T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

SOAP KULLANILARAK ENTEGRASYON SERVİSLERİNİN GELİŞTİRİLMESİ

Yüksek Lisans Tezi

OKAN PULUKÇU

İSTANBUL, 2014

T.C.

BAHÇEŞEHİR ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİ TEKNOLOJİLERİ YÜKSEK LİSANS PROGRAMI

SOAP KULLANILARAK ENTEGRASYON SERVİSLERİNİN GELİŞTİRİLMESİ

Yüksek Lisans Tezi

OKAN PULUKÇU

Tez Danışmanı: PROF.DR. ADEM KARAHOCA

İSTANBUL, 2014

T.C.
BAHÇEŞEHİR ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİ TEKNOLOJİLERİ YÜKSEK LİSANS PROGRAMI

Tezin Adı: SOAP Kullanılarak Entegrasyon Servislerinin Geliştirilmesi
Öğrencinin Adı Soyadı: Okan PULUKÇU
Tez Savunma Tarihi: 29.05.2014

Bu tezin Yüksek Lisans tezi olarak gerekli şartları yerine getirmiş olduğu Fen Bilimleri Enstitüsü tarafından onaylanmıştır.

Doç.Dr. Tunç BOZBURA
Enstitü Müdürü
İmza

Bu tezin Yüksek Lisans tezi olarak gerekli şartları yerine getirmiş olduğunu onaylarım.

Prof.Dr. Adem KARAHOCA
Program Koordinatörü
İmza

Bu Tez tarafımızca okunmuş, nitelik ve içerik açısından bir Yüksek Lisans tezi olarak yeterli görülmüş ve kabul edilmiştir.

Jüri Üyeleri	İmzalar
Tez Danışmanı Prof.Dr. Adem KARAHOCA	----------
Üye Y.Doç.Dr. Dilek KARAHOCA	----------
Üye Y.Doç.Dr. Yalçın ÇEKİÇ	----------

Sevgili Eşim Karin'e…

TEŞEKKÜR

Bu tez çalışmasının planlanmasında, araştırılmasında, yürütülmesinde ve oluşumunda ilgi ve desteğini esirgemeyen, engin bilgi ve tecrübelerinden yararlandığım, yönlendirme ve bilgilendirmeleriyle çalışmamı bilimsel temeller ışığında şekillendiren sayın hocam Prof.Dr. Adem Karahoca'ya sonsuz teşekkürlerimi sunarım.

İstanbul, 2014 Okan Pulukçu

ÖZET

SOAP İLE ENTEGRASYON SERVİSLERİNİN GELİŞTİRİLMESİ

Okan Pulukçu

Bilgi Teknolojileri Yüksek Lisans Programı

Tez Danışmanı: Prof.Dr. Adem Karahoca

Mayıs 2014, 64 Sayfa

Her geçen gün artan yazılım ihtiyacı ile yükselen proje geliştirme maliyetleri, birçok uygulamanın birlikte çalışarak tekrar kullanılabilir olmasını, dolayısıyla uygulamaların birbirleri ile entegrasyonunu gerekli kılmıştır. Entegrasyon katmanı üzerinde, yazılım uygulamalarının servislerini entegre etmek için hazırlanan servislerin, geliştirme ve yönetim maliyeti gittikçe gelişen sistemlerde artmaktadır. SOA kullanan entegrasyon platformlarının, yönetiminin ve entegrasyon servislerinin geliştirilmesinin kolaylaştırılması bir ihtiyaç haline gelmektedir.

Birinci bölümde, entegrasyon ihtiyacının sebepleri, bu ihtiyacının ortaya çıktığı ilk yıllarda nasıl karşılandığı, SOA modelinin doğuşu, SOAP ile entegrasyon servislerinin geliştirilmesinin anlaşılabilmesi için bilinmesi gereken bilgiler, "SOAP ile Entegrasyon Servislerinin Geliştirilmesi" tez konusunun seçiminde etkili olan faktörler, yapılan çalışmanın bilime yapacağı katkılar ve çalışmanın amacı üzerinde durulmuştur.

