma yapısı, servis yeniden kullanılabilirliğini dolaylı olarak desteklemektedir. Mesajlar işlem komutları ve iş kurallarını içermekte ve alıcı servislere nasıl işlenmeleri gerektiğini belirtmektedir. Böylelikle daha genel servislerin oluşturulmasına olanak sağlanmaktadır (ServiceOrientation, 2006).

- Servisler Ortak Bir Kontratı Paylaşmaktadır

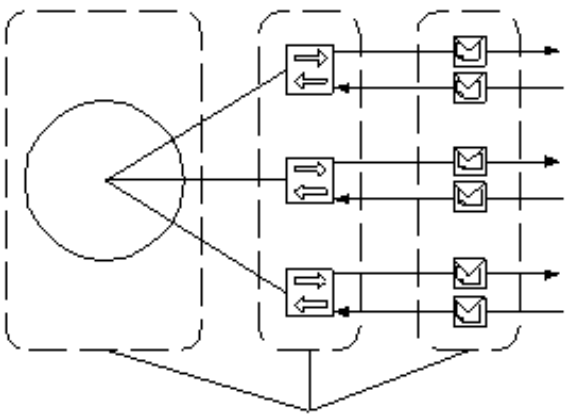
Servisler mevcutta tanımlanmış olan standartlara uygun şekilde geliştirilmelidirler. Örnek verilirse, USB' den önce her donanımın kendine göre bir ara yüzü vardı ve o donanımı kullanabilmek için adaptörler alınmak zorundaydı. Halbuki USB' nin bir standart olarak ortaya çıkmasıyla beraber, üreticiler ürünlerini USB' ye uyumlu üretmeye başladılar.

Böylelikle kullanıcılar için bir donanımı alıp onu hemen kullanmaya başlamak mümkün olmuştur. Yazılım dünyasının da SOA ile yakından ilişkili web servisleri standartlarını benimsemesiyle beraber, uygulamalar daha rahat birlikte çalışabilir hale gelmiştir.

Servis kontratları aşağıdaki tanımlamaları standart bir şekilde yapmaktadır;

- Servis uç noktası,
- Servis operasyonları,
- Her operasyon tarafından desteklenen girdi çıktı mesajları,
- Servisin ve operasyonun kuralları ve özellikleri

Kontratlar genellikle servis odaklı mimarinin temel kısımlarını tanımlamaktadır. İyi bir servis kontratı aynı zamanda servisin belirli bir görevi nasıl yaptığını açıklayan mantıksal bilgi de sağlamalıdır.



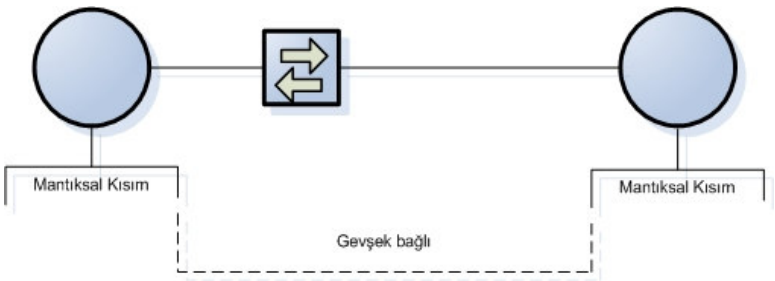
Şekil 2-3 Servis, kontrat tarafından tanımlanmaktadır

Kontratlar servisi, operasyonu ve mesaj bileşenlerini tanımlamaktadır. Kontratlar servisler arasında paylaşıldığı için tasarımları oldukça önemlidir. Bu yüzden, ilk kez yayınlandıktan sonra dikkatlice sürüm altına alınmalı ve yönetilmelidirler.

Web servisleri uygulama çatısında bulunan servis tanımlama dokümanları (WSDL, XSD) servisin programsal olarak nasıl erişildiğini gösteren iletişim kontratlarıdır (ServiceOrientation, 2006).

- Servisler Gevşek Bağlıdır

İstekler sürekli değiştiğinden dolayı uygulama ortamları da sürekli değişmektedir. Bu değişimlerin önceden planlanmasına imkan yoktur. Değişimlere kısa sürede cevap verebilmek açısından servis odaklı mimari doğru bir seçimdir.

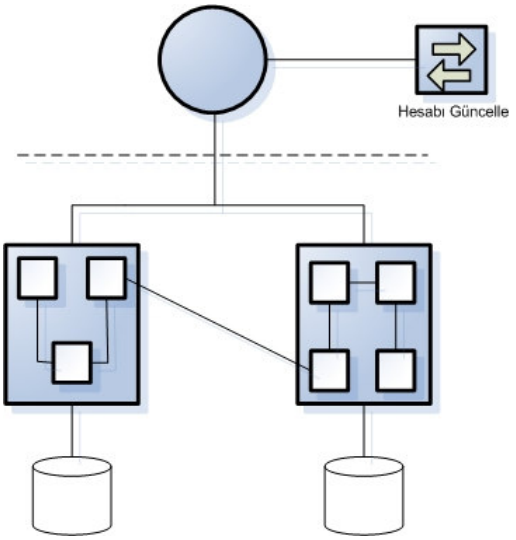


Şekil 2-4 Servisler gevşek bağlıdır

Servislerin bağımlılıkları, servis kontratları kullanılarak azaltılmaktadır. Servis kontratları sayesinde önceden tanımlanmış parametrelerle iletişim kurulabilmektedir (ServiceOrientation, 2006).

- Servisler Çalışma Mantığını Gizlemektedir

Servisler uygulama detaylarını dış dünyadan saklayarak kara kutu gibi davranmaktadırlar. Servisin çalışma mantığının herhangi bir sınırı yoktur. Bir servis basit bir işi gerçekleştirebilir yada bir otomasyon çözümünde arabirim görevinde bulunabilir. Kullanacağı kaynaklar konusunda da herhangi bir kısıtlama yoktur. Servis isterse uygulama mantığını iki ayrı sisteme açabilir (ServiceOrientation, 2006).



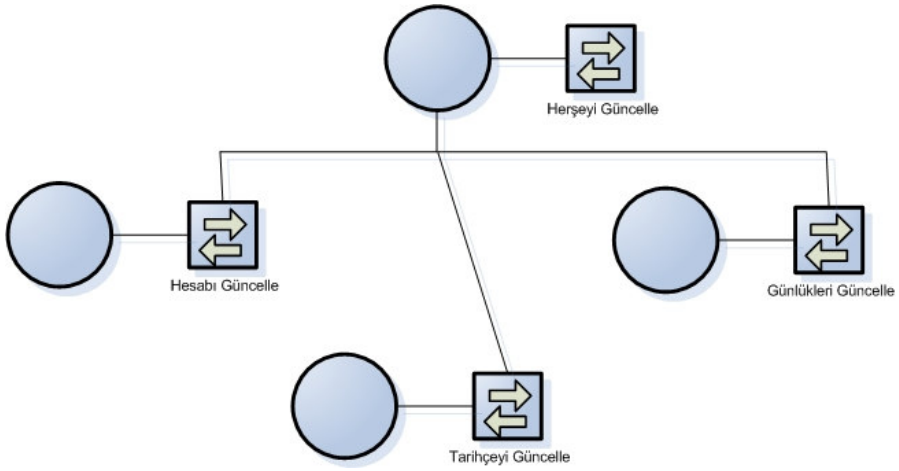
Şekil 2-5 Servisler çalışma mantıklarını gizlemektedirler

- Servisler Birleştirilebilir

Bu prensibin temel amacı, servislerin ihtiyaç duyulduğu zaman, başka bir servisin iş sürecinde katılımcı olarak yer almasını sağlamaktır.

Birleştirilebilirliği vurgulayan temel servis odaklı mimari ilavesi orkestrasyondur.

Şekil 2-6' da bir süreç servisi katılımcı servisleri birleştirmekte ve süreci kontrol etmektedir. Farklı güncelleme görevlerine sahip olan servisler, bir süreç mantığı içerisinde her şeyi güncellemekten sorumlu diğer bir servis tarafından yönetilmektedir (ServiceOrientation, 2006).

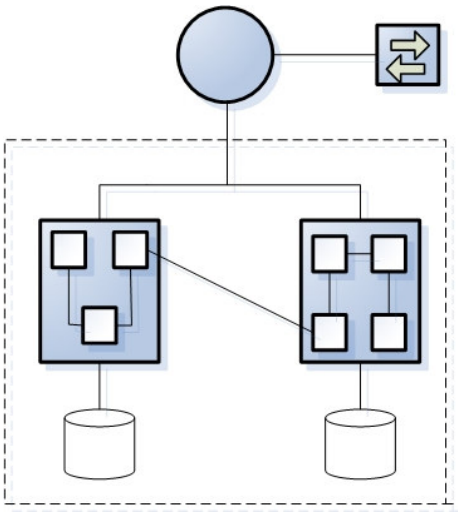


Şekil 2-6 Her şeyi güncelle, servis birleşiminden oluşmakta

- Servisler Bağımsız Çalışabilmelidir

Bir servis başka bir servisten bağımsız ve etkilenmeden çalışabilmelidir. Yani bir servisi kullanabilmek için başka bir servisin orada hazır olmasına gerek olmamalıdır.

Servislerin bağımsız çalışması, uygulama servislere ayrılmaya çalışılırken, hangi operasyonların bir servis altına toplanması gerektiğine karar verilmesi sürecinde temel belirleme koşuludur (ServiceOrientation, 2006).



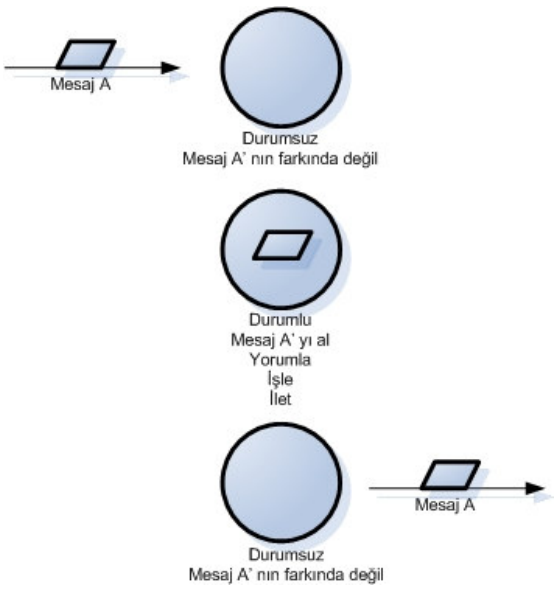
Şekil 2-7 Özerk servisler kendi kaynaklarını kontrol etmektedir.

- Servisler durumsuzdur

Servisler durum bilgisi yönetimini ve süresini en aza indirmelidirler. Durum bilgisi güncel uygulamada veriye özeldir. Servis bir mesajı işlerken geçici olarak durum içerir. Eğer uzun süre duruma

sahip olması gerekirse diğer servislerin isteklerine cevap vermesine engel olunacaktır.

Yeniden kullanılırlığı ve ölçeklenebilirliği arttırdığından dolayı, durumsuzluk servisler için tercih edilen bir koşuldur (ServiceOrientation, 2006).



Şekil 2-8 Servisin mesajı işlerken durum içermesi

- Servisler keşfedilebilir

Servisler çalışma zamanında dinamik olarak keşfedilmelidir. Servisin müşterisi kayıtçıyı tarayarak servisi çalıştırmak için gerekli olan

tüm bilgileri bulmaktadır. Servise bağlanmak için derleme bağımlılığı yoktur (Güner, 2005).

2.2.2. Servis Odaklı Mimari Bileşenleri

Servis Odaklı Mimari bileşenleri şu şekilde listelenebilir:

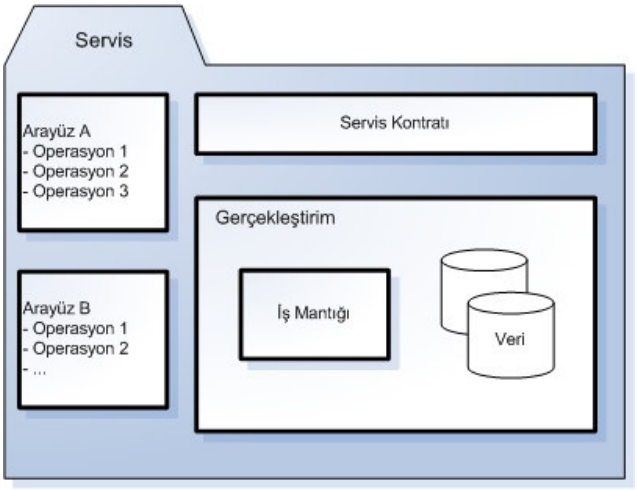
- Uygulama Önuçları

Servis odaklı mimarinin aktif oyuncuları durumundadırlar. Bir web uygulamasındaki gibi grafiksel kullanıcı arayüzü yada son kullanıcı ile etkileşimde olan zengin içerikli istemci uygulama önuçları farklı tiplerdeki örnekleridir. Uygulama önucu olmak için mutlaka son kullanıcı ile etkileşim halinde olmak şart değildir. Periyodik olarak bir işlemi başlatan "batch" programlar ve uzun süreli işlemler de uygulama önuçlarıdır.

Uygulama önuçları her zaman bir işlemi başlatır ve sonuçlarını alır. Geleneksel çok katmanlı uygulamalardaki üst katmanlara benzemektedir (Enterprise SOA, 2004).

- Servisler

Servis, üst seviye iş mantığını barındıran yazılım bileşenidir. Bir servisi oluşturan parçalar şunlardır (Enterprise SOA, 2004).



Şekil 2-9 Genel bir servis görüntüsü.

Kontrat: Amacı fonksiyonellik, kısıtlar ve servisin kullanımı hakkında tarifler sağlamaktır. Kontratlar servisin tipine göre değişebilmektedir. Zorunlu olmayan tek elemanı dillere bağlı olan arayüz tanımıdır. Zorunlu olmamasına rağmen arayüz tanımı yapılması durumunda, teknolojiden bağımsız servisler yaratılabilecektir. Kontrat fonksiyonellik ve parametreler hakkında da detaylı bilgiler bulundurabilmektedir (Enterprise SOA, 2004).

Arayüz: Servisin fonksiyonelliği, servise bağlı olan istemcilere arayüz aracılığı ile sunulmaktadır. Arayüz tanımı kontratın bir kısmı olmasına rağmen fiziksel gerçekleştirimi, servisin istemcilerine dahil edilen koçanlardan (“stub”) oluşmaktadır (Enterprise SOA, 2004).

Gerçekleştirim: İş mantığını ve uygun veriyi sağlamaktadır. Programlar, konfigürasyon verisi, veritabanları gibi varlıklardan oluşmaktadır (Enterprise SOA, 2004).

İş Mantığı: Servis tarafından kapsanan iş mantığı gerçekleştirimin bir parçasıdır. Arayüzler aracılığı ile erişilebilir hale getirilir (Enterprise SOA, 2004).

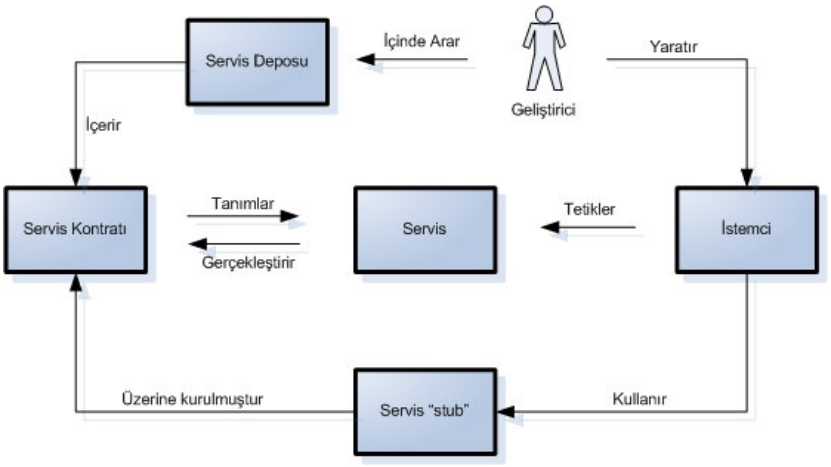
Veri: Servisler veri içerebilmektedir.

Daha önceden de belirtildiği gibi, servisler sadece uygulamanın alt katmanlarının kodunu içermemektedir. Her servis, üst seviye bir iş mantığı taşıyan varlıktır. İstemcinin bakış açısıyla servis bir kara kutudur (Enterprise SOA, 2004).

- Servis Ambarı

Servislerin yaratıldıkları proje dışından erişilebilir ve bulunabilir olmaları gerekirse, bu servislerin bulunması ve kullanım bilgilerinin sağlanması servis ambarının görevidir (Enterprise SOA, 2004).

Birçok bilgi kontratlarda yer aldığından servis ambarı fiziksel yer sağlayıcı, iletişim noktaları, güvenlik ve kısıtlar gibi konularda ek bilgiler sunmaktadır (Enterprise SOA, 2004).



Şekil 2-10 Servis Odaklı Mimari bileşenleri arasındaki etkileşim.

- Servis Yolu

Servis odaklı mimarinin tüm katılımcılarını (servisler ve uygulama önuçları) birbirine bağlamaktadır. Bir uygulama önucu, bir servisin fonksiyonunu kullanmak isterse bunu servis yolu mümkün kılmaktadır (Enterprise SOA, 2004).

Temel özellikleri şunlardır (Enterprise SOA, 2004):

- SOA katılımcılarını birbirine bağlar

- Farklı işletim sistemlerinde yer alan farklı dillerle yazılmış katılımcıları birbirine bağlar

- Günlükleme, güvenlik, mesaj değişimi veya hareketler gibi teknik hizmetler de sağlar.

2.2.3. Servis Tipleri

- Uygulama önuçları

Servis olmamalarına rağmen SOA' nın aktif elemanlarındandır. Tüm iş süreçlerini başlatır ve sonuçlarını alırlar. Grafiksel arayüzler ve küme ("batch") işlemler uygulama önucu örnekleridir (Enterprise SOA, 2004).

- Temel Servisler

Servis odaklı mimarinin temelidir. Veri odaklı yada mantık odaklı olarak ikiye ayrılmaktadırlar. Veri odaklı servisler verinin saklanması, verinin elde edilmesi, kilitleme mekanizması ve hareket yönetimi gibi kavramları içermektedirler. Mantık odaklı servisler ise içerisinde karmaşık iş kuralları ve karmaşık algoritmalar barındırmaktadır. Birçok servis hem veri hem de iş mantığı ile ilgili olduğundan kesin bir ayrım yapılamamaktadır (Enterprise SOA, 2004).

- Aracı Servisler

Mimarideki teknik uyumsuzlukları ve tasarım ayrıklıklarını köprü görevi görüp çözen durumsuz servislerdir. Servis odaklı mimaride hem istemci hem de sunucu konumundadırlar (Enterprise SOA, 2004).

- Süreç Merkezli Servisler

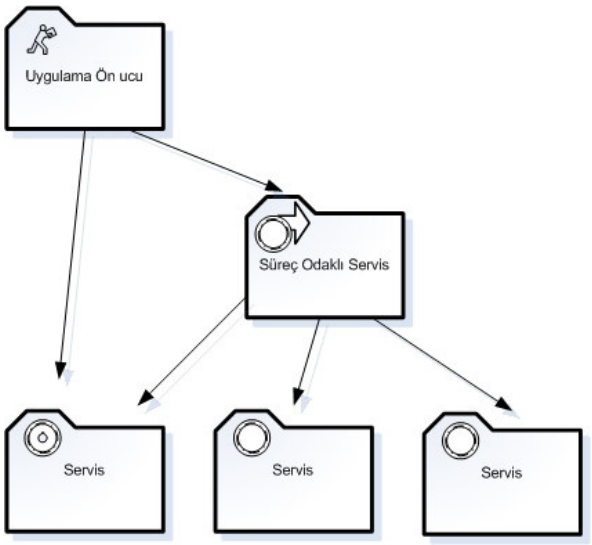
Organizasyonun iş süreci bilgisini içermektedirler. Servis odaklı mimaride hem istemci hem de sunucu görevindedirler ve sürecin durumunu yönetmektedirler. Etkin bir gerçekleştirim için dikkatli bir tasarım gerektiren servislerdir. Aracı servislerden temel farkı, istemciler için sürecin durumunu yönetmek zorunda olduğundan durum bilgili olmasıdır (Enterprise SOA, 2004).

Her mimaride olduğu gibi yeni eklenen her elemanın karmaşıklığı arttırdığı bilinse de süreç merkezli servislerin önemli faydaları vardır (Enterprise SOA, 2004):

Süreç karmaşıklığını kendi bünyesinde barındırmaktadır: Sürecin kontrolünün karmaşıklığını uygulamadan gizlemektedir. Farklı takımların sunum ve işlemlerin gerçekleştiriminde aynı anda çalışmalarına olanak sunmaktadır (Enterprise SOA, 2004).