İkinci bölümde, SOA modeli ile ilgili çalışmalar incelenmiştir. Tezin konusuna ilişkin saptanan olası sorunlar belirlenmiştir. Bu sorunlar arasında çok katmanlı bir mimari yapıda çalışma yapılacak olması, tez konusunca geliştirilecek yazılımın karmaşıklığını arttıracağı ve test süresince alınacak hataların izlenmesinin zorlaşacağı belirlenmiştir.

Üçüncü bölümde, SOA modeli ve ESB şablonu açıklanmış, entegrasyon platformlarından TIBCO ve IBM ESB ile entegrasyon servisi geliştirme ve yönetme prosedürleri anlatılmıştır.

Dördüncü bölümde, incelenen entegrasyon platformlarının dezavantajları ve bu dezavantajlara yönelik çözümlerden bahsedilmiştir. Entegrasyon servisi geliştirme ve yönetiminin Lightweight SOA modeli ile kolaylaştırılabileceği ortaya konmuştur. Bu modele örnek olabilecek bir "Web Service Test ve Entegrasyon Aracı" geliştirilmiştir. Geliştirilen bu yazılımda web servislerin test edilmesi ve web servislerin birbirleri ile entegrasyonu Lightweight SOA modeli ile gerçekleştirilmiştir.

Anahtar Kelimeler: SOA, ESB, SOAP, Lightweight SOA, Entegrasyon

ABSTRACT

DEVELOPMENT OF INTEGRATION SERVICES WITH SOAP

Okan Pulukçu

Information Technologies Master Program

Thesis Supervisor: Prof.Dr. Adem Karahoca

May 2014, 64 Pages

Day by day, software project development costs rising with growing software needings, many applications working together to be used again, so it was necessary to integrate applications with each other. On the integration layer, development and management costs of services which are prepared to integrate software application services are steadily increasing on growing systems. Management and integration service development on SOA using integration platforms are becoming a need to facilitate.

In the first part, the reasons for the need for integration, how the emergence of these needs are met in the first years, the emergence of SOA model, information of need to know in order to understand the development of integration services with SOAP, the factors that influence the choice of thesis topic, contributions to science and the aim of the study were focused on.

In the second part, studies about SOA were examined. Identified potential issues related to the topic of the thesis is determined. Among these problems; attempting to building a multi-tier architecture increase complexity of software to be developed and during testing period determined errors were difficult to monitoring.

In the third part, SOA model and ESB pattern are described, developing and managing the integration service procedures are described on TIBCO and IBM ESB integration platforms.

In the fourth parth, disadvantages of investigated integration platforms and their soluitons are mentioned. Integration service development and management can be facilitated by Lightweight SOA model have been revealed. As an example of this model "Web Service Test ve Integration Tool" developed. By the developed software, testing of web services and web service integration with each others are accomplished with Lightweight SOA model.

Keywords: SOA, ESB, SOAP, Lightweight SOA, Integration

İÇİNDEKİLER

TABLOLAR

ŞEKİLLER

KISALTMALAR

API	:	Application Programming Interface
ASP	:	Active Server Pages
ATM	:	Automatic Teller Machine
BA	:	Backend Application
BO	:	Business Objects
BPM	:	Business Process Management
BW	:	Business Works
CGI	:	Common Gateway Interface
CORBA	:	Common Object Request Broker Architecture
CPY	:	Copybook
CRM	:	Customer Relationship Management
EA	:	Enterprise Architecture
EAR	:	Enterprise Archive
EJB	:	Enterprise JavaBeans
EMS	:	Enhanced Message Service
ESB	:	Enterprise Service Bus
FTP	:	File Transfer Protocol
Hostcpyadı	:	Host copybook adı
Hostprgadı	:	Host program adı
HTTP	:	Hyper-Text Transfer Protocol
IA	:	Integration Architecture
IBE	:	Backend Interface
IDE	:	Integrated Development Environment
IFE	:	Frontend Interface
IMS	:	IBM Information Management System
ISAPI	:	Internet Server Application Programming Interface
J2EE	:	Java Platform Enterprise Edition
JAX-WS	:	Java API for XML Web Services
JDBC	:	Java-based Data Access Technology
JDK	:	Java Development Kit

JMS	:	Java Message Service
JNDI	:	Java Naming and Directory Interface
JSP	:	Java Server Pages
PGM	:	Program
PREPROD	:	Pre-production
PROD	:	Production
REST	:	Representational State Transfer
RMI	:	Remote Method Invocation
SCA	:	Service Component Architecture
SLA	:	Service Level Agreement
SMTP	:	Simple Mail Transfer Protocol
SOA	:	Service Oriented Architecture
SOAP	:	Simple Object Access Protocol
SVN	:	Apache Subversion Client
TFS	:	Team Foundation Server
TRNXID	:	Transaction Id
UAT	:	User Acceptance Testing
UDDI	:	Universal Description, Discovery and Integration
UML	:	Unified Modeling Language
URL	:	Uniform Resource Locator
W3C	:	World Wide Web Consortium
WID	:	Websphere Integration Developer
WSDL	:	Web Service Definition Language
XML	:	Extensible Markup Language
XPATH	:	XML Path Language
XSD	:	XML Schema Definition
XSL	:	Extensible Stylesheet Language
XSLT	:	Extensible Stylesheet Language Transformations

1. GİRİŞ

Her geçen gün gelişen bilişim sektörü, yeni dünya ekonomisinde yadsınamaz bir öneme sahiptir. Mevcut süreçlerin bilişim ile çözümlenmesi ihtiyacı ve sürekli çoğalan teknolojik talepler, bilgi teknolojilerinde farklı platformlarda birçok uygulama geliştirilmesine ve proje sayısının artmasına yol açmaktadır.

Artan teknoloji ihtiyacı doğrultusunda birçok uygulamanın birlikte çalışması gerekliliği, uygulamaların birbirleri ile entegrasyonunu gerekli kılmıştır. Entegrasyon ihtiyacı sebebiyle; uygulama servislerinin haberleşme yapılarının kurulması, meta-data dönüşümlerinin yapılması, günlük tutma (loglama), kimlik doğrulama (*authentication*), yetki kontrolü (*authorization*) için farklı ya da aynı platformlarda geliştirilmiş uygulamalar arasında yer alan entegrasyon katmanı oluşturulmuştur. Kurumlar entegrasyon katmanını yönetmek için çeşitli entegrasyon platformları (*middleware*) kullanmaktadırlar.

Bu platformlar üzerinde, yazılım uygulamalarının servislerini entegre etmek için hazırlanan entegrasyon servislerinin geliştirme ve yönetim maliyeti gittikçe artmıştır. Çoğu zaman farklı platformlarda yer alan basit sayılabilecek iki web servisin haberleşmesi için hazırlanan bir entegrasyon servisinin geliştirme, test ve kurulum maliyeti üç adam/gün olarak hesaplanmaktadır. Daha fazla servis içeren daha karmaşık entegrasyon servislerinde bu maliyet artmaktadır.

Servis Yönelimli Mimari (*Service Oriented Architecture* - SOA) kullanan entegrasyon platformlarının yönetiminin ve entegrasyon servislerinin geliştirilmesinin kolaylaştırılması bir ihtiyaç haline gelmiştir. Ayrıca, küçük ve orta ölçekli işletmelerin maliyet açısından SOA modelini kullanan entegrasyon platformları satın almakta zorlanmaları yine basitleştirilmiş (*lightweight*) SOA ihtiyacını doğurmaktadır. Bu yukarıda anlatılan ihtiyaçlardan dolayı entegrasyon servisleri *Simple Object Access Protocol* (SOAP) kullanılarak *lightweight* SOA metodolojisi ile geliştirilmelidir.