Yük dengelenmesini olanaklı kılmaktadır: Uygulama önuçları sunuma odaklanacağından servis işlemleri farklı makinelerde çalıştırabilecektir.

Ayrı süreç mantığı: Süreç mantığı dikkatli bir şekilde çekirdek iş mantığından ve diyalog kontrolünden ayrılmalıdır. Süreç merkezli servisler düzgün bir şekilde belirlenirse, temel servislere de iş mantığı atanabilir. Kullanıcı ile diyalogda önuçlar tarafından yürütülür. Süreç mantığının ayrılması etkin iş süreci yönetimi için ön koşuldur (Enterprise SOA, 2004).



Şekil 2-11 Süreç Odaklı Servis

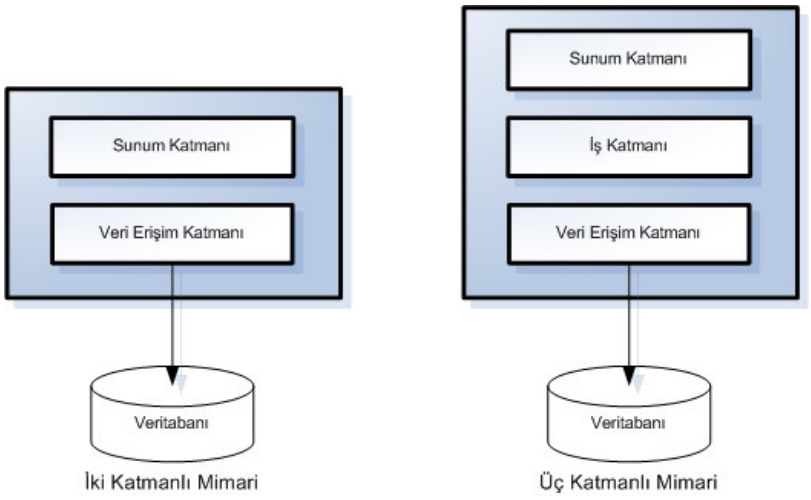
- Genel Kurumsal Servisler

Şu ana kadar anlatılan servisler organizasyon sınırları içerisinde yer alan servislerdir. Genel servisler ise ortaklar ve müşteriler ile iletişim halinde olan servislerdir. Örneğin bir nakliye şirketi müşterilerinin yüklerini takip edebilmesi için bir servis sunabilir (Enterprise SOA, 2004).

2.2.4. Servis Odaklı Mimari Katmanları

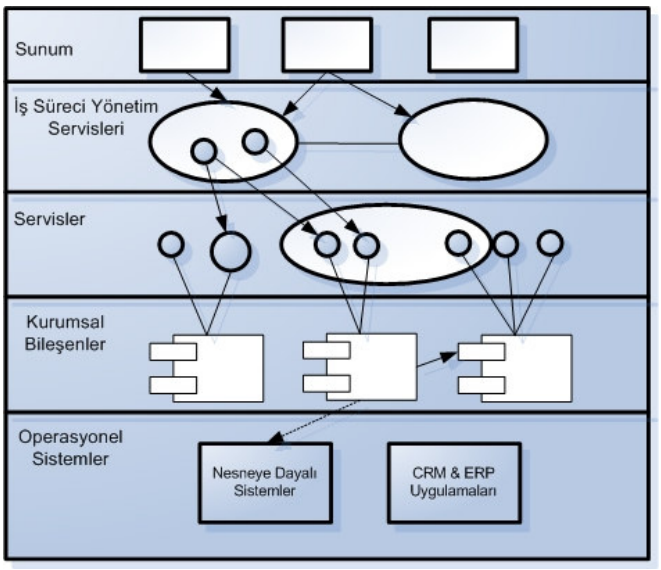
İstemcinin veritabanı veya ağa, arada herhangi bir mantıksal katman kullanmadan eriştikleri eski sistemler iki katmanlı olarak yapılandırılmışlardır. Bu yaklaşım küçük boyutlu uygulama veya ön ürün geliştirmede hala kullanılmaktadır.

Günümüzde en çok kullanılan uygulama geliştirme modeli üç katmanlı mimari yapısına dayanmaktadır. Üç katmanlı mimaride istemci ile veri katmanları arasında ek bir katman yer almaktadır. Bu katman iş mantığı katmanı olarak isimlendirilmekte ve kod yalıtımı ile uygulama paylaşımı sağlamaktadır.



Şekil 2-12 İki ve üç katmanlı yazılım mimarileri

SOA çok katmanlı uygulama geliştirme modelini desteklemektedir. Servisler fonksiyonelliği sağlamak için görevli olan bileşenlerin üst katmanını oluşturmaktadır (Güner, 2005).



Şekil 2-13 Servis Odaklı Mimarinin katmanları

2.3. Uygulamalar Arası Birlikte Çalışılabilirlik

Birçok kurumun, farklı üreticilerin teknolojileri ile geliştirmiş olduğu ve farklı platformlarda çalışan çeşitli uygulamaları bulunmaktadır. Bu uygulamalar arasındaki birlikte çalışılabilirlik sorunlarının çözümü için geleneksel Kurumsal Uygulama Entegrasyon (EAI) ürünleri ortaya çıkmıştır. Son zamanlarda ise bu ürünler üzerinde birçok geliştirmeler yapılmıştır. Sonuç olarak entegrasyon uygulamaları tek bir üreticiye bağlı hale gelmiş ve sıkı bağlı bileşenler ortaya çıkmıştır.

Bir üreticiye ait olan bu entegrasyon çözümlerinin yönetimi kurumlara önemli bir yük getirmekte ve yetişmiş personel gerektirmektedir.

İki uygulama arasındaki birlikte çalışabilme ihtiyacı bir iş süreci olarak düşünülebilir. Bu tipteki iş süreçlerinde en büyük sorun süreç bütünlüğünün sağlanmasıdır. Kurumsal Uygulama Entegrasyon mimarilerini incelemeden önce süreç bütünlüğü yönetimi hakkında bir sonraki kısımda bilgi verilmektedir.

2.3.1. Süreç Bütünlüğü Yönetimi

Birçok alt sistemlerden oluşan karmaşık iş süreçlerinin tutarlılığını sağlamak bilişim sektöründeki en uğraştırıcı problemlerden birisidir. Bu kısımda öncelikle sorun incelenmekte, süreç bütünlüğü için genel çözümler belirtilmektedir.

Süreç bütünlüğü çok iyi tanımlanmış bir kavram değildir. Süreç bütünlüğünü oluşturan çekirdek yapılar veri bütünlüğü kavramlarının kurulumuna bağlıdır. Ancak veri bütünlüğü, karmaşık iş süreçlerinin

bütünlüğü sağlamak için ihtiyaç duyduklarını karşılamakta yetersizdir. Bu yüzden süreç bütünlüğü kavramı oluşturulmuştur .

Önceden de belirtildiği gibi, karmaşık iş süreçlerindeki problem klasik veri bütünlüğü sağlamaktan öte çözümler gerektirmektedir. Bu durumlarda merkezi bir veri saklama deposundaki verinin kısa süreli değişimleriyle değil farklı sistemler arasındaki uzun süreli işlemlerle ilgilenilmektedir. Tüm katılımcılara vaktinde erişim yapılamayacağı için bu tip işlemler iyi tanımlanmış durum bilgilerine sahip değildir. Örneğin, tedarikçiden hammadde alan imalatçı senaryosu ele alınırsa, imalatçı bir siparişi için son dakika iptali alırsa, içsel sistem bu iptali diğer sistemlere yansıtana kadar süreç tutarsızlıkları oluşabilecektir (Enterprise SOA, 2004).

Süreç bütünlüğünü sağlamak için birçok çözüm bulunmaktadır. Bu çözümler arasında günlükleme ve izleme gibi basit teknik çözümlerin yanında ileri seviyede hareket kavramları bulunmaktadır.

Süreç bütünlüğünün nasıl sağlanabileceğini anlayabilmek için öncelikle ACID özellikli hareketler incelenecektir.

"ACID" Hareketler

OLTP sistemleri paylaşılan verinin birçok kullanıcı tarafından eş zamanlı olarak işlenmesine olanak sağlamaktadır. Hareket kelimesi, OLTP sisteminde verinin durumunu değiştiren birim iş olarak tanımlanmaktadır. En üst seviyede veri bütünlüğünü taahhüt eden, eş zamanlı ve dağıtık çalışabilen hareketlerin ideal karakteristik özelliklerini belirmek için "ACID" kavramı ortaya çıkarılmıştır .

"ACID", hareketlerin karşılaması gereken şu dört ihtiyacı tanımlamak için kullanılan bir yoldur (Pehepe, 2006):

Bölünmezlik ("Atomicity"): Hepsi yada hiçbiri prensibine dayanan atomik ("atomic") iş birimleridir. Hareket sırasında bir hata oluşması durumunda tüm hareket geri alınmaktadır. Para transferi başarılı olsa bile kredi güncellenmesinde sorun olduğu taktirde transfer işlemi geri alınacaktır .

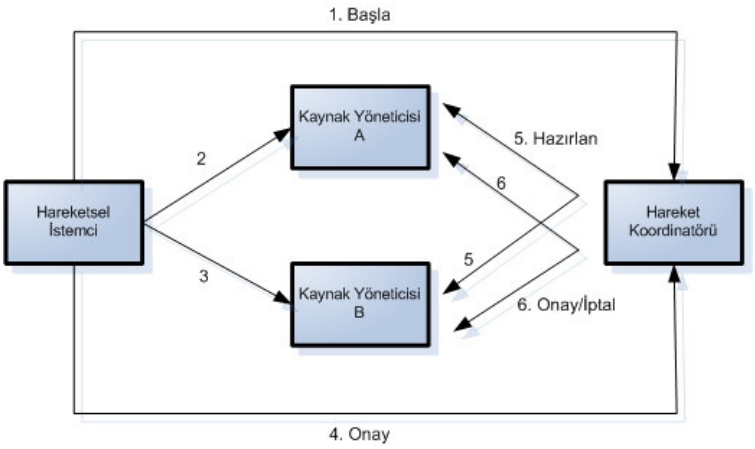
Tutarlılık ("Consistency"): Veriyi tutarlı bir durumdan başka bir tutarlı duruma çevirmelidir. Örneğin bir müşteri hesabının silinmesi durumunda müşteriye ait adres bilgileri de silinmelidir . Bir hareket, veri kaynağının tutarlı bir durumda kalmasını sağlamalıdır (Pehepe, 2006).

İzolasyon ("Isolation"): Çalışmakta olan bir hareketin içsel durumu bir diğer hareket tarafından bilinmemelidir . Tamamlanmamış hareketler veri kaynağının diğer kullanıcıları tarafından görülmemelidir. Yani tamamlanana kadar izole edilmiş olarak kalmalıdır (Pehepe, 2006).

Dayanıklılık ("Durability"): Hareketin onaylanmış güncellemeleri süreklidir. Hareketin çalışması sırasında hata oluşması durumunda sistem hareket başlamadan önceki haline geri döndürülür . Bir hareket veritabanına yazıldıktan sonra kalıcı ve dayanıklı olmalıdır.

Hareket İzleme ve Dağıtık İki Aşamalı Onay (2PC)

Farklı sistemlerin etkileştiği bir uygulama "ACID" özellikleri taşıyacak şekilde gerçekleştirilmesi zordur. Şekil 2-14' te görüldüğü gibi, birden fazla veri tabanı veya hareketsel kaynak üzerinde çalışan bir hareketin "ACID" özelliklerde olmasını sağlamak için Hareket Dinleyicileri ("Transaction Process Monitors - TPM") kullanılabilir .



Şekil 2-14 Hareket Koordinatörü

2PC (İki Aşamalı Onay) protokolü, birden fazla kaynak yönetimi ile çalışan hareketlerin "ACID" özelliklerde olması için kullanılmaktadır. Farklı kaynaklardaki hareketler, hareket izlemenin bir parçası olan hareket koordinatörü tarafından düzenlenmektedir.Her dağıtık hareketin sonunda koordinatör katılımcı kaynak yöneticilerindeki hareketin onaylanmasını ayarlamaktadır :

- İlk aşamada (hazırlanma) tüm katılımcı kaynak yöneticileri, onay veya ret cevabını göndermek için hazırlanırlar.
- İlk aşamanın sonuçlarına göre koordinatör katılımcıları onayların yada işlemi geri alın diye bilgilendirir.
- Tek bir iptal oyu bile tüm hareketin geri alınmasına yeterlidir. Tüm katılımcıların onay vermesi durumunda tüm değişiklikler kalıcı hale getirilmektedir.

"ACID" hareketler tek bir veritabanı yada dağıtık sistemler için bütünlüğü sağlamak açısından iyi bir teorik kavram olmasına rağmen, gerçek uygulamalarda kullanılması pratik olmayan yapılardır. Performans konusunda yaşanan sıkıntılar, uzun süreli hareketlere destek vermemesi (kaynakların uzun süre kilitli tutulmasından dolayı), varolan uygulamalarla ve paket programlarla birlikte çalışılabilirlik sorunlarından dolayı kullanım zorlukları bulunmaktadır.

Hareketsel Adımlar

Hareketsel adım, tek bir hareket kapsamında işletilen birbirleri ile alakalı aktiviteler kümesidir. Hareketin başlangıcında, bir mesaj girdi kuyruğundan okunmaktadır. İlgili aktivitelerin sonucu çıktı kuyruğuna yazılmaktadır. Hareketsel adımlar, karmaşık ve uzun süreli iş süreçlerinin, hareketsel bütünlük sağlayan ve kısa süren özel adımlara ayrılmasına yardım ettiğinden, süreç bütünlüğü için anahtar kavramdır. Ayrıca dağıtık sistemlerin dayanıklılığını ve esnekliğini arttırmaktadır .

Hareket Zincirleri ve Telafi

Hareketsel adımlar, özel süreç adımlarının bütünlüğünü sağlamak için etkin bir yol sunmaktadırlar. Özel adımların birbirine bağlanması ile karmaşık iş akışları ve süreçler yaratılabilmektedir. Tüm sürecin ve iş akışının bütünlüğünün sağlanması gerekmektedir. Başarısızlıkla bitirme sayısı kısıtlamaları ve hata kuyrukları, özel bir adımın hata oluşturması durumunda bilgi kaybını önlemekte iyi bir yöntem olmasına rağmen, bu

hataların tespit edildikten sonra düzeltme işlemlerinin de yapılması gerekmektedir .

Olası bir çözüm, hata oluşması durumunda telafi hareketinin işletilmesi, mantıksal olarak bir önceki hareketin geri alınmasıdır. Örneğin hesaptan para düşmenin telafi hareketi hesaba düşen para kadar ilave yapmaktır .

Dikkat edilirse hareket zincirleri, "ACID" özellikli hareketlerin yalıtım özelliği konusunda daha esnek bir davranış sergilemektedir. Zincirin her bağlantısının sonucu dış dünya tarafından erişilebilir durumdadır. Örneğin, bir hareket zincirinin ilk adımında A hesabına para aktarıldığını varsayalım. Diğer adımda B hesabından bu miktar düşülmek istendiğinde, B hesabının varolmadığı fark edilmekte. Bu durumda A hesabından paranın düşmesi için bir telafi hareketi çalıştırılmalıdır. Ancak para eklenmesi ve düşülmesi işlemleri arasında bir zaman aralığı bulunmaktadır. Para eklemesi yapıldıktan sonra, başka bir hareket tarafından para çekilebilecektir. Bu durumda telafi hareketi de hataya düşecektir .

Bir iş akışında telafi hareketlerinin uygulanması için, iş akışının adımlarının girdileri veya çıktılarının günlüklenmesi gerekmektedir. Bu veriler telafi hareketleri için girdi oluşturabilecektir .

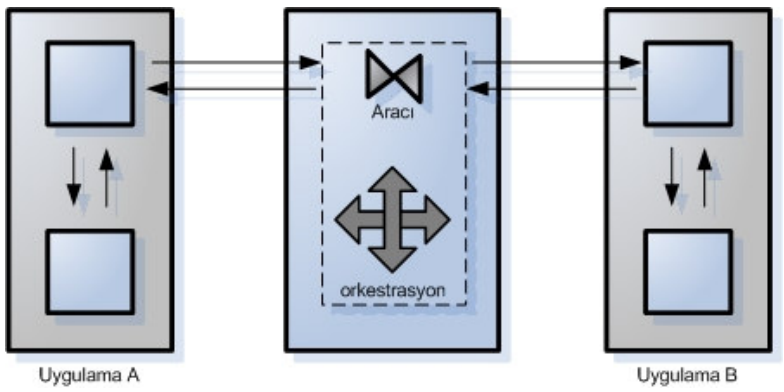
Özetle, "ACID" hareketler karmaşık iş akışları için yetersizdir. Bu tip hareketler yerine, telafi hareketleri içeren hareket zincirleri süreç bütünlüğü sağlanması için daha iyi bir yöntem sunmaktadır. Hareket zincirleri, özel hareket adımlarını birleştirip karmaşık iş akışlarına koymaktadır. Telafi hareketleri, zincirde yer alan bir hareketin çalışması

sırasında bir problem oluşması durumunda, önceden çalıştırılmış olan adımların yaptıklarını geri almaktadır .

2.3.2. Kurumsal Entegrasyon Mimarisi Temel Bileşenleri

EAI çözümleri, iki çekirdek bileşen yardımıyla uygulamalar arasındaki iletişimi merkezileştirmekte ve yeni süreçleri otomatikleştirmektedir. Bu bileşenler, aracı ve orkestrasyon bileşenleri olarak bilinmektedir (Prentice Hall, 2004).

Bileşenlerin fonksiyonlarını anlatmak amacıyla, basit veri değişim senaryoları bu kısımda örnek olarak verilecektir.



Şekil 2-15 Orkestrasyon ve aracı bileşenleri

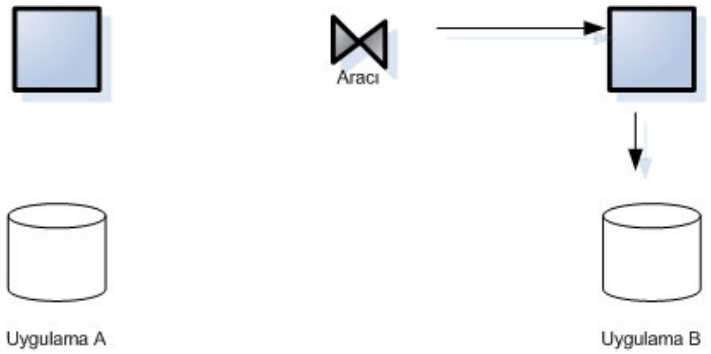
Aracı

Aracı bileşeni veri dönüşümleri yapmakta, farklı kaynaklardan gelen dokümanları düzgün bir şekilde birleştirmekte ve bir uygulamadan gelen veriyi, diğer bir uygulamanın anlayacağı dile çevirmektedir.

Temel görevi ise, bir kaynaktan alınan veriyi hedefin beklediği biçime dönüştürmektir.

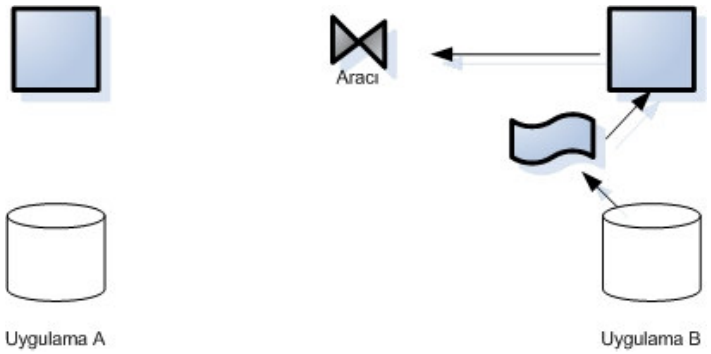
Aracı Bileşen ile örnek veri değişim örneği:

Adım 1: Uygulama A, B' ye bir istek gönderir. B, bu isteği alır ve kendi veritabanından ilgili veriyi çeker.



Şekil 2-16 Adım 1

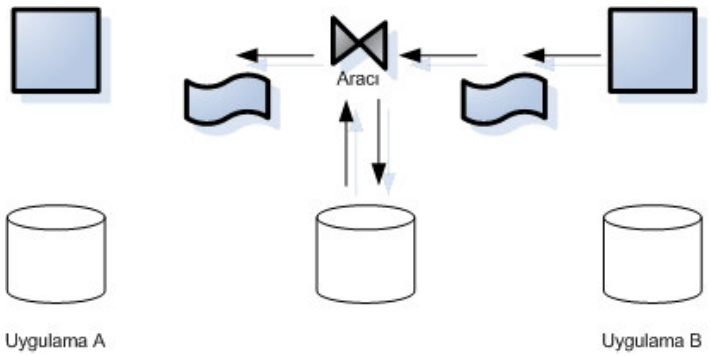
Adım 2: Uygulama dışına çıkacak olan verinin doğruluğu kontrol edilir.