1.1 SOAP İLE ENTEGRASYON SERVİSLERİNİN GELİŞTİRİLMESİNİN ANLAŞILABİLMESİ İÇİN BİLİNMESİ GEREKEN ÖN BİLGİLER

1.1.1 SOA Öncesi

Mainframe sistemleri, 1960'lı ve 70'li yıllarda nadiren birbirleri ile haberleşirdi. Tek bir donanım ile tüm fonksiyonları sağlamak, bu tip sunucuların kullanım sebebiydi. Haberleşme, teyp verinin transferi söz konusu olduğunda yapılmaktaydı. İlerleyen yıllarda, sistemler arası gerçek zamanlı erişim, özellikle bir organizasyon içinde birden fazla sistem bulunmaya başlayınca ihtiyaç haline geldi.

Özellikle finansal piyasalarda, gerçek zamanlı (*online*) işlem yapılması ihtiyacı gittikçe arttı. Başlangıçta, *online* erişim düşük seviye soket haberleşmeleri ile sağlanmaktaydı. Genellikle, Assembly ve C programlama dilinde yazılan soket programları karmaşık ve altyapısını oluşturan ağ protokolleri sebebiyle ileri düzeyde bilgi birikimi gerektiriyordu. Bu sebeple, bazı şirketler mesajlaşma ve haberleşmeyi kolaylaştırmak için orta-seviye yazılımlar geliştirmeye başladılar.

Sonunda, *Remote Procedure Call* (RPC) ile dağıtık yapıda uygulamalar geliştirilmeye başlandı. Böylece RPC ile farklı fonksiyonlar, uzak bilgisayarlar tarafından çağrılabilir ve kullanılabilir hale geldi.

1980'li yıllarda, kişisel bilgisayarlar (*Personal Computer* - PC) üretildi ve geliştiriciler masaüstü bilgisayarların daha etkin kullanım yollarını araştırmaya başladılar. Donanım fiyatlarının her geçen gün düşmesi, kurumsal sunucuların sayısını oldukça arttırdı. Bu *trend* RPC'nin olgunluğunun artmasıyla daha da hızlandı ve dağıtık bilgisayar mimarisinde iki önemli ilerlemenin lideri oldu.

1991 yılında, *Common Object Request Broker Architecture* (CORBA) programlama fonksiyonlarının dağıtık kullanımını standartlaştıran bir kavram olarak ortaya çıkmıştır. Tam olarak standartlaşamadığı için kabulü biraz gecikmeli olmuştur. 1998'de çıkan 2.0 versiyonu ile birlikte, ek birkaç dil desteği gelmesine ve önceki standartlardaki sıkıntılar gidermesine rağmen, Java ile gelen *Remote Method Invocation* (RMI) dağıtık sistemleri

oldukça basitleştirmiş ve *Extensible Markup Language* (XML) ile CORBA'nın büyük oranda terk edilmesine sebep olmuştur.

Distributed Computing Object Model (DCOM), CORBA'ya karşı Microsoft tarafından geliştirilmiş, ilk kez 1993 yılında kullanılmış bir teknolojidir. Microsoft dünyasında başarılı uygulanmasına karşın Microsoft dışındaki sistemlerde kullanımı kısıtlı kalmıştır. Kurumsal çapta büyük uygulamalar (*Enterprise Resource Planning* - ERP), bu dönemde ortaya çıkmış ve genelde Microsoft dışı teknolojiler tarafından tercih edilmiştir. Daha sonra Java'nın *Enterprise JavaBeans* (EJB) platformu DCOM'un alternatifi olacaktır. İnternetin 1990'lı yılların sonunda yaygınlaşması ile firmalar, kendi platformlarını müşterileri ve iştirakleri paylaşmanın faydalarını anlayacaktır.

İnternetin yaygınlaşmasından önce organizasyonlar arası bağlantılar kiralık hatlar (leased lines) üzerinden sağlanmaktaydı. CORBA veya DCOM'un piyasa paylaşımı hedefleri ve İnternet üzerinden kullanımında ciddi oranda network kısıtlamaları olması bu teknolojilerin alternatiflerinin doğmasına yol açmıştır (Davis 2009).

1.1.2 *Simple Object Access Protocol (SOAP)*

Simple Object Access Protocol (SOAP), 2000 yılında ortaya çıkmıştır. SOAP, genel anlamda CORBA ve DCOM'a bir RPC alternatifi olarak düşünülmektedir. Bunun sebebi RPC'lerin dağıtık sistemlerin baskın modeli olması ve SOAP ile aynı kapasitede kullanılmaya başlanmasıdır. Aslında SOAP, RPC'nin daha çok platformun konuşmasını sağlayan, XML tabanlı gelişmiş halidir. Günümüzde ise *Service-Oriented Architecture*, Internet'in ve özellikle *Hyper-Text Transfer Protocol* (HTTP)'ün yaygın kullanımı ile başlamıştır.

SOAP uzak yordam çağırma yani *request / response* mantığı ile çalışır. İstemci program geçerli bir SOAP XML belgesinin oluşturur ve HTTP veya *Simple Mail Transfer Protocol* (SMTP) üzerinden sunucuya göndererek *request*'i oluşturur. Sunucu, SOAP *request*'i alır ayrıştırır ve doğrular, verilen parametrelerle herhangi bir yöntemini

çağırır, geçerli bir SOAP XML *response*'u oluşturur ve HTTP veya SMTP üzerinden geri gönderir.

1.1.3 *Service-Oriented Architecture*

SOA bir takım teknolojileri tanımlayan bir kavram olmaktan ziyade, teknolojilerden bağımsız, tamamıyla işe yönelmiş bir mimari modeldir.

SOA yaklaşımında izlenmesi gereken süreci özetlemek gerekirse;

i. Bu mimari modelde iş süreçleri mevcut mimari yapı dâhilinde ele alınır ve büyük resim çizilir, bunun sonucunda süreç uygulaması veya senaryolar üretilir.
ii. Bu süreçlerde gerekecek servisler ve bu servislerin ihtiyaçlar doğrultusunda etkileşimleri tasarlanır.
iii. Bu servisleri kategoriler altında düşünmek mümkündür.(Finans Servisleri, Kimlik doğrulama servisleri, Destek Servisleri vb.) Bu servislere ve servislerin kullanımına ilişkin kısıtlamaları (*Service Level Agreement* - SLA) belirten politikaların hazırlanması ve test senaryolarının oluşturulmasından sonra servislerin teknik implementasyonu yapılır.

1.1.4 *Application Programming Interface (API)*

Application Programming Interface (API), bir programın işlevselliğini başka programların da kullanabilmesini sağlayan fonksiyonlara verilen addır. Örneğin; bir programın .dll uzantılı dosyası bir API'dır, bu dll dosyası kullanılarak o programın fonksiyonları çağırılabilinir. API'lerin *public* olması güvenlik açısından büyük bir dezavantajdır. Diğer bir dezavantaj ise farklı uygulama platformları için ayrı ayrı API'ler hazırlama zorunluluğudur (Aydın ve Kaya 2008).

1.1.5 Web Servis

Web servisler, internet üzerinden kullanılabilen platform bağımsız bir programın işlevselliğini başka programların da kullanabilmesini sağlayan fonksiyonlardır (Aydın ve Kaya 2008). Web servisler için İnternet çağının API'leridir denebilir. Tek cümleyle web servis "internet üzerinden kullanılabilen platform bağımsız API" diye nitelendirilebilir.

Web servisler üzerinden haberleşecek sistemlerin birbirlerinin yapılarından haberdar olması veya platformların birbirleri ile uyumlu olması gerekmez. Örneğin; Java programlama dili ile geliştirilmiş ve UNIX işletim sistemi üzerinde çalışan bir uygulama ile, .NET ile geliştirilmiş ve Windows işletim sistemi üzerinde çalışan bir uygulama XML iletişim standartları aracılığıyla iletişim kurabilir.