Şekil 2-17 Adım 2

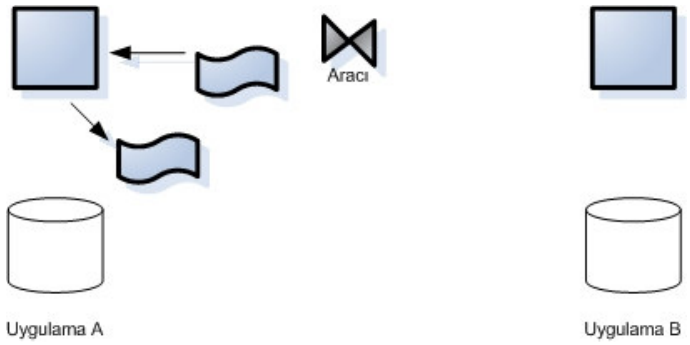
Bu adım genellikle uygulamanın kendi kodları içerisinde veya veritabanında gerçekleştirilmektedir.

Adım 3: Bu adımda aracı bileşeni devreye girmektedir. Her iki uygulamadan şemalara sahip olan aracı, nasıl eşleyeceğini bildiğinden, veriyi uygulama A' nın isteyeceği hale dönüştürür.



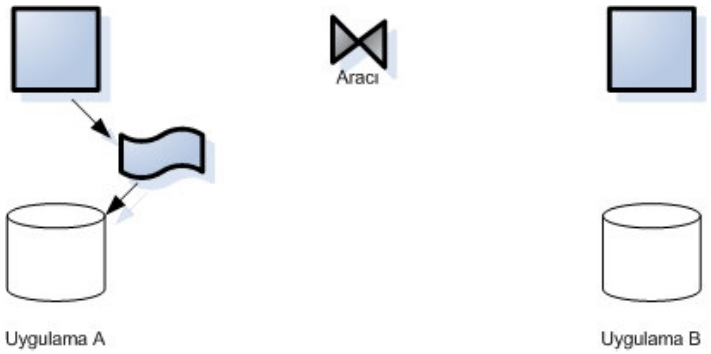
Şekil 2-18 Adım 3

Adım 4: A veriyi alır ve kontrol eder.



Şekil 2-19 Adım 4

Adım 5: Kontrol işlemi başarıyla sonlandırıldıktan sonra A, veriyi veritabanına yazar.



Şekil 2-20 Adım 5

Orkestrasyon

Orkestrasyon motoru, süreç iş mantığını saklamak ve çalıştırmak için kullanılmaktadır. Bir süreç, farklı uygulamaları ve veri kaynaklarını kullanmayı gerektiren karmaşık kurallar ve iş akışları ile istisna ve hareket yönetimi özelliklerini içermektedir.

Otomasyonu sağlanmış iş süreçlerinin mantıklarının direk olarak bir programlama dili ile yazılmış bir uygulama ile gerçekleştirilme olanağı mevcuttur. Fakat geleneksel programlama dilleri kullanılarak karmaşık yapıya sahip iş süreçlerinin yaratılması,bakımının yapılması ve yönetilmesi oldukça zahmetli bir iştir. EAI araçları bu yaklaşımı tercih

etmez. Bunun yerine iş süreçlerinin grafiksel olarak yaratılması olanağını sağlarlar.

Bunu yapmak süreci direk olarak bir programlama dilinde oluşturmaktan daha hızlıdır ve aynı zamanda bu yaklaşım sürecin daha kolay anlaşılmasına, açıklanmasına ve değiştirilmesine önemli ölçüde yardımcı olur. Bu tarz oluşturulan iş süreçleri daha kolaylıkla yönetilme olanağını sunar (Understanding BizTalk Server 2006).

Bu iş süreci tanımlandıktan sonra orkestrasyon olarak nitelendirilir. Orkestrasyon yapıları XML dokümanları ile çalışırlar. Her biri bazı XML şemalara uyan yapılara sahiptir. Dolayısıyla bu şemaları tanımlamak için bir yol olması gereklidir. EAI araçları doküman bilgilerinin tiplerinin ve yapılarının tanımları olarak düşünülebilecek şemaları yaratmaya olanak sağlar. Bu aşama da XML Şema Tanımlama Dili ("XML Schema Definition Language", XSD) kullanılır. Bir araç desteği olmadan yeni XSD şemaları yaratmak basit bir iş değildir (Understanding BizTalk Server 2006).

Orkestrasyon;

- Ek veri alabilmek için diğer uygulamalar ile entegre olabilmektedir.
- Verinin işlenmesi için aracı bileşeni tetikleyebilmektedir.
- Elde edilen verinin kontrolü ve işletilmesi için var olan uygulama mantığını kullanmaktadır.
- Verinin doğru olmadığı anlaşılırsa ret etmektedir.

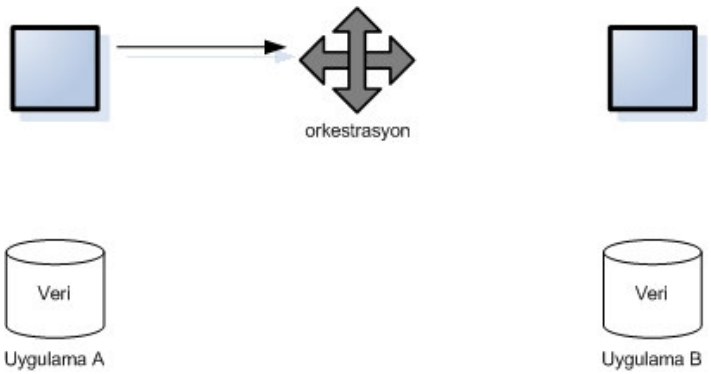
Basit entegrasyon senaryolarında kullanılabilecek olan SOAP mesajlarına kıyasla orkestrasyon tarafından yönetilen mesajlaşma sistemi

daha gelişmiş ve karmaşıktır. EAI bileşenleri tarafından işlenen mesajlar, yönlendirme ve hareket bilgilerini, standart bilgilerinin yanında taşıyabilmektedir.

Orkestrasyon kullanımı ile, entegrasyon çözümü sanal bir uygulama halini almaktadır.

Orkestrasyon ile veri değişimi örneği;

Adım 1: A uygulaması orkestrasyon motoruna gönderilmek üzere bir veri istek mesajı yaratır. Burada dikkat edilmesi gereken önemli nokta, A uygulaması B' nin varlığı ile ilgilenmemektedir. A uygulaması isteklerini süreci simgeleyen orkestrasyon bileşenine iletmekte ve sonuçları yine bu bileşenden almaktadır.

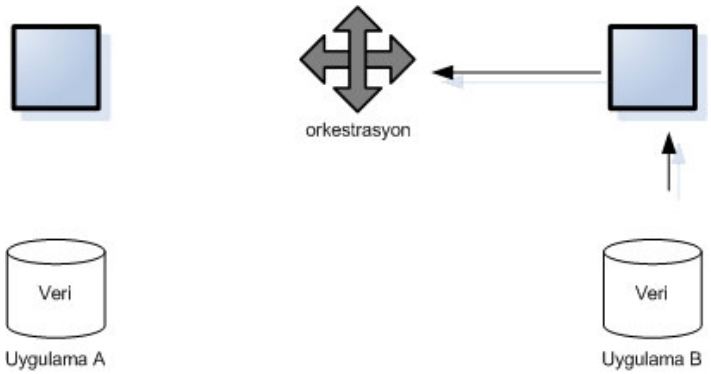


Şekil 2-21 Adım 1

Adım 2: Orkestrasyon motoru isteği B uygulamasına yönlendirmektedir. Arada başka bir işlem bu örnek için gerekmemektedir.

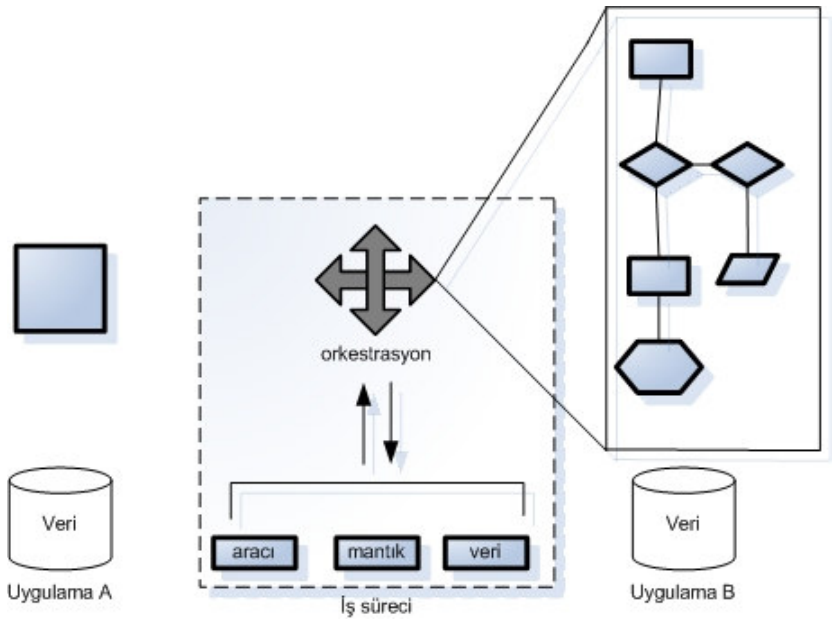
Şekil 2-22 Adım 2

Adım 3: B uygulaması çektiği veriyi orkestrasyon motoruna iletmektedir. Bu senaryona B veriyi kimin istediğini bilmemektedir.



Şekil 2-23 Adım 3

Adım 4: Bu adımda süreç mantığına bağlı olarak, orkestrasyon motoru farklı işlemler yapabilmektedir.. Örnekte, orkestrasyon motoru verinin doğruluğunu kontrol etmektedir.



Şekil 2-24 Adım 4

Adım 5: Tüm orkestrasyon işlemleri tamamlandıktan sonra veri A uygulamasına iletilmektedir.

Şekil 2-25 Adım 5

2.3.3. Kurumsal Entegrasyon Mimarileri

Sayısız EAI çözümü üreticisi, iş süreçlerini birbirlerine uygun hale getirmek için ileri düzeyde ortamlar sağlamışlardır. Bu ortamların en önemli iki bileşeni, varolan süreçlerin değişmesine ve yeni iş süreçleri eklenmesine olanak sağladığı için daha önceden de anlatıldığı gibi aracı ve orkestrasyon bileşenleridir.

Birçok EAI çözümü asenkron mesajlaşma iletişim çatısına dayanmaktadır. XML, yapısı gereği bu çatıya uyum göstermektedir ve kendisini çatının standart veri temsil teknolojisi olarak kabul ettirmiştir (Prentice Hall, 2004).

"Hub and Spoke"

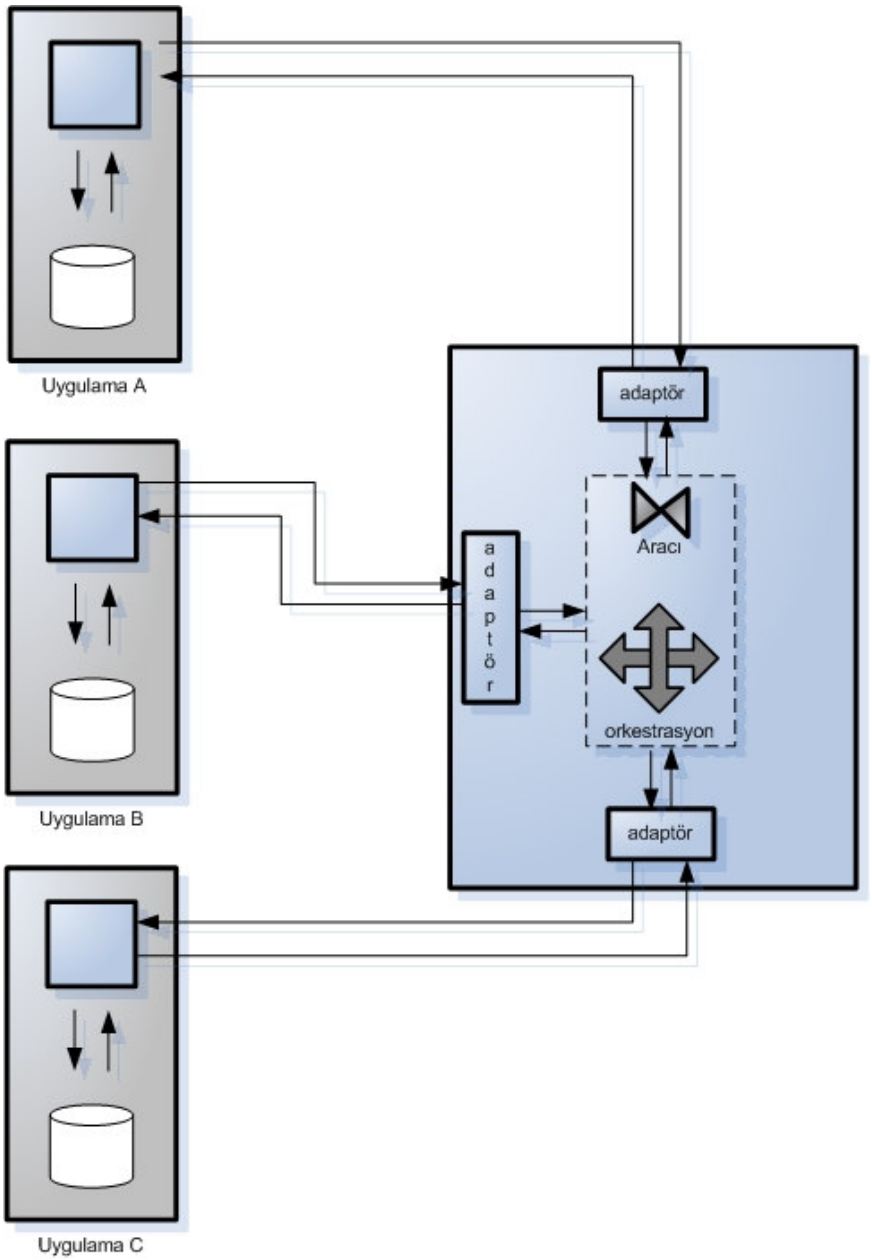
En popüler EAI modelidir. Bir üretici tarafından sağlanan "hub" aracılığı ile merkezi işleme bu mimaride sunulmaktadır. Merkezi bir sunucu, orkestrasyon ve aracılık işlemlerini kontrol eden entegrasyon mantığını barındırmaktadır (Prentice Hall, 2004).

Geleneksel Mimari:

Adaptörler yardımı ile, farklı tipteki istemci uygulamalar, “hub” ta bulunan orkestrasyon iş akışı ile veri alışverişinde bulunmaktadır. “Hub” a bağlı olan bir uygulamanın, dolaylı yoldan etkileşimde olduğu diğer uygulamaları bilmesi gerekmemektedir. Şekil 2-26’ da uygulamaların “hub” a nasıl bağlandığı gösterilmektedir (Prentice Hall, 2004).

Tüm veri akışı tek bir merkezden geçtiği için gereksiz veri girişi ve işlem sayısı azalmakta ve iletişimin izlenmesi daha kolay sağlanmaktadır. Bu tasarım süreç bakımını tek bir noktaya taşımakta ve yeniden kullanımı yaygınlaştırmaktadır (Prentice Hall, 2004).

Modelin taşıdığı bazı riskler de bulunmaktadır. En yoğun ve en popüler kısım olan “hub” ortamında yaşanacak olan bir darboğaz tüm uygulamaları etkileyecektir. Bu modelin reklamının çok yapılmasının temel sebeplerinden birisi olarak üreticiye bağlılık yaratması gösterilebilir. Alınması, kurulması ve değiştirilmesi pahalı bir çözümdür (Prentice Hall, 2004).

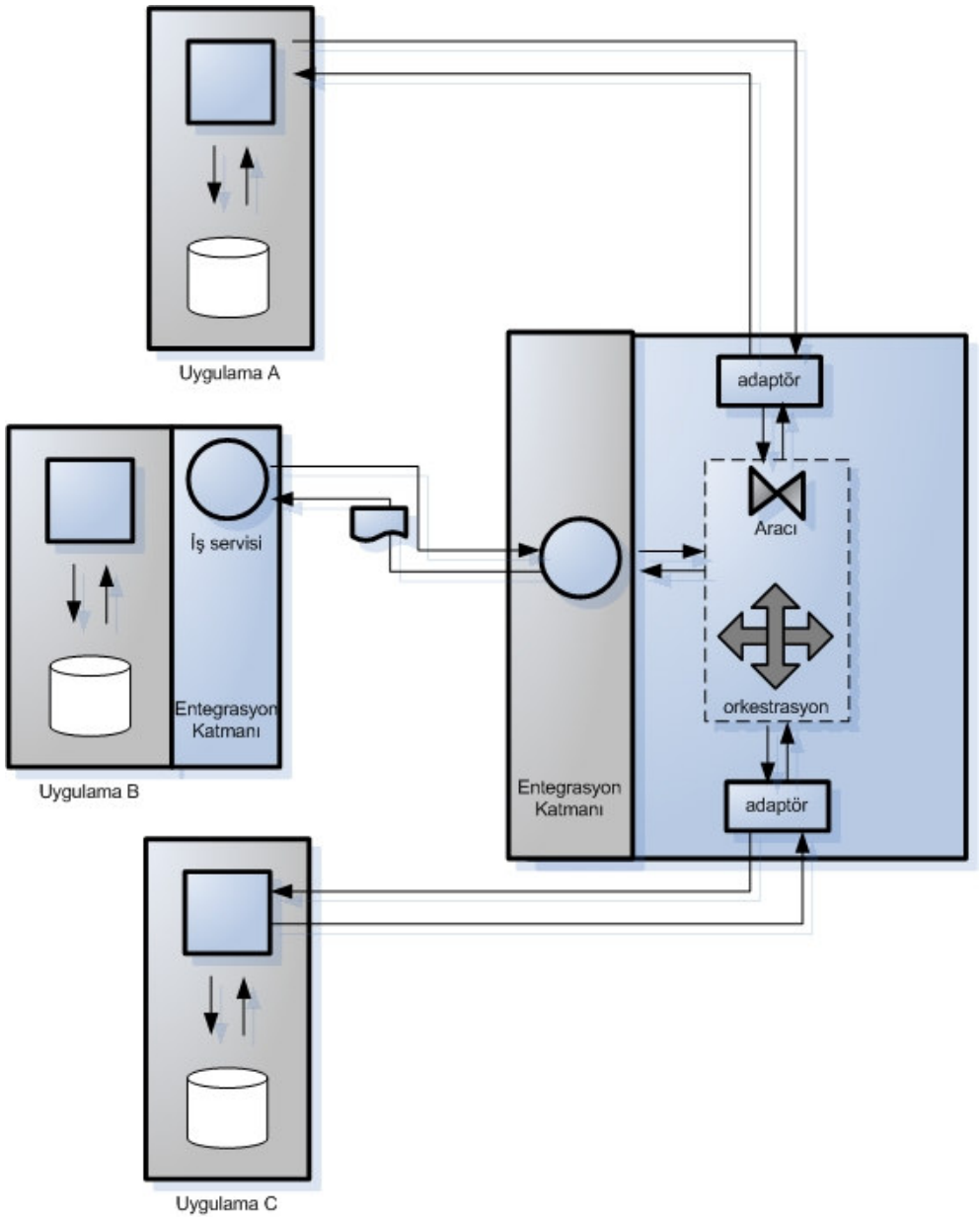


Şekil 2-26 Temel “Hub and Spoke” Mimarisi

Web Servisleri ile Entegrasyon Katmanı Eklemek:

Şekil 2-27' de görüldüğü gibi web servis entegrasyon katmanları bu ortama eklenmiştir. Böylelikle servis tabanlı uygulamaların, adaptörler kullanmadan daha fazla yer almasına olanak sağlanmaktadır (Prentice Hall, 2004).

Servis entegrasyon katmanlarının bu mimariye dahil edilmesi "hub-spoke" sisteminin bazı tipik sorunları azaltmaktadır. Servis arayüzleri, entegrasyon ortamı dışındaki kaynaklardan ek işleme olanak sağlamaktadır (Prentice Hall, 2004).



Şekil 2-27 Servis uyumlu “hub and spoke” mimarisi

Mesajlaşma Yolu

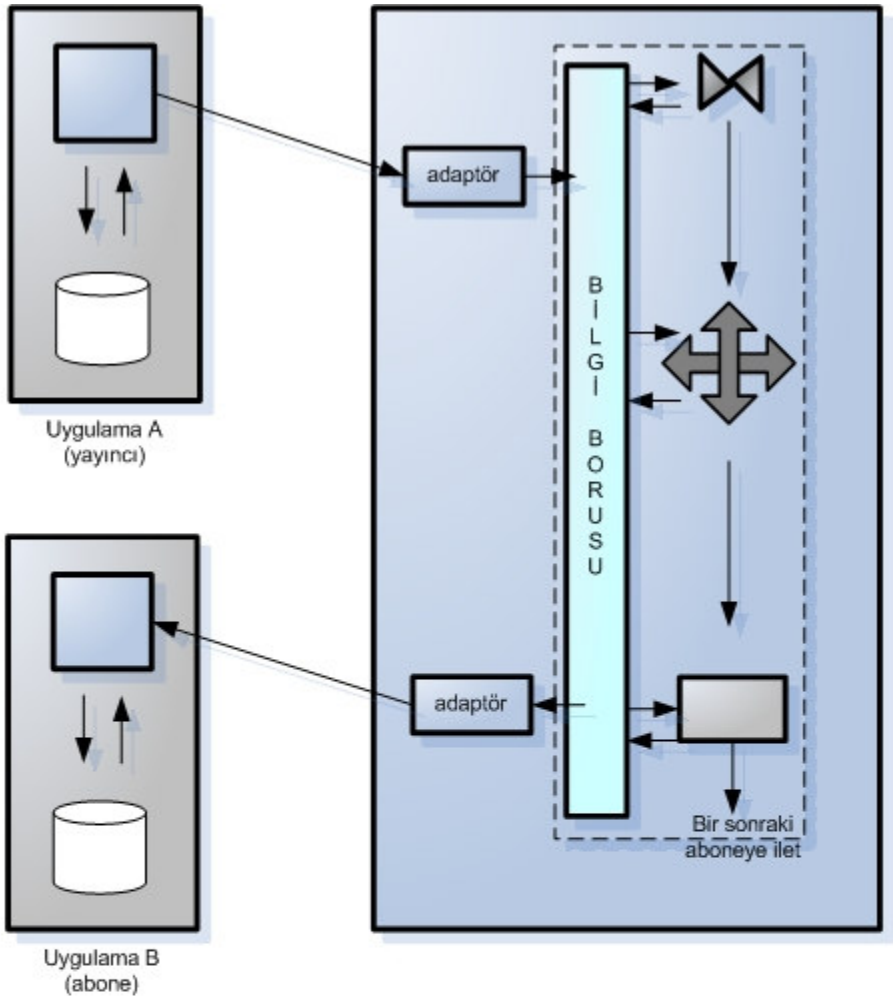
Yayımcı ve Abone modeli olarak ta bilinmektedir. Genel mimari "hub-spoke" mimarisine benzemektedir, istemci uygulamaların bağlandığı merkezi entegrasyon ortamından oluşmaktadır. Veri işleme yapısı farklılık göstermektedir (Prentice Hall, 2004).

Geleneksel Mimari:

Şekil 2-28' de görüldüğü gibi, bu mimari gelen giden mesajları taşıyan bilgi borusunu ortaya çıkartmıştır. Uygulamalar yayımcı veya abone rollerindedirler (Prentice Hall, 2004).

Her olası entegrasyon kaynağı, diğer uygulamaların abone olabileceği bir yayımcı olarak düşünülmektedir (Prentice Hall, 2004).

Yayımcı bilgiyi gönderdiği zaman yol, veriyi her aboneye iletmektedir. "Hub-spoke" ta olduğu gibi mesajlar hedefe ulaşmadan bir takım işlemden geçebilmektedir. Veri, süreç akışı sırasında, dönüştürülebilir, yönlendirilebilir, şartlı mantıklara tabi tutulabilir. İşleme, "hub-spoke" mimarisine göre daha az merkezidir. Böylelikle performans sorunu riskleri biraz azalmaktadır. Bu yapının yönetimi oldukça zordur. Bakımı ve hata durumlarının kontrolü uğraştırıcıdır (Prentice Hall, 2004).

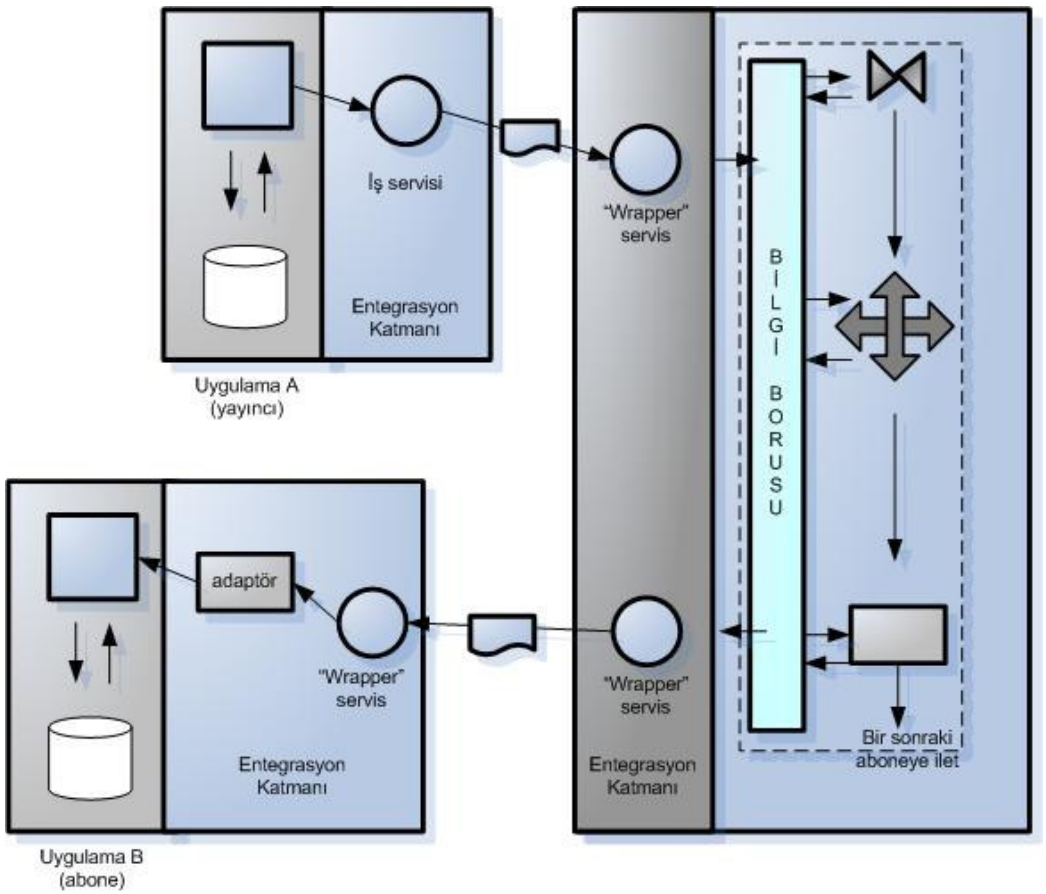


Şekil 2-28 Mesaj Yolu Mimarisi

Web Servislerini Kullanan Mesajlaşma Yolu Çözümleri:

Şekil 2-29' da görüldüğü gibi, bu yapıda servis entegrasyon katmanları, daha standart iletişim çatısı oluşturmaktadır. Web servisleri

adaptör teknolojisini soyutlamaya eğilimli olduğundan, mesajlaşma yolu sunucusunun işleme yükünü azaltabilirler (Prentice Hall, 2004).



Şekil 2-29 Servis entegrasyon katmanlı mesaj yolu mimarisi

Kurumsal Servis Yolu

Tam anlamıyla servis odaklı mimari kurulması durumunda varolan entegrasyon yöntemlerinin yarattığı problemlerin yaşanmayacağı bir çözüm oluşturulabilmektedir. Yapılması gereken, web servislerinden oluşan bir mimari için zaman ve para harcamaktır (Prentice Hall, 2004).

Kurumsal servis yolu yaklaşımında, geleneksel olan orkestrasyon ve aracı bileşenler dahil tüm bileşenler web servisleridir. Bu yapı sayesinde beğenilmeyen ürünlerin yerine yenilerinin alınması yada ek servislerin sisteme dahil edilmesi oldukça kolaylaşmaktadır (Prentice Hall, 2004).

Kurumsal entegrasyon mimarisinin temel kavramları bu kısımda anlatılmıştır. Dikkat edileceği üzere bağlılıkların azaltılması ve daha esnek yapıların oluşturulabilmesi için bu mimarilerde web servisleri kullanımı oldukça önemlidir.

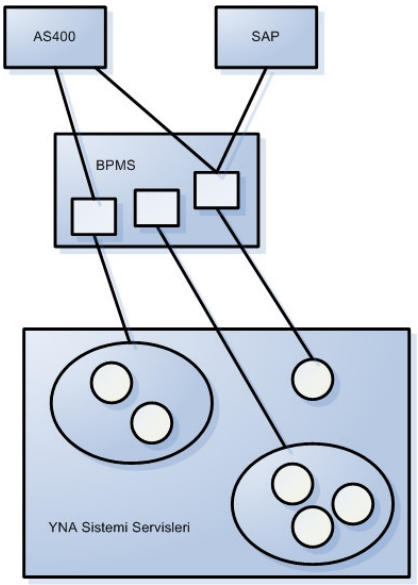
2.4. YNA Sistemi ile Varolan Sistemlerin Entegrasyonu

Daha önceden de belirtildiği gibi YNA sisteminin varolan sistemler ile entegre çalışması temel gereksinimlerden birisidir.

Kurumun büyüklüğü, iş hacmi ve süreçlerinin karmaşıklığına göre birbirleri ile etkileşim içerisinde olacak olan uygulamaların sayısının arttığı düşünülecek olursa bu tip entegrasyon katmanlarının farklı katmanlar olarak geliştirilmesinin önemi fark edilmektedir.

2.4.1. Mevcut Durum

YNA, SAP ve AS400 sistemleri arasındaki entegrasyon katmanı mesajlaşma yolu mimarisine uygun hazır bir yazılım (Microsoft BizTalk) ile oluşturulmuştur. Tanımlanmış olan entegrasyon iş süreçleri, SAP ve AS400 sistemleri ile adaptörler aracılığı ile, YNA sistemi ile web servisleri aracılığı ile konuşmaktadır.



Şekil 2-30 Yna Sistemi Entegrasyon Katmanı

2.4.2. BizTalk Nedir?

Organizasyonlar servise yönelik dünyaya yöneldiğinden beri asıl hedef sadece yazılımları birbirine bağlamak değil, ayrı sistemleri bir araya getiren etkili iş süreçlerini yaratmaktır.

BizTalk Sunucu bu hedefi desteklemektedir. Ayrı yazılımları bağlamaya izin verir ve yazılımı kullanan süreç mantığının grafik olarak yaratılmasına ve değiştirilmesine olanak sağlar. Ayrıca bu ürün bilgi çalışanlarını çalışan süreçleri izlemelerine, alışveriş ortaklarıyla iletişim içinde olmasına ve iş bazlı işlem yapılmasına olanak sağlar.

BizTalk Sunucu motorunun 2 ana bölümü vardır:

- "Mesajlaşma" kısmı birçok farklı yazılım ile konuşma yeteneğini sağlar. Farklı cinste iletişimler için kullanılan

takılan adaptörlere bağlı olarak,motor çeşitli protokol ve veri formatlarını destekler.

- "Yönetim" olarak adlandırılan grafik olarak tanımlanmış süreçlerin yaratılmasına ve çalışmasına destek sağlar. Motorun mesajlaşma bölümünün üzerinde yaratılmış olan, yönetim iş süreçlerinin tümünü veya bir kısmını süren mantığı oluştururlar.

BizTalk Sunucu motoru çeşitli yazılımlarla konuşmak zorunda olduğundan, bunu gerçekleştirebilmek için adaptörlere ihtiyaç duymaktadır. Adaptör, iletişim mekanizmasının gerçekleştirimidir. Geliştirici belirli bir durumda hangi adaptörü kullanacağını belirleyebilmelidir. BizTalk Sunucu' sunun sunduğu adaptörlerden birini seçebilmekte, örneğin popüler bir ürün olan SAP için yaratılmış olan adaptör veya bunun kişiselleştirilmişini kullanabilmektedir.

2.4.3. BizTalk Kullanımının Getirdiği Sorunlar

BizTalk Sunucu, sadece Microsoft tabanlı ürünler de başarılı bir performans ortaya koyarken işin içine Linux veya farklı bir platform girdiğinde performansı düşmektedir. Her ne kadar BizTalk, farklı sistemlerdeki programlarda çalışabilme yeteneğini sağlasa da bunun gerçekleşmesi için ilgili program adaptörünün geliştirilmiş olması gerekmektedir.

Adaptör alımları ve bu adaptörlerin kullanımı için gerekli olan öğrenme süresi, kurumlar için önemli bir maddi yük olmaktadır. Ayrıca her türlü teknik eğitim ve destek konusunda üreticiye bağlı kalınmaktadır.

Son zamanlarda bu araçta ve benzerlerinde web servis desteği olmasına rağmen, yinede standart bir entegrasyon ortamı kurulumu mümkün değildir.

2.5. Standartlara Dayalı Entegrasyon Çözümü

Uygulamalar arası bilgi entegrasyonu, "Uygulamadan Uygulamaya Entegrasyon" yada "Kurumsal Uygulama Entegrasyonu" olarak bilinmektedir. Kurumsal Uygulama Entegrasyonu' nda;

Birinin tescilli malı olan

1. Mesaj akış dilleri
2. Veri dönüşüm tanımlamaları
3. Her uygulama için ayrı bir adaptör

kullanılmaktadır.

Bu tip bir yapı yerine;

Standart olan

1. Mesaj akış dilleri
2. Veri dönüşüm araçları
3. Adaptörler

kullanımı önemli avantajlar sağlayacaktır.

Herhangi bir üreticiye bağlı kalmadan standartlara dayanan bir entegrasyon çözümü için mesaj akış dili olarak BPEL4WS, veri dönüşüm aracı olarak XSLT ve standart adaptör olarak Web servisleri kullanımı iyi bir alternatif olabilir. Gevşek bağlı BPEL süreci sayesinde üretici bağımlılığı yok edilebilir, entegrasyon masrafları azaltılabilir ve birlikte çalışabilirlik sağlanabilir.

BPEL4WS tanımı yapılmadan önce XML' in standart entegrasyon çözümleri için önemi ve web servisleri ile web servislerinin katılımcısı

olduğu iş süreçlerinin yönetimi ile ilgili bilgi sahibi olmak gerekmektedir.

2.5.1. XML Dokümanlarının Farklı Sistemler Arası Entegrasyonda Kullanılması

Farklı sistemlerin entegrasyonu veya bütünleştirilmesi söz konusu olduğunda HTTP, SSL vb. protokolleri destekleyen XML teknolojisinin kullanılması vazgeçilmezdir.

XML standart bir yapı olduğundan,mevcut sistemlerle entegrasyon noktasında bir anlamda çıkış yolu rolü üstlenir. Birbirinden bağımsız ve tamamen farklı sistemler, XML sayesinde ortak bir noktada buluşabilir.

XML, verinin yapısını belirlediği ve veriyi görünümden ayırdığı için, büyük esneklikler sunmaktadır. Aynı XML verisini değişik servisler aracılığı ile değişik şekillerde işlemek olanaklıdır. Ayrıca veritabanı yerine XML yapılarını kullanmak, sisteme büyük bir esneklik getirmiştir. Gerek yeni XML yapılarının eklenmesi, gerekse mevcut yapılarda değişiklik yapılması oldukça kolaydır (Haluk Cavkaytar).

Bir iş sürecini gerçekleştiren orkestrasyon tipik olarak bazı dokümanları alır ve diğerlerini yollar. Alınan dokümanlardaki bilgilerin bir kısmının gönderilen dokümanlarda iletilmesi sıklıkla rastlanan bir durumdur. Örneğin sipariş gerçekleşme işlemi çeşitli sayıda kalem içeren bir sipariş alabilir ve daha sonra siparişin belirli bir sebepten dolayı kabul edilmediğine dair bir mesaj geri döndürebilir. Talep belirleyici ve sipariş verilecek miktar gibi siparişte bulunan bilgilerden bazıları gelen sipariş mesajından kopyalanıp ret mesajına aynen eklenmesi karşılaşılabilecek

bir durumdur. Aracı bileşenler, bir dokümandan diğerine bir dönüşüm tanımlamak için kullanılabilir. Buna eşlem veya harita adı verilir.

Her bir eşlem, iki XML şema arasında karşılıklı ilişkiyi açıklar. W3C, XML şemalar arası bu tür dönüşümleri belirtmek için bir standart yol tanımlamıştır. Bu tanımlama "Extensible Stylesheet Transformation" (XSLT) olarak isimlendirilir (Understanding BizTalk Server 2006).

2.5.2. Web Servisleri

Bir Web Servisi, uzak istemcilerin başvuruda bulunduğu çeşitli işlevsel metot çağrımlarını barındıran, çok yönlü ve merkezileştirilmiş bir ünitedir. Bir web servisi, çok sayıda istemci tarafından erişilebilen bir yapıya sahiptir. Onu diğer dağıtık nesne modellerinden farklı kılan, sahip olduğu alt yapı sistemi sayesinde, platform bağımsız uygulanabilirliği sağlamasıdır (W3Schools, 2006).

Bir web servisi, standart olarak HTML iletişim protokolü üzerinden veri alışverişine izin veren bir yapıdadır. HTML tabanlı bu sistemin bilgi otobanı XML temelleri üzerine dayandırılmıştır. XML' in sağladığı esneklik, kolay geliştirilebilirlik özelliklerinin yanı sıra, sağlamış olduğu küresel standartlar, platform bağımsız veri transferi kavramını daha da geliştirmiştir.

Bir web servisi, tek başına bir anlam ifade etmez. Web servisini kullanan istemcilerin de olması gerekmektedir. İstemciler Internet ortamında olabileceği gibi, şirketin ağ sisteminde yada evimizdeki makinenin yerel sunucusu üzerinde olabilmektedir. Bir istemci, bir web servisini kullanmak istediğinde tek yapması gereken, bu web servisi ile

konuşabilecek ortak bir takım standartları uygulamaktır. XML tabanlı bu standartlar sayesinde istemciler, web servisine ulaşabilmekte, bu servis üzerinden metotlar çağırabilmekte, bu metotlara parametreler gönderebilmekte ve metotlardan dönen değerleri, örneğin veri kümelerini elde edebilmektedir.

Web servislerini kullanacak istemciler ile arada kurulacak ilişkinin belli standartlara dayandırılması gerekmektedir. Her ne kadar, web servisleri HTML üzerinden gidecek XML veri parçalarını kullanıyor olsa da, bunların, istemcilerin işleyebileceği ve anlayabileceği bir hale getirilmeleri gerekmektedir. Bu noktada devreye Web Servisleri için önemli ve gerekli temellerden birisi olan Basit Nesne Erişim Antlaşması – SOAP girmektedir.

Bir istemci, kullanacağı web servisine ait bir takım bilgilere sahip olmak zorundadır. İstemci bu bilgileri kullanarak web servisinden SOAP protokolüne uygun olarak hazırlanan XML mesajını gönderir. Kodlanarak gönderilen bu mesaj, Web Servisi tarafından çözülür, gerekli parametreler ve metot çağrım bilgileri eşliğinde bir takım işlemler gerçekleştirir. Bu işlemler sonrasında Web Servisi, istemciye döndüreceği cevap bilgileri için yine SOAP protokolüne uygun XML mesajlarını oluşturur. Bu mesajlar HTTP üzerinden istemci uygulamaya ulaşır, burada çözülür ve değerlendirilir (W3Schools, 2006).

WSDL

İstemcilerin kullanacakları web servisindeki bilgileri önceden bilmeleri gerekir. Web Servisleri Tanımlama Dili – WSDL bu noktada devreye giren bir diğer önemli unsurdur. İstemci uygulamalar WSDL yardımıyla, kullanacakları web servisine ait bilgileri önceden tedarik

ederler. Bu istemcinin web servisi üzerindeki bir web metodunun varlığından haberdar olması, onu nasıl kullanacağını bilmesi anlamına gelmektedir.

WSDL web servislerini ve bu servislere nasıl erişileceğini tanımlayan XML tabanlı bir dildir (W3Schools, WSDL Tutorial).

Temel özellikleri şunlardır:

- XML ile yazılmaktadır
- XML dokümanıdır
- Web servislerini tanımlama için kullanılmaktadır
- Web servislerinin yerini belirleme için de kullanılmaktadır

Temel olarak bir WSDL dokümanı 5 ana kısımdan oluşmaktadır.

- types
- message
- portType
- binding
- service

"Types" kısmında, web servisi ile ilişkili SOAP mesajlarında taşınacak parametre ve geri dönüş değerlerine ait şema elemanları tanımlanır (WSDL, 2006).

"Message" kısmında, web servisinin kabul edeceği ve geri döndüreceği mesajlara ait özet bilgiler yer alır (WSDL, 2006).

"portType" kısmında ise, her bir web servisi metodu için birer operasyon tanımlaması yapılır. Bu sayede, "Proxy" nesnesi üzerindeki bir metot ile web servisi üzerinde kullanılabilir çağrılar gerçekleştirilebilecektir. Başka bir deyişle, web servisi üzerinden gerçekleştirilebilecek operasyonların tanımlamaları yapılmaktadır. Operasyon isimleri web servisindeki metot isimleri ile aynıdır. Buradaki eleman isimleri ile fiziki metotlar, bağlama kısmında eşleştirilecektir (WSDL, 2006).