Web servisler ile bir uygulamanın fonksiyonları, çok geniş bir kullanıcı kitlesine sunulabilir. Örneğin; Meteoroloji Genel Müdürlüğü'nün hava tahminlerini haber sitelerinin, ajansların vs. alması. HTTP altyapısı üzerinden çalıştırıldığı için, sistemleri dışarıya güvenlikten ödün vermeden açma imkanı vardır, bu da web servislerin API'lere olan üstünlüğüdür. Platform bağımsız olduklarından web servis kullanılarak geliştirilen uygulamaların maliyeti daha düşüktür.

Bir web servisin çağırılmasındaki temel adımlar şunlardır (İmre 2012):

i. Web servisi çağıran SOAP istemci (*client*) uygulaması, *Universal Description Discovery and Integration* (UDDI) biriminden web servisi bulur. (UDDI, Kurum ve web servisleri bilgilerini saklayan ve yayınlayan sunucudur.)
ii. *Client* bir SOAP istek (*request*) mesajı hazırlar. SOAP *request* mesajı bir XML belgesidir.
iii. *Client* SOAP *request* mesajını web *server* ya da uygulama sunucusunda çalışan SOAP *request* dinleyicisine (*listener*) gönderir. *Request listener* gelen *request*'lere cevap veren Java *Server Pages* (JSP), *Active Servled Pages* (ASP), *Common Gateway Interface* (CGI) veya *Internet Server Application Programming Interface* (ISAPI) gibi sunucu (*server*) programlardır.

iv. SOAP *server* gelen SOAP *request* mesajını çözümler (*parse* eder) ve gerekli parametreleri göndererek belirtilen nesnenin belirtilen metodunu çağırır.

v. Çağırılan nesnedeki metot çalışır ve sonuçları (*response*'u) SOAP *server*'ına gönderir. SOAP *server*'ı gelen *response*'u SOAP mesajı formatında biçimlendirerek *client*'a gönderir.

vi. *Client* gelen SOAP mesajının içindeki bilgileri alarak istekte bulunan programa gönderir.

1.1.6 SoapUI

SoapUI; açık kaynak kodlu fonksiyonel bir test aracıdır. Genellikle web servis testlerinde kullanılan SoapUI; SOAP, *Representational State Transfer* (REST), HTTP, Java *Message Service* (JMS) ve Java-based *Data Access Technology* (JDBC) gibi birçok protokolü de desteklemektedir. Çok hızlı bir şekilde performans testleri oluşturulmasına ve otomatik fonksiyonel testler hazırlamasına olanak sağlar (SoapUI 2014).

1.1.7 COBOL *Copybook*

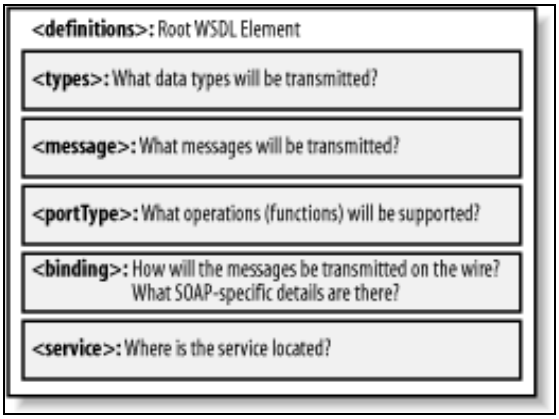
Copybook COBOL programlarının veri yapısını ifade eden kod bölümüdür (IBM [tarih yok]). *Mainframe* ortamında COBOL programları ve *copybook*'ların içeriği büyük harflerle yazılır. *Copybook*'larda seviyeler mevcuttur. Bu seviyeler sayılar ile ifade edilir. Sayı büyüdükçe veri tipinin alt seviyelerine inilmiş olur. Örneğin 05 seviyesindeki bir alanın altında 10 seviyesi tanımlandıysa, bu 05 seviyesindeki alanın bir bölümünü ifade eder. 88 seviyesi ise *enumeretion* veri tipi olarak düşünülebilir. Bir alan için 88 seviyesinde bir tanımlama varsa bu o alanın alabileceği sabit değerleri temsil eder. Veri tipi olarak temelde X yani *string* ve 9 yani nümerik veri tipleri mevcuttur. PIC X(3) 3 karakter uzunluğunda bir *text*'i ifade eder. PIC S9(9) ise 9 basamaklı bir sayıyı ifade eder, S harfi ise işaret (*sign*) anlamındadır. Başında yıldız karakteri (*) olan satır yorum (*comment*) satırı anlamına gelir.