"Binding" kısmında, WSDL dokümanındaki her bir operasyon için, bu operasyona web servisinde karşılık gelecek metot tanımlamaları yapılır. Bir başka deyişle her bir operasyon elemanı için fiziki olarak metot adresleri belirlenir. Bu adresler için kullanılacak operasyonlar belirli olduğu için, bu operasyonlara bağlı mesajlarda, fiziki adreslere bağlanmış olur (WSDL, 2006).

2.5.3. Web Servisleri ve İş Süreçleri

Web servisleri uygulamaların karşılıklı olarak haberleşebilmesi ve çoklu işlemlerin otomatik ve ortaklaşa çalışabilmesi için yeni olanaklara imkan vermektedir (Net.ObjectDays, 2002).

Son dönemlerde teknoloji ve iş dünyasındaki bazı gelişmeler, kurumların elektronik ticaret modeline benzer bir şekilde, farklı platformlarda ve farklı lokasyonlarda çalışan ekiplerinin çalışmaları arasında koordinasyon sağlamalarını zorunlu kılmaktadır. Bilgi ile uzmanlığın koordineli olarak paylaşılması ve ticari iş süreçlerinin ayrı ayrı organizasyonları kapsaması, dağıtık sistemler arasındaki haberleşmenin olması gereken düzeyde devam etmesi halinde

mümkündür. Buna bağlı olarak, büyük ölçekli sistemlerde işlemlerin dağıtık olarak gerçekleştirilebilmesi için mevcut sistemlere iş akışı ve iş süreç yönetim çözümleri de ilave edilmektedir. Uygulamalar senkron/asenkron haberleşme desteğine, uzun çalışabilen işlemlere, bileşik servisleri kullanma olanağına, durum, akış ve hata kontrollerine ihtiyaç duymaktadır. "Web servis orkestrasyon" konusu tüm bu olayları kapsamaktadır (Net.ObjectDays, 2002).

Çeşitli işlemlerin türdeş olmayan sistemlerde birlikte çalışabilmesi için, işlemleri direkt erişilebilen (Web) servislere dönüştürme yoluna gidilebilir. Fakat günümüzde iş senaryolarını gerçekleştiren uygulamaların Web servis özellikleri ile donatılabilmesi bazı zorlukları beraberinde getirmektedir. Temel bir Web servis belirli bir işlem modeli veya gerçek-zamanlı işlemleri desteklememektedir. Ayrıca ana standartlar karmaşık süreçler ve işlem hareketleri için yeterli altyapıyı sunmamaktadır (Net.ObjectDays, 2002).

İşletmelerde operasyonların tümü sadece veri paylaşımına dayalı değildir; karmaşık süreçler ve çeşitli adımlardan oluşan uzun süreli işlemler sıralı bir akış içerisinde çalışıyor olabilir. Özel uygulamalar ise karmaşık yapıda özel hazırlanmış yazılımlardan oluşabilir. Bu bağlamda, işlemleri meydana getiren süreçler de iş odaklıdır. Web servis mimarisi de servis odaklı bir yapı öne sürmektedir, ancak çıkış noktası teknolojik bir perspektife dayanmaktadır. Web servis arayüzlerinin süreçler arası ilişkilerin tanımlanabilmesi, uygulama sıralarının belirlenebilmesi, seri işlem desteği ve işlem mesajlarının sonuçlarının izlenebilmesi gibi fonksiyonları içeren nitelikler ile zenginleştirilmesi gerekmektedir. Süreç seviyesinde bütünleştirme için de, dinamik süreç modelleme ve yönetim seçenekleri mevcut olmalıdır (Net.ObjectDays, 2002).

Günümüzde geleneksel işlem yapıları için kurumsal uygulama entegrasyonu araçları kullanılmaktadır. Ancak bu tür uygulamalar daha önceden de belirtildiği gibi üretici özelliklerine bağlı kalmaktadır ve sadece merkezi mesajlaşma sayesinde işlem akışı için denetim sağlamaktadır. Web servislerinde ise farklı işlem gereksinimlerine ortak bir çözüm arayışına gidilmektedir. Bunun ötesinde EAI uygulamalarını hayata geçirme aşaması çoğunlukla yüksek maliyetli bir projeye dönüşmektedir. Bir işleme ait hareketleri dağıtık sistemlerde işleyebilmek için veri tutarlılığını sağlamak gerekir. Geleneksel entegrasyon çözümlerinde sıkı bağlı bilgisayar mimarileri tasarlanarak, olası veritabanı, sistem ve iletişim sorunlarının önüne geçilmeye çalışılırdı. Bu tür sıkı bağlı bir ortamı Web' de sağlamak imkansızdır. Ayrıca Web servis mesajlaşma yapıları hataların izlenmesi açısından da önem taşımaktadır (Leymann, 2001).

Dağıtık işletim konusu başlı başına bir sorundur. İşletmeye özgü entegrasyon çözümleri farklı uygulamalar ve platformlar arasında uyumsuzluğa neden olabilir. Bu durumda ortak işlemleri gerçekleştiren tarafların, uzun işlem sürelerine ve performans düşüklüğüne hazırlıklı olmaları gerekir. Buna ek olarak, olası sistemsel hataların yakalanabilmesi için tüm aktiviteleri izlemek gibi ön koşullar ortaya çıkabilir. Web servislerin gerçek zamanlı ve dağıtık özellik taşıyan işlemleri desteklemesi de zorlukları olan bir girişim olarak tasvir edilebilir. Ancak dinamik altyapı ve standart protokollere destek sunmaları nedeniyle, Web servisler ortak bir işletim yapısı oluşturmak için uygun bir seçim olabilir (Leymann, 2001).

2.5.3.1. Web Servis Orkestrasyonu

İşletmelerin Web servis teknolojilerini alternatif bir iş entegrasyon metodolojisi olarak benimseyebilmesi için, yeni çözüm arayışına girilmiştir. Web servis orkestrasyon konusu işlem aktivitelerinin ne şekilde gerçekleştirilmesi gerektiğini, işlemlerin nasıl servise dönüştürebileceğini ve genel akışa dahil olan ortakların işlemlerin hangi bölümlerinden sorumlu olduğunu belirtebilmektedir (Leymann, 2001).

Web servis orkestrasyon konusu farklı ortamlarda yer alan çeşitli servisler için senkron/asenkron haberleşme ve dağıtımlı sistemler için Web servis mimarisinde iş süreç yönetimi başlıklarını içermektedir. Dağıtımlı sistemlerin bütünleştirilebilmesi amacıyla uygulamalar için birlikte işlerlik , taşınırlık ve saydamlık anahtar öğelerdir (Leymann, 2001).

Orkestrasyon standartları bazı ana koşulları gerçeklemek durumundadır:

- Yazılım dilinden bağımsız çözümler sunulabilmesi için, standartlar XML tabanlı olmalıdır.
- Önerilen standartlara ait uygulamalar ürün ve üreticiden bağımsız olmalıdır. Bu sayede genel çözümler üretilebilir.
- Servislerin dinamik etkileşimi için statik WSDL dilinin kullanılması gerekir.

Web Servis orkestrasyonu sayesinde basit işlemleri yerine getiren servisler biraraya getirilerek, karmaşık iş süreçlerinden sorumlu Web servisler oluşturulmaktadır (Leymann, 2001).

Standartları

Web servis orkestrasyon standartları, Web servis yığın yapısını iş süreç yönetim özellikleri ile genişletmektedir. Önerilen standartlar genel olarak interaktif çalışan ortak Web servislerini birbiri ile ilişkilendirme imkanı sunar (BPEL4WS 1.0, 2003)..

OASIS grubu "Business Transaction Protocol (BTP)" standardını sunmuştur. IBM' in hazırladığı "Web Service Flow Language (WSFL)" ve Microsoft'un hazırladığı "XML Language (XLANG)" birleşerek IBM, Microsoft ve BEA tarafından desteklenen "Business Process Execution Language for Web Services (BPEL4WS)" standardına dönüşmüştür. BPEL4WS'in tamamlayıcı iki protokolü mevcuttur; mesaj koordinasyonu sağlayan "WS-Coordination" ve işlem hareketlerine yönelik hazırlanan "WS-Transaction". BPMI.org grubu "Business Process Modeling Language (BPML)" isimli bir çalışma sunmuştur. SUN ise "Web Services Choreography Interface (WSCI)" standardını desteklemektedir (BPEL4WS 1.0, 2003). Bir sonraki kısımda BPEL4WS ve tamamlayıcı standartları detaylı olarak incelenmektedir.

2.5.4. Web Servisleri İçin İş Süreci Modelleme Dili

2.5.4.1. WS – Coordination

Günümüzdeki Web servis tarifnameleri (WSDL, SOAP) web servislerinin birlikte çalışabilirliğini sağlayan protokoller tanımlamaktadır (WS-Coordination, 2005).

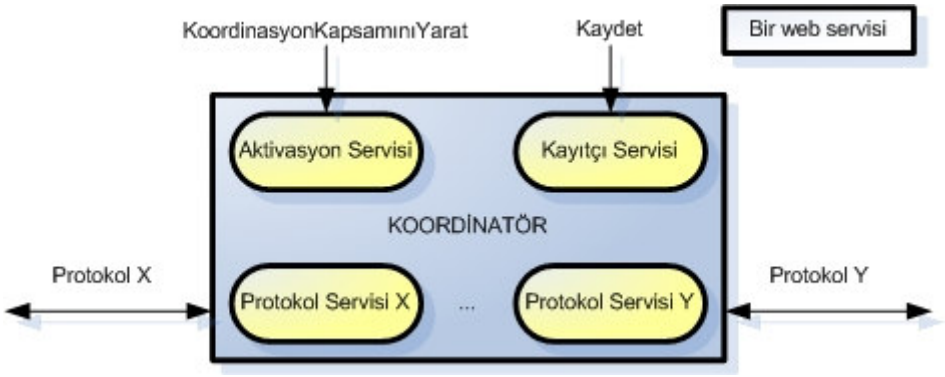
Katılımcıları arasındaki karmaşık ilişkilerden dolayı aktivitelerin yapısı da karmaşıklaşmaktadır. Bu tip aktivitelerin çalışma süresi iş kuralları ve kullanıcı etkileşimleri gibi nedenlerden dolayı genel olarak uzun sürmektedir (WS-Coordination, 2005).

WS-Coordination tarifnamesi, aktivitelerin koordine edilebilmesi için bir koordinatör ile koordinasyon protokollerini kullanan genişletilebilir uygulama çatısı tanımlamaktadır. Uygulama çatısı, dağıtık aktivitelerin sonuçlarının katılımcılar tarafından tutarlı bir şekilde alınmasına olanak sunmaktadır. Uygulama çatısında tanımlanan koordinasyon protokolleri, hem basit kıra süreli operasyonlar hem de karmaşık uzun süreli iş aktiviteleri için protokollere yer vermektedir (WS-Coordination, 2005).

Model

Bu tarifname, 3 adet servisten oluşan koordinasyon servisi (koordinatör) için varolan uygulama çatısını tanımlamaktadır (WS-Coordination, 2005).

- Uygulamanın koordinasyon nesnesi veya kapsamı yaratmasına olanak sağlayan aktivasyon servisi.
- Uygulamanın koordinasyon protokolleri için kaydolmasını sağlayan kayıtçı servisi.
- Koordinasyon tipi (özelleştirilmiş koordinasyon protokolleri kümesi)



Şekil 2-31 Koordinatör

Uygulamalar bir aktivitenin koordinasyon kapsamını ("context") yaratmak için aktivasyon servisini kullanmaktadır. Kapsam, uygulamanın takip edeceği aktiviteye özel koordinasyon davranışına kaydolabilmek için gerekli olan bilgiyi içermektedir (WS-Coordination, 2005).

Ayrıca, koordinasyon kapsamı alan bir uygulama, gerçek uygulamanın veya güvenilir farklı bir uygulamanın kayıtçı servisini kullanabilmektedir (WS-Coordination, 2005).

Koordinasyon Tipleri ve Protokolleri

Her koordinasyon servisi, aktivitenin çalışma mantığını belirleyen bir koordinasyon tipine bağlıdır. Koordinasyon tipleri farklı tarifnamelerle özelleştirilmişlerdir. Ancak ilk olarak düşünülen koordinasyon tipleri WS-AtomicTransaction ve WS-BusinessActivity dir. Bu tarifnamelere özel kavramlar WS-AtomicTransaction ve WS-BusinessActivity kısmında anlatılmaktadır (Thomas Earl, 2006).

2.5.4.2. WS-AtomicTransaction

WS-AtomicTransaction tipi "ACID" hareketler gibi çalışmaktadır. Aktivitenin tümü başarılı olduğu zaman onaylayan, aksi durumda her şeyi ilk durumuna döndüren mekanizmayı gerçekleştirmektedir (Prentice Hall, 2004).

Atomik Hareket Süreci

Atomik harekete başlayabilmek için istemci uygulama WS-Transaction destekleyen koordinatörün yerini belirlemelidir. Belirleme işleminden hemen sonra istemci "CreateCoordinationContext" mesajını aktivasyon servisine göndermekte ve uygun WS-Transaction kapsamını geri almaktadır. Hareket kapsamı kendisine ait "CoordinationType" elemanı bulundurmaktadır. Bu elemanın değeri WS-Transaction "Atomic Transaction" olarak atanmıştır ve koordinatör kayıtçı servisine referans içermektedir. Kayıtçı servisi tüm katılımcıları bilen servistir.

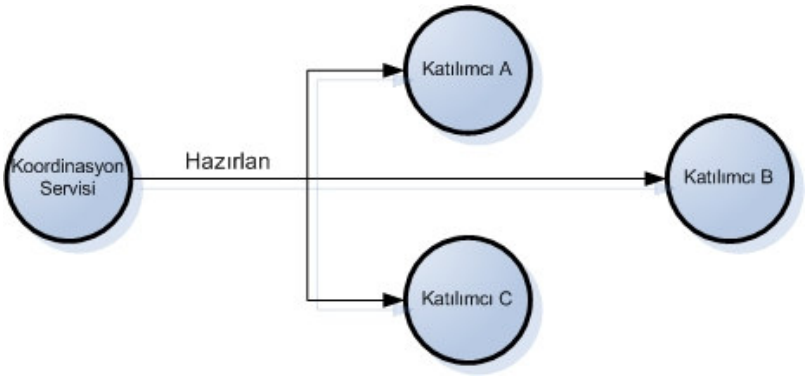
Bu işlemlerin ardından istemci uygulama iş sürecini gerektiren operasyonlar için web servisleri ile etkileşime geçmektedir. Servislere yapılan her bir istekte istemci kapsamı iletmekte böylelikle her istek hareket kapsamına alınmaktadır.

Uygulama seviyesindeki bütün işler tamamlandıktan sonra istemci hareketi bitirebilmektedir (Little-Freund, 2003).

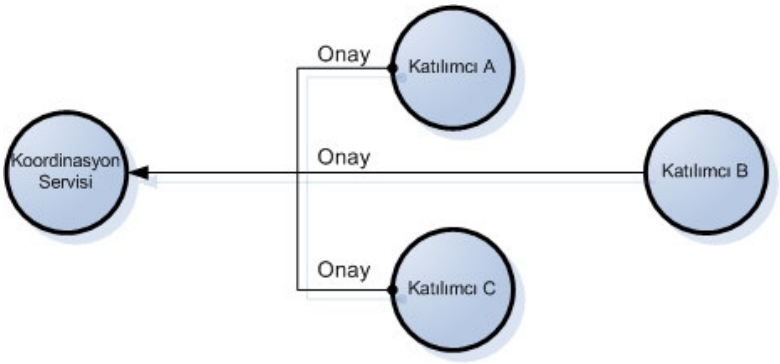
Daha önceden de belirtildiği gibi atomik hareket koordinatörü işlem sonuçlarına göre karar verme sorumluluğuna sahiptir. Bu kararı verirken tüm katılımcılardan gelen bilgileri kullanmaktadır. Uygulama sona erdiği anda koordinatörden işlem için sonuçları değerlendirmesi

istenmektedir. Koordinatör tüm katılımcılara bir hata olup olmadığını sorar, eğer tüm katılımcılardan işlem başarılı yanıtını alırsa yapılan tüm işlemler onaylanır, ancak bir katılımcıdan bile işlem iptal isteği gelirse veya katılımcıdan hiç yanıt alamazsa o ana kadar yapılan tüm işlemler iptal edilir (WS-AtomicTransaction, 2005).

Şekil 2-32 ve Şekil 2-33' te görüldüğü gibi, katılımcılardan geri bilgi alınması iki aşamaya ayrılmıştır. Hazırlanma aşamasında tüm katılımcılar koordinatör tarafından oylarını hazırlamaları ve göndermeleri için uyarılmaktadır. Her bir katılımcı onay veya iptal isteklerini göndermektedirler (Thomas Earl, 2006).

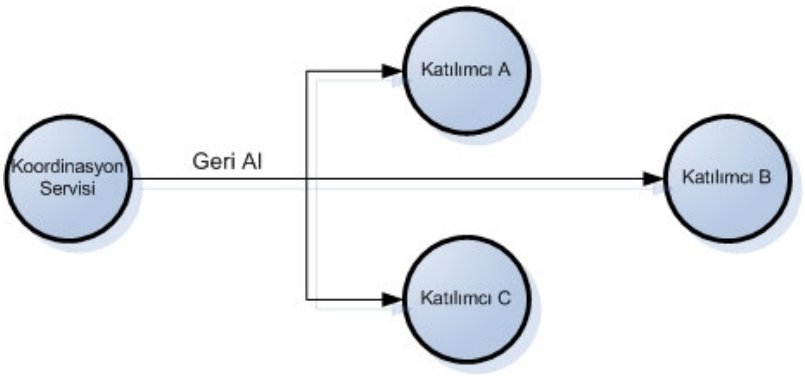


Şekil 2-32 Koordinatör katılımcı oylarını hazırlamalarını istemekte



Şekil 2-33 Katılımcılar onay veya iptal isteklerini koordinatöre göndermekte

Tüm oylar toplandıktan sonra koordinatör değerlendirmeye geçmektedir. Tüm oylar onay ise işlemin başarılı olduğu koordinatör tarafından ilan edilmektedir. Ancak bir tane bile iptal oy olması durumunda veya bir katılımcıdan bile oy gelmemesi durumunda işlemin başarısız olduğu tüm değişikliklerin geri alınması gerektiği ilan edilmektedir (Thomas Earl, 2006).



Şekil 2-34 Koordinatör işlemi iptal etmekte ve tüm katılımcılara değişikleri geri alın emrini vermekte

2.5.4.3. WS-BusinessActivity ile Uzun Süreli Hareketler

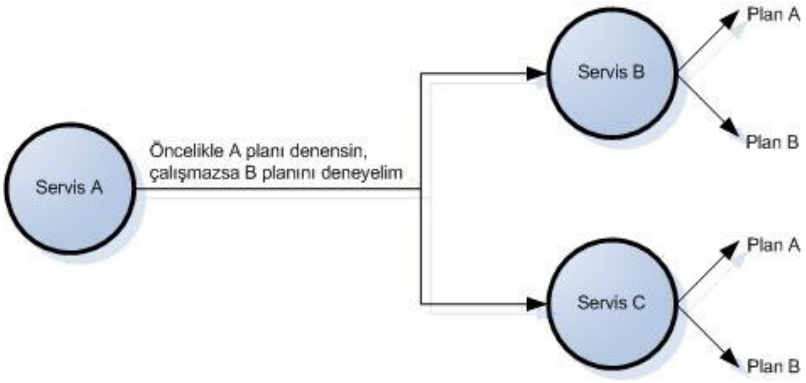
İş hareketleri genellikle birkaç atomik hareket içermektedirler. Bir atomik hareket tarafından güncellenmiş olan kaynaklar tüm iş hareketinin başarısız olması durumunda ilk hallerine geri dönmelidirler. Telafi bu sorunun çözümü için yaygın kullanılan çözüm metodolojilerinden birisidir. (Freund, 2002).

Atomik hareket çalışması süresince aktivitenin kullandığı kaynaklar işlem bitene kadar genellikle kilitlenmektedir. Bu tip işlemlerin yaşam süresi kısa olduğundan dolayı sistem kaynaklarının kullanımı açısından bir sorun çıkmamaktadır (Prentice Hall, 2004).

İş aktiviteleri uzun süreli ve çoklu servis içeren karmaşık işlemlerden oluşmaktadır. Bir aktivitenin tamamlanması saatler, günler hatta haftalar sürebilmektedir. Bu zaman dilimi içerisinde aktivite birçok katılımcının yer aldığı sayısız iş gerçekleştirebilmektedir. Bu tip farklılıklardan dolayı iş aktiviteleri için WS-BusinessActivity protokolü geliştirilmiştir (Thomas Earl, 2006).

İş aktivitesini karmaşık bir aktiviteden ayırt eden, katılımcılarının özel protokol kurallarına uymasının istenmesidir. İş aktiviteleri istisnalar ve protokol kurallarının getirdiği kısıtlardan dolayı protokol tabanlı atomik hareketlerden de farklıdır (Thomas Earl, 2006).

Örneğin, iş aktivite protokolleri geri alma yeteneğini sunmamaktadır. Bunun yerine telafi işlemleri sağlamaktadırlar. Telafi işlemi herhangi bir istisna oluşması durumunda bir diğer planın işletilmesidir (Thomas Earl, 2006).



Şekil 2-35 Telafi işlemleri

Protokolleri

WS-BusinessActivity de WS-AtomicTransaction gibi WS-Coordination uygulama çatısının özelleştirilmiş hali olan bir koordinasyon tipidir. Birbirine çok benzeyen iki protokol sunmaktadır. Bu protokoller katılımcının aktivite sırasında nasıl davranması gerektiğini belirtmektedirler (WS-BusinessActivity, 2005).

- BusinessAgreementWithParticipantCompletion protokolü, Katılımcının aktivite içerisindeki görevini tamamlayıp tamamlamadığına kendisinin karar vermesini sağlar (WS-BusinessActivity, 2005).

- BusinessAgreementWithCoordinatorCompletion protokolü, katılımcının görevini tamamlayıp tamamlamadığı koordinatör tarafından gönderilen uyarılar ile belirlenir (WS-BusinessActivity, 2005).

İş aktivite katılımcıları WS-Coordination tarafından belirlenmiş olan standartlar ile iletişim kurmaktadırlar (bir protokol için kayıtlanmak vb.).

İş aktivitelerini atomik hareketlerden ayıran bir diğer fark ise, işini bitiren bir servisin aktivite sırasında kalma zorunluluğu olmamasıdır. Kendisine ait katkıları yapan servis aktiviteden ayrılabilmektedir. Bunu gerçekleştirebilmek için koordinatöre bir çıkış uyarı mesajı göndermektedir (Thomas Earl, 2006).

2.5.4.4. BPEL4WS

"Business Process Execution Language for Web Services" (BPEL4WS) XML tabanlı bir programlama dilidir. Bu standart ile Web servis etkileşimi tanımlanabilir ve birlikte çalışma ("cooperation") protokolleri kullanılabilir. Ayrıca BPEL4WS dili, iş süreçleri oluşturmak amacıyla farklı aktivitelerin birleştirilmesinde de kullanılabilir (BPEL4WS 1.0, 2003).

BPEL4WS web servisleri için geliştirilmiş olan iş akışı tabanlı birleşim dilidir. Farklı tiplerde basit aktiviteler içermektedir. Bunlar sayesinde uygulamaların birbirlerini tetiklemelerine ve mesajlaşabilmelerine olanak sunmaktadır. Etkileşim sırasında bekleme sağlamakta yada veriyi bir noktadan diğer bir noktaya taşımayı desteklemektedir. Hata durumlarını gösterebilmekte ve tüm bileşimi sonlandırabilmektedir (Khalaf).

Bu basit aktiviteler yapısal aktiviteler yardımı ile daha karmaşık algoritmalar oluşturma için kullanılabilmektedir. Bunlara örnek olarak belli bir sırada çalıştırma veya koşula göre işletme veya döngüsel işletme gösterilebilir (Khalaf).

BPEL4WS iş akışı modeli aktiviteleri kavrayıp hata ve telafi yakalayıcılarını bu kapsamlar için belirleme yeteneğine sahiptir. Hata yakalayıcı bir istisna fırlatıldığı zaman çalıştırılmaktadır, telafi yakalayıcıları ise hatalara veya telafi aktivitelerine bağlı olarak tetiklenmektedir (Khalaf).

Tamamlayıcı Standartlar

"Web Services Coordination (WS-Coordination)" ve "Web Services Transaction (WS-Transaction)" standartları IBM ve Microsoft tarafından BPEL4WS'i tamamlamak üzere ortaklaşa hazırlanmıştır. Bu standartlar birlikte çalışan işlem hareketleri için genişletilebilir dağıtımlı bir işletim modeli sunmaktadır (WS-Coordination, 2002).

WS-Coordination ve WS-Transaction ile dağıtımlı Web servis operasyonlarında veri tutarlılığı korunmaktadır. Ayrıca bu standartları uygulamalara spesifik protokoller ile birlikte kullanmak da olasıdır.

BPEL4WS, WS-Coordination ve WS-Transaction standartları arasındaki genel etkileşim şu şekilde olmaktadır (WS-Coordination, 2002):

- Süreçlerin servis olarak tanımlanması,
- Sürece ait hangi aktiviteler ile servise doğrudan erişimin sağlanacağının belirlenmesi,

- Çoklu Web servisleri aktivitelerinin genel iş süreci içinde koordine edilmesi,
- Servis sunucuları ile servislerin genel işlem akışından elde edilen veri doğrultusunda dinamik olarak bağlantı sağlanması.

Standart protokoller kullanılarak yönetilen basit etkileşimlerden farklı olarak, sistem entegrasyonları daha yetenekli protokollere ihtiyaç duymaktadırlar. Standart bir işlem entegrasyon modeli kullanılması durumunda, uygulamalar ve iş süreçleri karmaşık etkileşimlerini bütünleştirebilecek ve böylelikle web servislerinin entegrasyon platformu oluşturması başarılacaktır (BPEL4WS 1.1, 2003).

WSDL tarafından desteklenmekte olan etkileşim modeli durumsuz senkron veya ilintisiz asenkron etkileşimler içindir. İş etkileşim modelleri ise farklı olarak senkron/asenkron mesaj değişimleri, birden fazla katılımcılı uzun süreli etkileşim modellerini benimsemektedir. Bu tip etkileşimlerin tanımlanması için, iş süreçleri tarafından etkileşimlerde kullanılan mesaj değişim protokollerinin resmi tanımlarına ihtiyaç duyulmaktadır (BPEL4WS 1.1, 2003).

Bu tip iş protokollerinin tanımlanması, protokol ile ilgili katılımcıların iş yapış yöntemlerini açığa çıkartmadan, karşılıklı görünür olacak şekilde mesaj değişim davranışlarını açıkça belirtmeyi gerektirmektedir (BPEL4WS 1.1, 2003).

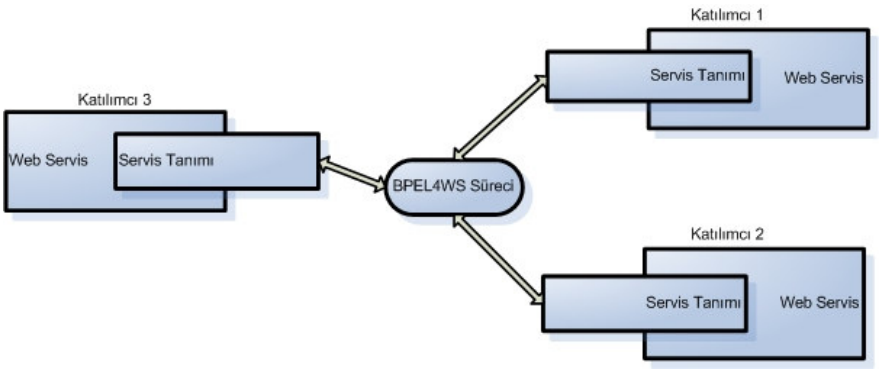
İş süreci davranışını içsel çözüm metotlarından ayırmanın iki önemli sebebi vardır. İşi yapanlar, bu işi nasıl yaptıklarını ortaklarına göstermek istemezler. Ayrıca bir işin gerçekleştirimi değiştiği zaman

sadece içsel dönüşümler yapılmakta, süreç bu değişimden etkilenmemektedir (BPEL4WS 1.1, 2003).

Detay

BPEL4WS, iş akış mantığını tanımlamak için ayrıntılı bir sözdizimi sunmaktadır. İş protokollerini tanımlayan soyut işlemlerin yanı sıra çalıştırılabilir işlemlerin yaratılmasına olanak sağlamaktadır (Thomas Earl, 2006).

Çalıştırılabilir işlem web servis üzerindeki işlem tanımını içermektedir. BPEL4WS dokümanı temel olarak, işlem tarafından yönetilmekte olan servisler ile ilgili akışı ve mantığı belirtmektedir. Üretici bağımlı ortamlarda, özel yazılım sunucuları çalıştırılabilir işlem tanımlarını oluşturmak için orkestrasyon motorları kullanmaktadırlar (Thomas Earl, 2006).



Şekil 2-36 BPEL4WS ve Katılımcılar

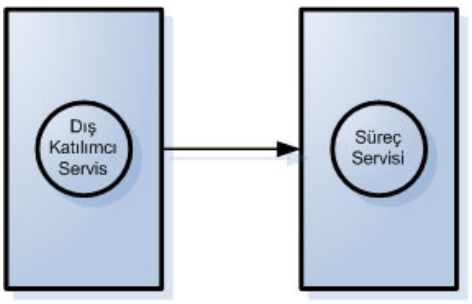
BPEL4WS ve WSDL

BPEL4WS süreç tanımlama, bir çok iş akışı ihtiyacını birbirine uydurabilmek için sayısız dil yapısını birleştirmektedir. BPEL4WS sürecini temsil eden WSDL dokümanı süreç servisi ile ilgili arayüzleri ve sürecin çalışmasında gerekli olan ek servislerin arayüzlerini içermektedir (Prentice Hall, 2004).

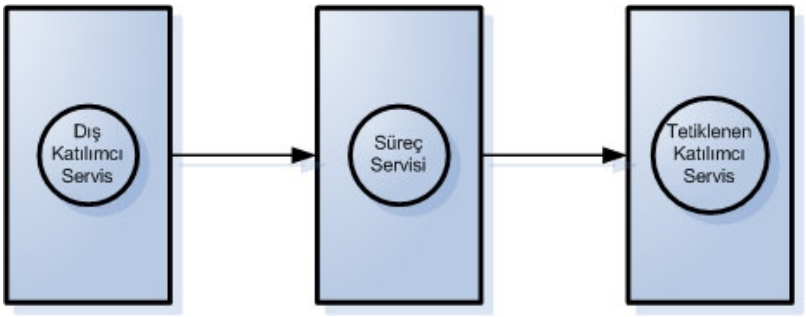
Diğer WSDL dokümanlarından farklı olarak, BPEL4WS süreci servis tanımı bağlantı bilgisi içermemektedir. Sürecin teknik geliştirim ortamından bağımsız olabilmesi için bilerek tanımlanmış bir özelliktir (Prentice Hall, 2004).

Ortak Servisler

BPEL4WS işlemi diğer web servisleri ile 2 farklı yol ile ilişki kurmaktadır (Prentice Hall, 2004).



Şekil 2-37 Süreç servisi, dış bir servis tarafından erişilebilir



Şekil 2-38 Süreç servisi kendisi sürecin çalışmasında yer alan bir servise erişebilir

Bir sürecin içerisinde yer alan servisler ortak (katılımcı) servisler olarak adlandırılmaktadır ve süreç tanımında belirtilmektedirler (Prentice Hall, 2004).

Süreç Örnekleri

BPEL4WS sürecine erişilmek istendiği zaman bu sürecin bir örneği çalışma tamamlanana kadar kalmaktadır. Bu yüzden süreç servisleri ve dış ortak servisler arasında 2 farklı etkileşim senaryosu bulunmaktadır (Prentice Hall, 2004).

- Ortak servis sürecin yeni bir örneğine erişebilir
- Ortak servis sürecin varolan bir örneğine erişebilir

Süreç Tanımları

BPEL4WS süreç tanımı içerisinde önceden tanımlanmış koşullara ve mantığa bağlı olan olaylar akışından oluşan bir iş akışını

tanımlamaktadır. Akış içerisindeki her adım temel aktivite veya yapısal aktivite kullanımı ile gerçekleştirilmektedir (Prentice Hall, 2004).

Temel Aktiviteler

İlkel iş akışı fonksiyonları içermektedir, örneğin;

- Al (receive)
- Tetikle (invoke)
- Cevap ver (reply)
- Fırlat (throw)
- Bekle (wait)

İlk 3 fonksiyon süreç servisi ile ortak servisler arasında ki etkileşimi (servis etkileşimi kısmında anlatılan) olanaklı kılmaktadır (Prentice Hall, 2004).

Süreç sırasında belirli istisnaların oluşması durumunda istisna işleme yöntemleri çalıştırılabilmektedir. Fırlat aktivitesi bu durumda hata koşulunu oluşturmakta ve işleyişi ilgili kısma kaydırmaktadır (Prentice Hall, 2004).

Programlanmış bir çalışma Bekle fonksiyonu ile sağlanabilmektedir. Belirli bir zamana gelene kadar yada belirli bir süre boyunca tüm çalışma beklemeye alınmaktadır (Prentice Hall, 2004).

Yapısal Aktiviteler

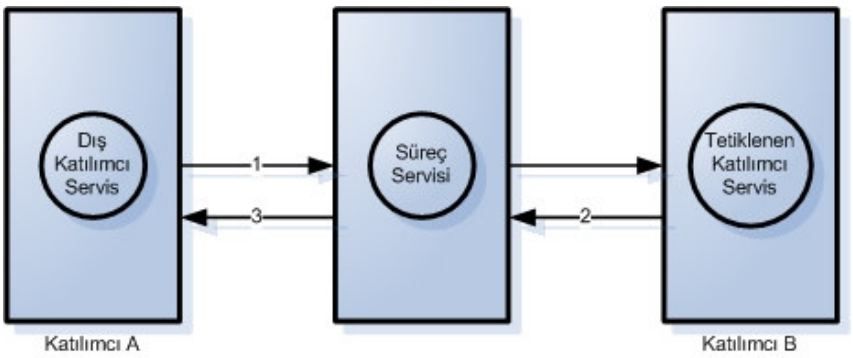
BPEL4WS iş akış mantığı oluşturabilmek için gerekli olan yapısal aktiviteler de sunmaktadır.

- Sıra (sequence)
- Akış (flow)
- Anahtar (switch)
- Süresince (while)

Yapısal aktiviteler temel aktivitelerden nasıl yararlanılacağına karar vermektedir. Hangi sırada çalıştırılacakları sıra aktivitesi ile belirlenebilmektedir. Aktivitelerin hangi sırada yer alacağının liste halinde tutulduğu bir yapı sunmaktadır. Akış aktivitesi ise eşzamanlı çalıştırılan aktivitelerin listesini saklamaktadır (Prentice Hall, 2004).

Anahtar ve süresince aktiviteleri ise select case, do-while loop gibi koşula bağlı mantıksal yapılardır.

Servis etkileşimi



Şekil 2-39 Dış katılımcı servis

Dışsal ortak servis, BPEL4WS süreç servisine mesaj göndererek irtibat kurmakta. Süreç tanımında yer alan Al (receive) aktivitesi A ortağından gelen mesaj için giriş noktası oluşturmakta (Prentice Hall, 2004).

Süreç servisi, süreç tanımından ötürü A ortağından gelen mesaj sonrasında tetikle (invoke) aktivitesi ile B servisi ile iletişim kurmakta (Prentice Hall, 2004).

Tetikle aktivitesi tamamlandıktan sonra süreç aktivitesi cevap ver (reply) aktivitesini kullanarak A servisine cevap vermekte (Prentice Hall, 2004).

SözDizimi

Süreç tanımı daha detaylı incelenirse; Süreç elemanı sürecin kök elemanınıdır.

```
<process name="MyProcess"...>
        <partnerLinks>
                ...
        </partnerLinks>
        <variables>
                ...
        </variables>
        <faultHandlers>
                ...
        </faultHandlers>
        <sequence>
                ...
        </sequence>
</process>
```

BPEL4WS süreç tanımının iskeleti

Bir süreç tanımı aşağıdaki yapıların yığınından oluşabilmektedir (Prentice Hall, 2004):

partnerLinks

partnerLink yapılarının yığını olan partnerLinks, ortak servisleri betimlemek için kullanılmaktadırlar. Süreç sırasında kullanılan servisin isim bilgisini taşımaktadır.

```
<process name="MyProcess"...>
  <partnerLinks>
    <partnerLink name="OrderEntry" ...>
      ...
    </partnerLink>
    <partnerLink name="InventoryControl" ...>
      ...
    </partnerLink>
  </partnerLinks>
</process>
```

variables

Uzun süreli işlemleri destekleyebilmek için durum bilgisi yönetim sistemine ihtiyaç duyulmaktadır. Bu yüzden BPEL4WS "variable" elemanını sağlamaktadır.

Global olan değişkenlere süreç tanımının her noktasından erişilebilmektedir. Değişken değerleri durum bilgisi taşıyan mesajlardan oluşmaktadır.

```
<process name="MyProcess"...>
  <variables>
    <variable xmlns:ORD="http://www.examples.ws/"
      name="OrderStatus"
      messageType="x:OrderStatus"/>
  </variables>
</process>
```

faultHandlers

Uzun süreli bir iş aktivitesinin başarısız olması durumunda süreç telafi işlemine dönecektir. Telafi işlemi, faultHandlers yapısında bulunan bir tip istisna işlemedir. Catch yapılarının kullanımı ile çeşitli hata koşullarına farklı tiplerde tepkiler üretilebilmektedir.

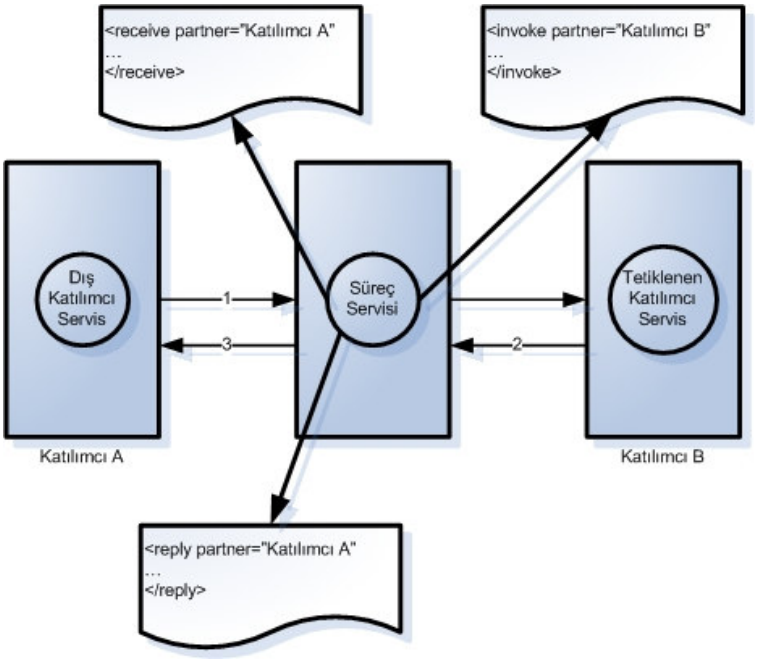
```
<process name="MyProcess"...>
  <faultHandlers>
    <catch faultName="x:condition1" faultVariable="err001">
      ...
    </catch>
    <catchAll>
      ...
    </catchAll>
  </faultHandlers>
</process>
```

Sequence

Etkileşim senaryosuna geri dönülecek olunursa; receive, invoke ve reply aktiviteleri sequence elamanı arasına aşağıda görüldüğü gibi yerleştirilecektir.

```
<process name="MyProcess"...>
  <sequence>
    <receive partner=”partnerA”>
      ...
    </receive>
    <invoke partner=”partnerB”>
      ...
```

```
    </invoke>
    <reply partner=”partnerA”>
      ...
    </reply>
        </sequence>
</process>
```



Şekil 2-40 Süreç iş akışları

Şekilde örnek senaryoda yer alan yapılar gösterilmektedir. Ortaklar arasındaki etkileşim süreç iş akışları içerisinde önceden tanımlanmıştır (Prentice Hall, 2004).

3. GELİŞTİRİLEN UYGULAMA

3.1. Özet

Kurumsal uygulamalarda saklanan tüm gemi bilgilerinin YNA uygulamasından yönetilmesi istenmektedir. Daha öncesinde yeni gemi tanımlanması, güncellenmesi ve silinmesi gibi işlemler varolan AS400 sistemi üzerinden yürütülmektedir.

Yeni uygulamanın gemi ile ilgili fonksiyonları üretim ortamına girdikten sonra AS400 sisteminden gemi ekleme, güncelleme ve silme yetenekleri alınacaktır. Yeni sistemde oluşan bir gemi hareketinin AS400 sistemine yansıtılması gerekmektedir. Bu amaçla Microsoft BizTalk sunucusu kullanılarak bir entegrasyon katmanı geliştirilmiştir.

Bu çalışmada, BizTalk sunucusu tarafından yönetilmekte olan bu sürecin BPEL aracılığı ile yönetimi uygulaması gerçekleştirilecektir.