Örnek COBOL Copybook için EK 1'e bakınız. Örnek COBOL Copybook'u aşağıda belirtilen kaynaktan alınmıştır. (Erişim tarihi: 09.10.2013)
https://code.google.com/p/cobolunit/source/browse/trunk/cobolunit+--username +hvaujour/COBOLUnit/CPY/CBUC0002.cpy?r=178

1.1.8 *Web Service Definition Language* (WSDL)

i. *Web* servislerin ara yüzünü belirten bir standarttır
ii. XML dokümanıdır
iii. Bir WSDL dosyasından *web service client* üretilir
    a. Java objelerinin XML formatına (veya tersine) dönüştürülmesi ve kullanılması.
iv. WSDL dokümanı ana kısımları Şekil 1.1'de belirtilmiştir.:
    a. *Types*: Kullanılan data tipleri detayını içerir
    b. *Message*: Operasyon input/output mesajlarını içerir
    c. *Port type*: Operasyon tanımlarını içerir
    d. *Binding*: Protokol detayını içerir (SOAP 1.1 vs.)
    e. *Service*: Servisin ayakta olduğu adres detayını içerir

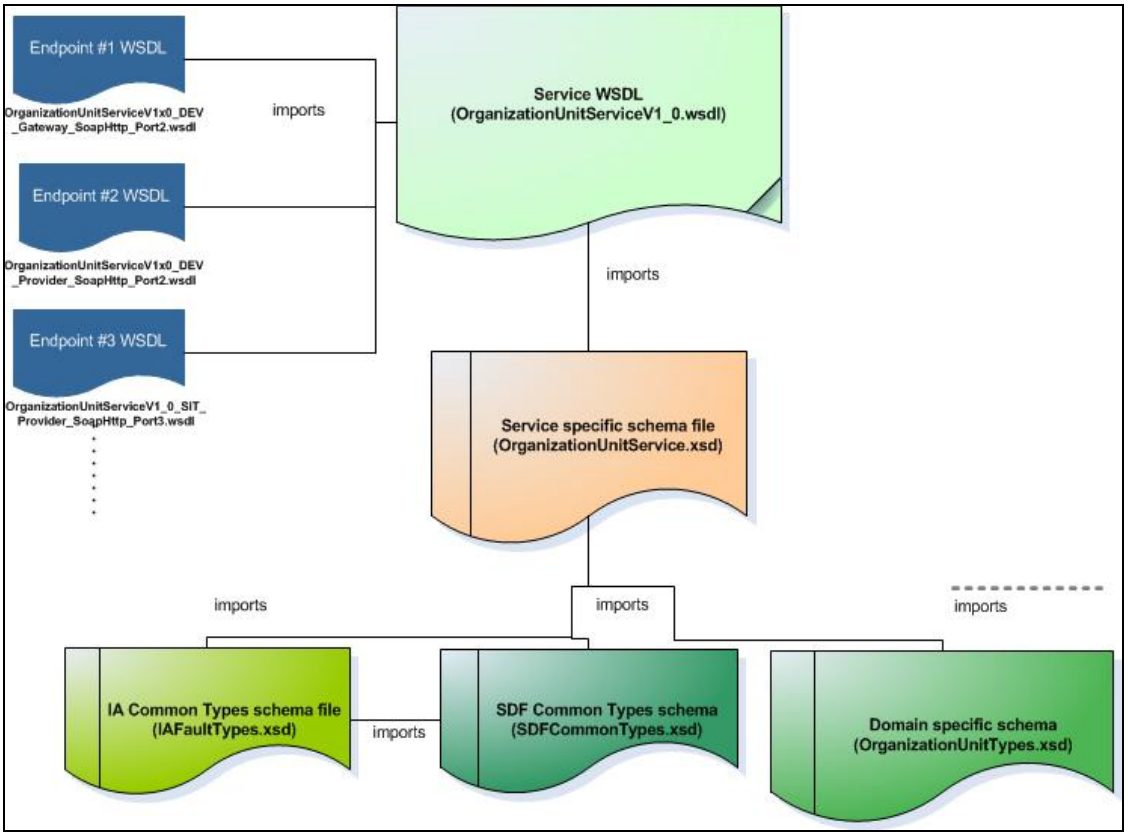
Şekil 1.1: WSDL fiziksel dosya yapısı



Kaynak: http://oreilly.com/catalog/webservess/chapter/ch06.html
Erişim tarihi: 05.10.2013

v. Parçalı WSDL temelleri için Şekil 1.2 incelenebilir.

Şekil 1.2: Parçalı WSDL fiziksel dosya yapısı



Kaynak: http://oreilly.com/catalog/webservess/chapter/ch06.html
Erişim tarihi: 09.10.2013

Parçalı WSDL'a örnek olarak tez konusu kapsamında geliştirilen *Web service* test ve entegrasyon aracı ile entegrasyon uygulaması yapılan geoipservice'in WSDL'ı EK 2'de verilmiştir.

1.2 SOAP ile ENTEGRASYON SERVİSLERİNİN GELİŞTİRİLMESİ KONUSUNUN SEÇİMİNDE ETKİLİ OLAN FAKTÖRLER

SOA modelinin farklı teknolojileri entegre etmedeki sağladığı avantajlardan dolayı tez konusu bu yönde seçilmiştir.

Bu avantajlar arasında;

i. Servislerin tekrar kullanılabilirliğinin sağlanması,