Gerekli orkestrasyonu ve BPEL kodunu yaratmak amacıyla "Oracle BPEL Process Manager" aracı kullanılmıştır. BPEL standart bir dil olduğundan yaratılan kod farklı bir BPEL motorunda da işletilebilmektedir. Bu çalışmada "Oracle BPEL Process Manager" seçilmesinin temel nedeni, grafiksel arayüz desteği sağlaması ve eğitim amaçlı deneme sürümünün indirilebiliyor olmasıdır.

Oluşturulacak olan entegrasyon çözüm mimarisi, kurumsal uygulama entegrasyonu mimarileri başlığı altında anlatılmış olan mesajlaşma yolu mimarisi ile örtüşmektedir. Servis odaklı mimaride yer alan temel servisler, süreç servisleri ve aracı servisler bu uygulama örneğinde oluşturulacaktır. YNA sistemi, AS400 sistemi ile ilgili olan servisler temel servisler sınıfına girmektedir. YNA sistemi ile AS400

sistemi arasındaki veri dönüşümlerinin yapılması için bir aracı servis oluşturulacaktır. BPEL servisi ise süreç servisidir.

3.1.1. Oracle BPEL Process Manager

OBPM aracı, BPEL iş süreçlerinin tasarlanması ve yönetilmesi için kolay çözümler sağlamaktadır.

BPEL tasarım kısmı, süreçlerin grafiksel arayüz aracılığı ile oluşturulmasını, BPEL motoru ise süreçlerin işletilmesini yönetmektedir. BPEL konsolu, BPEL sunucusuna gönderilmiş olan süreçlerin yönetimi ve test amaçlı adım adım işletilmesi için web tabanlı bir arayüz sunmaktadır.

OBPM aracı http://otn.oracle.com/bpel adresinden indirilebilmektedir.

3.2. Örnek Gemi Entegrasyonu

3.2.1. Ön Hazırlık

YNA ve AS400 Sistemleri Hakkında Bilgi

Geliştirilmekte olan YNA daha önceden de belirtildiği gibi SOA prensiplerine uygun bir yapıda kurulmuştur. Birbirleri ile gevşek bağlı iş parçacıkları içermekte ve bu iş parçacıkları sadece kendi içlerinde "ACID" özellikli hareketleri kullanmaktadır.

Uygulamanın katmanları arasında veri, istek ve cevap nesneleri aracılığı ile taşınmaktadır. Bu nesneler, eklenen metotlar sayesinde tek bir komut ile XML' e dönüştürülebilmektedir.

Sistemi dikine kesen işlemler, boru ("Pipeline") adı verilen yazılımsal parçalar içerisinde, ilgili işlem için yazılmış olan işleyiciler

(“handler”) aracılığı ile gerçekleştirilmektedir. Örnek vermek gerekirse, her işlem için gerekli olan yetki kontrolleri, istek mesajı güvenli geçiş borusundan geçerken yetki işleyicisi aracılığı ile yapılmaktadır. Güvenli geçiş borusunda günlükleme ve benzeri işleyiciler de yer almaktadır.

İş parçacıklarına dış sistemlerin erişimi için istek mesajı alan ve cevap mesajı döndüren web servisler uygulamada yer almaktadır.

AS400 sistemi, YNA sisteminden farklı olarak herhangi bir web servis arayüzü içermemektedir. Bu yüzden, BPEL ile entegrasyon çözümüne başlarken, AS400 sisteminin gemi ekleme, güncelleme ve silme işlemlerini dışa açan web servisleri oluşturulmalıdır.

AS400 uygulaması için web servisinin oluşturulması

AS400 sistemindeki gemi yönetimini sağlamak için BPEL sürecinin yönetebileceği bir web servisinin gerekliliğinden bahsedilmişti. Web servisi oluşturulurken öncelikle girdi ve çıktılar belirlenmiştir. Süreç içerisinde YNA sisteminden gönderilen XML, AS400 sisteminin beklediği XML biçimine dönüştürülmektedir.

Web servisi ASP.NET ortamında geliştirilmiş ve sunucuya aktarılmıştır. Gemi ekleme operasyonunu sunmaktadır.

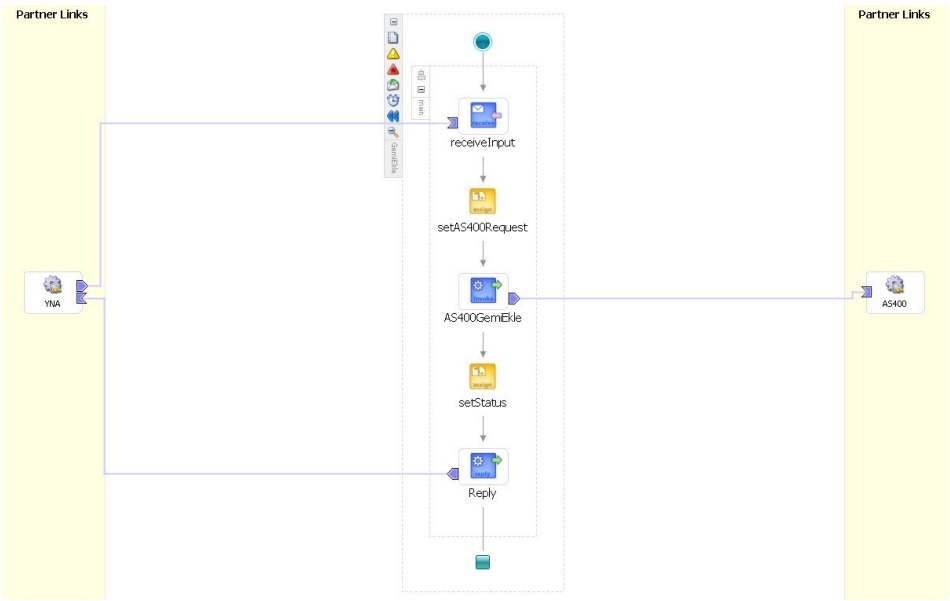
3.2.2. Orkestrasyonun Kurulması – Versiyon 1

AS400 sistemi operasyonları web servisleri haline getirildikten sonra orkestrasyon oluşturulmaya başlanmıştır. Web servis orkestrasyonu ücretsiz bir araç olan ve BPEL oluşturma işlemini grafiksel olarak destekleyen "Oracle BPEL Designer" ile yapılmıştır.

İlk olarak oluşturulan orkestrasyonda istisnai durumlar ve telafi hareketleri göz ardı edilmiştir. Şekil 3.1' de gösterilmekte olan bu orkestrasyonda, YNA sisteminde bir gemi oluşturulursa, AS400 sisteminde de bir gemi oluşturmak için yapılan işlemler gösterilmektedir.

Olay akışı kısaca aşağıdaki adımlardan oluşmaktadır.

1. YNA sisteminde gemi yaratılır.
2. Gemi bilgilerini taşıyan XML gönderilerek orkestrasyon başlatılır.
3. YNA isteğindeki veri, AS400 sisteminin anlayacağı istek mesajı haline çevrilir.
4. AS400 için yazılmış olan web servisindeki gemi ekle metodu hazırlanmış olan istek mesajı ile tetiklenir.



Şekil 3-1 Gemi entegrasyonu versiyon 1

“Partner Links”

Şekil 3.1’ de görüldüğü gibi orkestrasyonda iki adet “partner link” yer almaktadır. Bunlar YNA ve AS400 olarak isimlendirilmiştir.

“Partner Link” lerin oluşan bpel kodunda görünümü şu şekildedir.

<partnerLinks>

<partnerLink name="YNA" partnerLinkType="ns1:Gemi" myRole="GemiProvider" partnerRole="GemiRequester"/>

```
<partnerLink name="AS400" myRole="AS400ServiceProvider"
partnerRole="AS400ServiceProvider"
partnerLinkType="ns2:AS400Service"/>
</partnerLinks>
```

Operasyonlar

Orkestrasyonu yöneten GemiEkle ve AS400 web servislerinin desteklediği operasyonlar ve bekledikleri mesaj tipleri WSDL dosyalar aracılığı ile tanımlanmaktadır.

Orkestrasyon için tanımlı olan operasyonlar şu şekildedir.

```
<portType name="Gemi">
    <operation name="initiate">
        <input message="client:GemiRequestMessage"/>
        <output message="client:GemiResponseMessage"/>
    </operation>
</portType>
```

AS400 için tanımlı olan operasyonlar şu şekildedir.

```
<portType name="AS400Service">
    <operation name="ekle">
        <input name="AS400Request"
message="tns:AS400RequestMessage"/>
```

```
        <output name="AS400Response"
message="tns:AS400ResponseMessage"/>
      </operation>
    </portType>
```

"Variables"

Orkestrasyonda dört adet değişken ("variable") yer almaktadır. YNA sistemi tarafından gönderilen değişken "inputVariable" olarak isimlendirilmiştir. AS400 sisteminin veri aldığı ve sonuç döndürdüğü değişkenler ise "as400Input" ve "as400Output" olarak isimlendirilmiştir. Bpel kodunda değişkenlerin görünümü aşağıdaki gibidir.

```
    <variables>
      <variable name="inputVariable"
messageType="ns1:GemiRequestMessage"/>
      <variable name="outputVariable"
messageType="ns1:GemiResponseMessage"/>
      <variable name="as400Input"
messageType="ns2:AS400RequestMessage"/>
      <variable name="as400Output"
messageType="ns2:AS400ResponseMessage"/>
    </variables>
```

“inputVariable” değişkeni “GemiRequestMessage” mesaj tipindedir. “GemiRequestMessage” mesaj tipi orkestrasyonun WSDL dosyasında tanımlanmaktadır. WSDL dosyası içerisindeki görünümü aşağıdaki gibidir.

```
<message name="GemiRequestMessage">
  <part name="payload" element="client:GemiProcessRequest"/>
</message>
<element name="GemiProcessRequest">
   <complexType><sequence>
         <element name="Id" type="long"/>
         <element name="Ad" type="string"/>
         <element name="BayrakUlkeLokasyonId" type="long"/>
         <element name="GemiSahibiKumpanyaId" type="long"/>
         <element name="GemiTipiId" type="long"/>...
     </sequence></complexType>
</element>
```

Örnek bir uygulama yapıldığından gemi ile ilgili tüm bilgiler dikkate alınmamıştır.

AS400 sistemi için yazılmış olan web servisini tanımlamakta olan WSDL dosyası incelenirse “as400Input” değişkeninin mesaj tipi olan “AS400RequestMessage” yapısı görülecektir.

```
<message name="AS400RequestMessage">
  <part name="gemiad" type="xsd:string"/>
</message>
```

“Sequence”

Görüldüğü gibi YNA sisteminin gönderdiği “inputVariable” değişkeninin “as400Input” değişkenine dönüştürülmesi gerekmektedir. Bu dönüşüm işlemi orkestrasyon mantığı içerisinde gerçekleştirilmektedir. Şekil 3.1’ de grafiksel gösterimi yapılmış olan orkestrasyonun mantıksal kısmı için oluşan kod aşağıdadır.

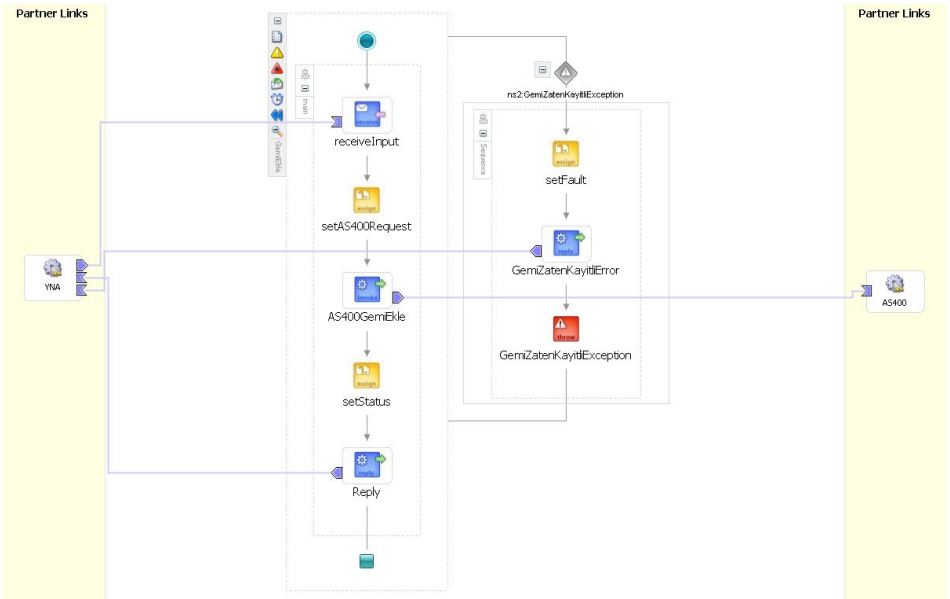
```
<sequence name="main">
  <receive name="receiveInput" partnerLink="YNA"
portType="ns1:Gemi" operation="initiate" variable="inputVariable"
createInstance="yes"/>
  <assign name="setAS400Request">
    <copy>
      <from variable="inputVariable" part="payload"
query="/ns1:GemiProcessRequest/ns1:Ad"/>
      <to variable="as400Input" part="gemiad"/>
    </copy>
  </assign>
```

```
        <invoke name="AS400GemiEkle" partnerLink="AS400"
portType="ns2:AS400Service" operation="ekle"
inputVariable="as400Input" outputVariable="as400Output"/>
        <assign name="setStatus">
          <copy>
            <from variable="as400Output" part="statu"/>
            <to variable="outputVariable" part="payload"
query="/ns1:GemiProcessResponse/ns1:result"/>
          </copy>
        </assign>
        <reply name="Reply" partnerLink="YNA"
portType="ns1:Gemi" operation="initiate"
variable="outputVariable"/>
      </sequence>
    </process>
```

3.2.3. Orkestrasyonun Kurulması – Versiyon 2

Bu kısımda orkestrasyona istisnai durumlar ve istisnai durumların ele alınması eklenecektir.

Örnek uygulamada YNA sisteminde yaratılan geminin AS400 sisteminde zaten kayıtlı olması durumunda nasıl bir yol izleneceği gösterilecektir. Şekil 3.2' de ilgili orkestrasyonun grafiksel gösterimi yer almaktadır.



Şekil 3-2 Gemi entegrasyonu versiyon 2

Şekilde de görüldüğü gibi oluşan istisnanın ismi "GemiZatenKayitliException" dır. Bu istisna ile ilgili tanımlamalar

AS400 WSDL dosyasında yapılmıştır. eklenen kodlar aşağıda görülmektedir.

```
<complexType name="GemiZatenKayitliExceptionType">
  <sequence>
    <any/>
  </sequence>
</complexType>
<message name="GemiZatenKayitliFaultMessage">
  <part name="payload"
type="tns:GemiZatenKayitliExceptionType"/>
</message>
<portType name="AS400Service">
  <operation name="ekle">
    <input name="AS400Request"
message="tns:AS400RequestMessage"/>
    <output name="AS400Response"
message="tns:AS400ResponseMessage"/>
    <fault name="GemiZatenKayitliException"
message="tns:GemiZatenKayitliFaultMessage" />
  </operation>
</portType>
```

Olası istisna tiplerinin yönetimi için orkestrasyon WSDL dosyasında da tanımlamalar yapılması gerekmektedir. Eklenen kodlar aşağıdadır.

```
<types>
  <schema attributeFormDefault="qualified"
elementFormDefault="qualified"
targetNamespace=http://xmlns.oracle.com/Gemi
xmlns="http://www.w3.org/2001/XMLSchema">
...
    <element name="GemiProcessFlowFault"
type="string"></element>
...
  </schema>
</types>

<message name="GemiFlowFaultMessage">
      <part name="payload"
element="client:GemiProcessFlowFault"/>
</message>
```

```
<portType name="Gemi">
  <operation name="initiate">
     <input message="client:GemiRequestMessage"/>
     <output message="client:GemiResponseMessage"/>
     <fault name="TransferFault"
           message="client:GemiFlowFaultMessage">
     </fault>
  </operation>
 </portType>
```

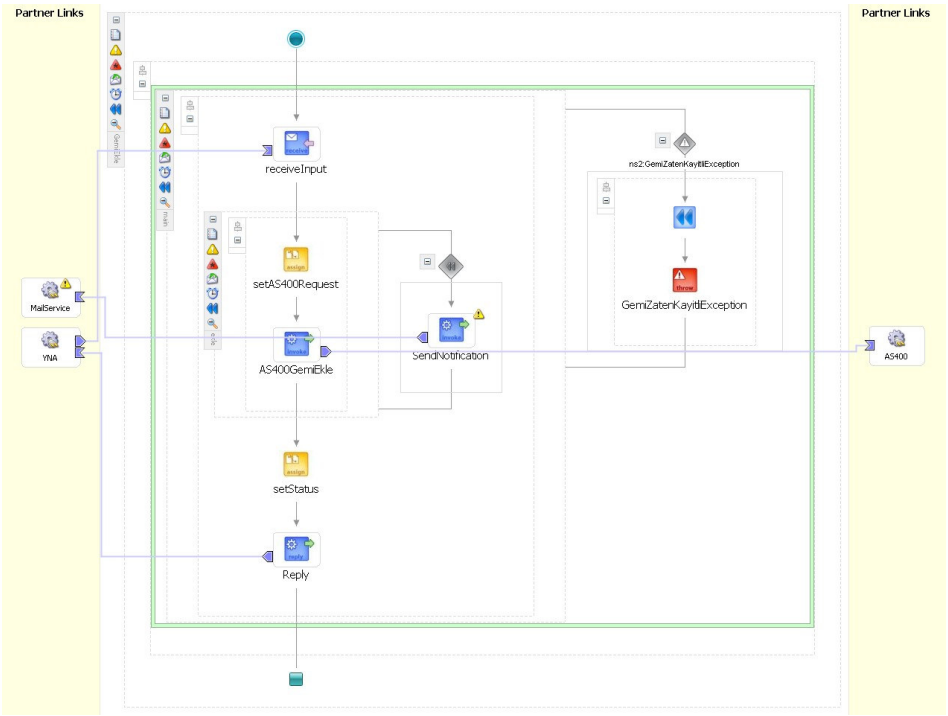
Bpel dosyasında da bazı değişiklikler yapılmıştır. Öncelikle “fault” isimli yeni bir değişken eklenmiştir. Bunun ardından “faultHandlers” kısmı dosyaya eklenmiştir. Yeni değişken ve “faultHandlers” kısmı ile ilgili eklenen kodlar aşağıdadır.

```
...
<variable name="fault"
messageType="ns1:GemiFlowFaultMessage"/>
...
<faultHandlers>
   <catch faultName="ns2:GemiZatenKayitliException">
     <sequence name="Sequence">
       <assign name="setFault">
```

```
<copy>
<from expression="string('GemiKayitli')"/>
<to variable="fault" part="payload"
query="/ns1:GemiProcessFlowFault"/>
</copy>
</assign>
<reply name="GemiZatenKayitliError" partnerLink="YNA"
portType="ns1:Gemi" operation="initiate" variable="fault"
faultName="ns1:TransferFault"/>
<throw name="GemiZatenKayitliException"
faultName="ns1:TransferFault"/>
</sequence>
</catch>
</faultHandlers>
```

3.2.4. Orkestrasyonun Kurulması – Versiyon 3

Bu kısımda telafi hareketleri kullanımı ile ilgili bir örnek yapılacaktır. Geminin AS400 sisteminde tanımlı olması durumunda orkestrasyon içerisinde telafi sürecinin BPEL dili ile nasıl tetiklendiği gösterilmektedir. Grafiksel gösterim Şekil 3.3' te görülmektedir.



Şekil 3-3 Gemi entegrasyonu versiyon 3

Şekilde de görüldüğü gibi "GemiZatenKayitli" istisna yönetimi aynen durmaktadır. Ancak versiyon 2' deki işleyiş değiştirilmiştir. İstisna yakalandığı an orkestrasyonda tanımlanmış olan telafi hareketi tetiklenmektedir. Bu örnekte telafi hareketi kapsamında uyarı gönderme

senaryosu işletilmiştir. Bu kısımda iş ihtiyaçlarına göre alternatif senaryolar üretilebilir.

Telafi hareketi eklenmesi sonucunda ilgili bpel dosyasının son hali aşağıdaki şekildedir.

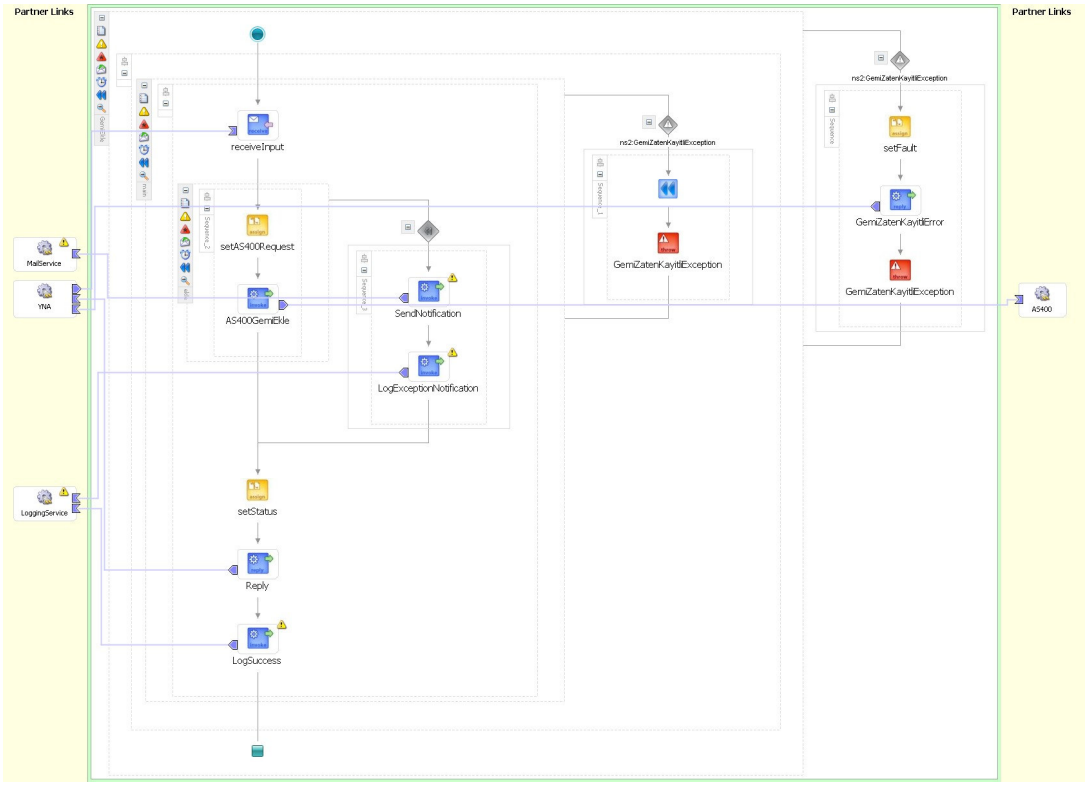
```
<scope name="main">
  <faultHandlers>
    <catch faultName="ns2:GemiZatenKayitliException">
      <sequence>
        <compensate scope="ekle"/>
        <throw name="GemiZatenKayitliException"
faultName="ns2:GemiZatenKayitliException"/>
      </sequence>
    </catch>
  </faultHandlers>
  <sequence>
    <receive name="receiveInput" partnerLink="YNA"
portType="ns1:Gemi" operation="initiate" variable="inputVariable"
createInstance="yes"/>
    <scope name="ekle">
      <compensationHandler>
```

```
<invoke name="SendNotification"
partnerLink="MailService"/>
</compensationHandler>
<sequence>
<assign name="setAS400Request">
<copy>
<from variable="inputVariable" part="payload"
query="/ns1:GemiProcessRequest/ns1:Ad"/>
<to variable="as400Input" part="gemiad"/>
</copy>
</assign>
<invoke name="AS400GemiEkle" partnerLink="AS400"
portType="ns2:AS400Service" operation="ekle"
inputVariable="as400Input" outputVariable="as400Output"/>
</sequence>
</scope>
<assign name="setStatus">
<copy>
<from variable="as400Output" part="statu"/>
<to variable="outputVariable" part="payload"
query="/ns1:GemiProcessResponse/ns1:result"/>
</copy>
```

```
            </assign>
            <reply name="Reply" partnerLink="YNA"
portType="ns1:Gemi" operation="initiate"
variable="outputVariable"/>
          </sequence>
        </scope>
```

3.2.5. Orkestrasyonun Kurulması – Versiyon 4

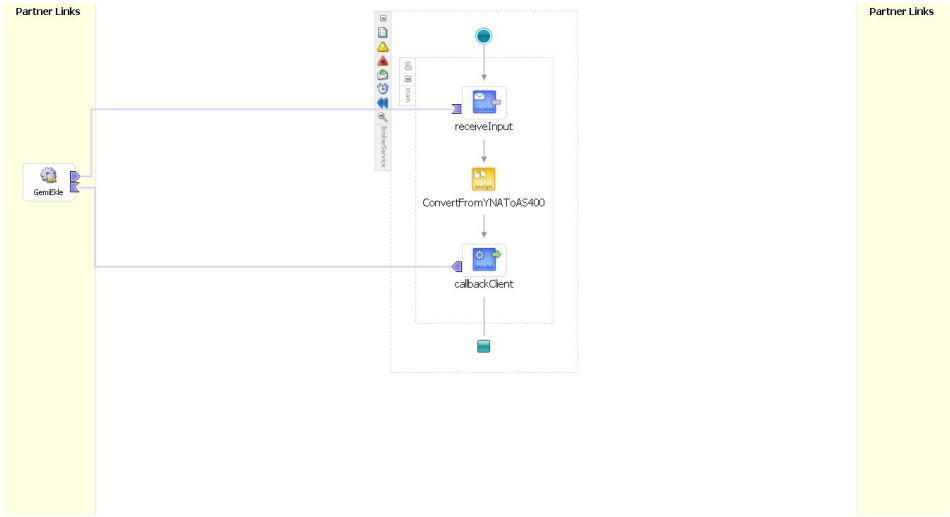
Bu kısımda orkestrasyona günlükleme servisi ile etkileşim eklenecektir. Geminin AS400 sisteminde tanımlı olması durumunda çalıştırılan telafi hareketi içerisinde ve işlem sona ermeden hemen önce bu servis tetiklenmektedir. O anki süreç durumuna göre günlükleme bilgisi oluşturmaktadır. Grafiksel gösterim Şekil 3.4' te görülmektedir.



Şekil 3-4 Gemi entegrasyonu versiyon 4

3.2.6. Orkestrasyonun Kurulması – Versiyon 5

Bu kısımda, daha önceden değişkenlerin birbirleri ile eşleştirilmeleri ile sağlanan veri dönüşümü öncelikle farklı bir servis haline dönüştürülecek, bu çalışma tamamlandıktan sonra süreci yöneten "GemiEkle" isimli servisin oluşturulan aracı servisi nasıl tetiklediği gösterilecektir. Şekil 3-5' te YNA sisteminden gelen isteği, AS400 sisteminin anlayacağı dile dönüştüren aracı servis grafiksel olarak gösterilmektedir.



Şekil 3-5 Gemi entegrasyonu versiyon 5-Aracı Servis

Aracı servisi tetikleyen şekilden de görüldüğü gibi "GemiEkle" servisidir. Burada "GemiEkle" servisi "PartnerLinks" kısmında tanımlanmıştır. Aracı servise girdi olarak YNA sisteminden gelen yeni

gemi bilgilerini iletmekte, çıktı olarak AS400 sistemi için uygun biçimde olan yeni gemi bilgilerini almaktadır.

Şekil 3-6' da sürecin son hali görülmektedir. "setAS400Request" isimli "assign" özelliği yerine, aynı isimde olan ancak aracı servisi tetikleyen bir metot eklenmiştir. Böylelikle geleneksel kurumsal uygulama entegrasyonu mimarilerinde yer alan orkestrasyon ve aracı bileşenler, standartlara dayanan entegrasyon çözümünde servisler olarak sunulabilmiştir.

Şekil 3-6 Gemi entegrasyonu versiyon 5

3.2.7. Orkestrasyonun Kurulması – Versiyon 6

Bu kısımda AS400 sistemine gemi eklenmek istendiği zaman, geminin zaten kayıtlı olması dışında, neden olduğu bilinmeyen bir hatadan dolayı AS400 sisteminde kayıt yaratılamaması durumunda işletilecek olan senaryo sürece eklenecektir.

Bu gibi durumlarda ilgili gemi kaydının YNA sisteminden fiziksel olarak silinmesi istenmemektedir, bunun yerine kaydın durumunun kullanılamaz hale getirilmesi gerekmektedir.

Bu gereksinimin karşılanması için öncelikle sürece yeni bir hata tipi eklenmiştir. Çalışma sırasında bu hata tipi oluştuğu taktirde süreçteki hataların yazıldığı "fault" değişkenine bilinmeyen hata oluştu bilgisi verilmektedir. Ayrıca bu hata tipi yakalandığı zaman yine bir telafi hareketi işletilmekte, bu telafi hareketinin içerisinde YNA sisteminde bulunan gemi kaydının kullanılamaz hale getirilmesini sağlayan YNA servisi tetiklenmektedir.

Bu işlem için yeni bir telafi hareketi oluşturulmamıştır. Varolan telafi hareketi tanımına "switch" yapısı eklenmiş, hata değişkeninin değerine göre gerekli operasyonlar tetiklenmiştir.

Telafi hareketinin "switch" yapısı eklendikten sonraki durumu aşağıda gösterilmektedir.

```
...
<compensationHandler>
        <sequence name="">
         <switch name="HataTipi">
```

```
                <case/>
                <case
condition="bpws:getVariableData('fault','payload','/ns1:GemiProcessFl
owFault') = string(&amp;quot;UnknownError&amp;quot;)">
                 <bpelx:annotation>
                  <bpelx:pattern>UnknownError
                  </bpelx:pattern>
                 </bpelx:annotation>
                 <sequence name="Sequence_6">
                  <invoke name="YNAGemiPasifYap"
partnerLink="YNA" portType="ns1:GemiCallback"
operation="gemipasifyap"/>
                  <invoke name="LogProcess"
partnerLink="LoggingService"/>
                 </sequence>
                </case>
                <otherwise>
                 <sequence name="Sequence_5">
                  <invoke name="SendNotification"
partnerLink="MailService"/>
                  <invoke name="LogExceptionNotification"
partnerLink="LoggingService"/>
```

</sequence>

</otherwise>

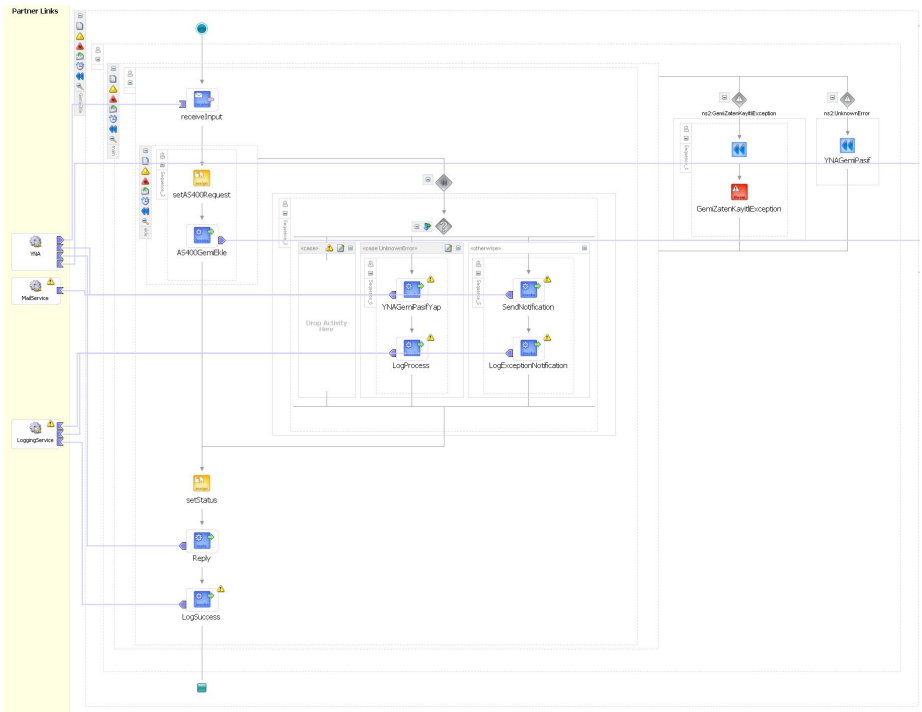
</switch>

</sequence>

</compensationHandler>

...

Sürecin grafiksel gösterimi Şekil 3-7' de görülmektedir.



Şekil 3-7 Gemi entegrasyonu versiyon 6

4. SONUÇ

Geliştirilen uygulamada ile iki sistem arasında var olan ve bir üreticiye ait olan bir araç yardımı ile geliştirilmiş olan bir entegrasyon çözümünün basit bir kısmı BPEL4WS ile geliştirilmeye çalışılmıştır.

Birbirleri ile entegrasyona giren sistemler örnekte .Net ile geliştirilmiş bir uygulama ve varolan bir SAP uygulamasıdır. Kurumun büyüklüğü, iş hacmi ve süreçlerinin karmaşıklığına göre birbirleri ile etkileşim içerisinde olacak olan uygulamaların sayısının arttığı düşünülecek olursa bu tip entegrasyon katmanlarının farklı katmanlar olarak geliştirilmesinin önemi daha iyi biri şekilde anlaşılabilecektir.

Bu katmanlar geliştirilirken herhangi bir üreticinin standartlarına bağlı kalmak daha önceden de belirtildiği gibi sıkıntılara yol açacaktır. Uygulamadan görülebileceği üzere, yeni geliştirilen sistemlerin servis odaklı mimari prensiplerine uygun olarak geliştirilmesi ve varolan sistemlerin gerekli olan operasyonları için web servisleri veya adaptörler yazılması ile uygulamalar arasındaki etkileşim, entegrasyon ve süreç yönetimi gibi konularda BPEL etkin bir şekilde kullanılabilecektir.

Bu sayede açık standartlara bağlı kalınarak gerçekleştirilen uygulamaların gücüne erişilebilecektir. Bunun en iyi örneği, bir çok veritabanı üreticisinin veri sorgulama dili olarak SQL' i kullanmasıdır. BPEL de iş süreçleri yönetimi için oldukça önemli bir role sahip bir açık standarttır.

BPEL süreçlerini tasarlamaya ve işletmeye yarayan araçlar klasik kurumsal uygulama entegrasyonu araçlarına göre oldukça ucuzdur. Bu

önemli bir maddi kazanç sağlayacaktır. Ayrıca farklı sistemler için farklı adaptörler alınması gerekmeyeceğinden, hem bu adaptör maliyetlerinden, hem de eğitim maliyetlerinden kurtulunacaktır.

KAYNAKLAR DİZİNİ

Fehmi Hüdayioğlu, 2006, "SOA ve ESB Nedir?",
http://turk.internet.com/haber/yazigoster.php3?yaziid=14659

ServiceOrientation, 2006,
http://www.serviceorientation.org/

Selda Güner, 2005, "Architectural Approaches, Concepts and Methodologies of Service Oriented Architecture", Master Thesis, www.sts.tu-harburg.de/pw-and-m-theses/2005/gune05.pdf

Enterprise SOA , 2004, The COAD Series, "Service –Oriented Architecture Best Practices", ISBN: 0-13-146575-9

Service Oriented Architecture, 2004, Prentice Hall, "A Field Guide to Integrating XML and Web Services",
ISBN: 0-13-142898-5

Pehepe, 2006
http://www.pehepe.org/dersler.php?sayfa=2&ana=2&alt=5&dersno=7

W3Schools, 2006, "Web Services Tutorial"
http://www.w3schools.com/webservices/ws_intro.asp

KAYNAKLAR DİZİNİ (devam ediyor)

WS-Coordination, 2005, Web Services Coordination Specification, Version 1.0 August 2005 Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machines Corporation, IONA Technologies, Microsoft Corporation, Inc.

http://specs.xmlsoap.org/ws/2004/10/wscoor/wscoor.pdf

WS-Coordination, 2002, Web Services Coordination Specification, BEA Systems, International Business Machines Corporation, Microsoft

http://www-106.ibm.com/developerworks/webservices/library/ws-coor

WS-AtomicTransaction, 2005, Web Services AtomicTransaction Specification, Version 1.0 August 2005 Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machines Corporation, IONA Technologies, Microsoft Corporation, Inc.

http://specs.xmlsoap.org/ws/2004/10/wsat/wsat.pdf

WS-BusinessActivity, 2005, Web Services Business Activity Framework Specification, Version 1.0 August 2005 Arjuna Technologies, Ltd., BEA Systems, Hitachi, Ltd., International Business Machines Corporation, IONA Technologies, Microsoft Corporation, Inc. http://schemas.xmlsoap.org/ws/2004/10/wsba/

BPEL4WS 1.1, 2003, Business Process Execution Language for Web Services Specification, Version 1.1 5 May 2003 BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems

http://www.oasisopen.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf

KAYNAKLAR DİZİNİ (devam ediyor)

Thomas Earl, 2006, "WS-Standards", http://www.ws-standards.com/

F. Leymann, D. Roller, M.-T. Schmidt, 2001, "Web services and business process management", IBM Systems Journal http://www.research.ibm.com/journal/sj/412/leymann.html

BPEL4WS 1.0, 2003, "Business Process Execution Language for Web Services 1.0**",** BEA, IBM, Microsoft http://www106.ibm.com/developerworks/webservices/library/ws-bpel/

SOAP Version 1.2, 2003, "Part 1: Messaging Framework**",** W3C Recommendation, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

Arthur Ryman, 2003, "Understanding Web Services**",** http://www128.ibm.com/developerworks/websphere/library/techarticles/0307_ryman/ryman.html#N10048

W3Schools, 2006, "WSDL Tutorial" http://www.w3schools.com/wsdl/wsdl_intro.asp

KAYNAKLAR DİZİNİ (devam ediyor)

WSDL, 2006, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", W3C Candidate Recommendation http://www.w3.org/TR/2006/CR-wsdl20-20060327

Mark Little-Thomas Freund, 2003, "A comparison of Web services transaction protocols", http://www.ibm.com/developerworks/webservices/library/ws-comproto/

Thomas Freund, 2002, "An overview of WS-Transaction and WS-Coordination",

http://www.ibm.com/developerworks/webservices/library/ws-wstx2

Rania Khalaf, Nirmal Mukhi, Sanjiva Weerawarana, "Service-Oriented Composition in BPEL4WS", IBM T.J. Watson Research Center

http://www2003.org/cdrom/papers/alternate/P768/choreo_html/p768-khalaf.htm

Net.ObjectDays, 2002, "Web Services between Demand and Reality" http://www.inubit.de, 2002

KAYNAKLAR DİZİNİ (devam ediyor)

Understanding BizTalk Server 2006, 2005
David Chappel

ServiceOriented.org, 2006
www.serviceoriented.org

Haluk Cavkaytar, 2002, XML & Java Teknolojilerini kullanarak gerçekleştirilen bir B2B projesi, Ege Üniversitesi SBS

EKLER

EK1 Sözlük

Servis Odaklı Mimari	Service Oriented Architecture
Orkestrasyon	Orchestration
Aracı	Broker
İş Süreçleri Yönetimi	Business Process Management
İş Süreçleri Yönetimi Sistemi	Business Process Management System
İş Süreçleri Modelleme Dili	Business Process Modeling Language
Web Servisleri için İş Süreci İşletme Dili	Business Process Execution Language for Web Services
Hareket	Transaction
İki Aşamalı Onay	Two Phase Commit
Web Servisler Tanımlama Dili	Web Services Definition Language
Basit Nesne Erişim Modeli	Simple Object Access Protocol

Kurumsal Uygulama Entegrasyonu	Enterprise Application Integration
Sıkı Bağlı	Tightly coupled
Gevşek Bağlı	Loosely coupled
Birlikte işlerlik	Interoperability
Taşınırlık	Portability
Saydamlık	Transparency
Telafi	Compensation
Mesajlaşma Yolu	Messaging Bus
Kurumsal Servis Yolu	Enterprise Service Bus
Uç Nokta	Endpoint
Özerk	Autonomous
Durumsuz	Stateless
Koçan	Stub
Giriş yolu	Gateway
Dönüştürücü	Adaptor

EK2 Kısaltmalar

SOA	“Service Oriented Architecture”
BPEL4WS	“Business Process Execution Language for Web Services”
BPMN	“Business Process Modeling Notation”
WSFL	“Web Services Flow Language”
ACID	“Atomicity Consistency Isolation Durability”
OLTP	“Online Transaction Processing”
2PC	“Two Phase Commit”
WS	“Web Services”
WSDL	“Web Services Definition Language”
SOAP	“Simple Object Access Protocol”
EAI	“Enterprise Application Integration”
WS-COO	“WS-Coordination”
WS-TXN	“WS-Transaction”

ÖZGEÇMİŞ

Kişisel Bilgiler:

Adı Soyadı	: Serkan ÜSTÜNDAĞ
Doğum Tarihi	: 18/01/1980
Cinsiyet	: Bay
Medeni Hali	: Bekar

Eğitim Durumu:

2002 – ...	: Yüksek Lisans, EÜ Bilgisayar Mühendisliği, İzmir
1997 – 2002	: Lisans, EÜ Bilgisayar Mühendisliği, İzmir
1990 – 1997	: Yunus Emre Anadolu Lisesi, İzmir

İş Deneyimi:

08. 2005 - ...	: BİMAR / ARKAS, İzmir Yazılım Uzmanı
11. 2003 – 08. 2004	: TEBA Yazılım Mühendisi
11. 2002 – 10. 2003	: UNIVERA Yazılım Uzmanı
08.2001 – 10.2001	: Ford Motor Company, Köln Stajyer