ii. Farklı sistemler üzerinde geliştirme, test ve entegrasyon işlemlerinde zaman kazanılması,

iii. Farklı platformlarda geliştirilen uygulamalarda aynı altyapının kullanabilinmesi,
iv. Uygulamaların farklı katmanlarını farklı *server*'larda kullanılabilmesiyle, performans artışı sağlanması,
v. Servislerin birbirlerine bağlılığın sınıflar yerine şemalar aracılığı ile sağlanarak bu bağlılığının azaltılması ve bu sayede sistemde oluşabilecek darboğazları da engellenmesi sayılabilir.

1.3 SOAP ile ENTEGRASYON SERVİSLERİNİN GELİŞTİRİLMESİNİN BİLİME YAPACAĞI KATKILAR VE ÇALIŞMANIN AMACI

SOA modelini kullanan entegrasyon araçları, genellikle büyük ve farklı teknoloji altyapıları olan sistemler tarafından kullandığı için akademik ortamlarda bu araçların simülasyonunun ve pratik uygulamalarının yapılması çoğunlukla mümkün olmamaktadır. Öte yandan, mevcut entegrasyon araçlarının yüksek maliyetleri de entegrasyon araçlarının kullanımı ve eğitiminin yaygınlaşmasını engellemektedir. Ayrıca, küçük işletmeler için basit bir entegrasyon yazılımının geliştirilmesi ihtiyacı da var olan ürünlerin yüksek maliyetleri sebebiyle ortaya çıkmaktadır. Tez konusu kapsamında, *light-weight* SOA yaklaşımıyla geliştirilmesi hedeflenen kullanıcı dostu yazılım, küçük ölçekli işletmelerin yukarıda bahsedilen ihtiyacını karşılamaya yönelik olmasından başka, pratik olarak entegrasyon uygulamalarının laboratuar ortamında yapılmasını sağlayarak, SOA modelinin daha iyi anlaşılması açısından da faydalı olacaktır.

2. LİTERATÜR TARAMASI

2.1 İLGİLİ ÇALIŞMALAR

Geleneksel hizmet odaklı mimarilerdeki (SOA) yaygın olan bazı zorlukları araştırarak, bu zorlukların basitleştirilmiş SOA olarak adlandırılabilecek daha ölçeklenebilir, birlikte çalışabilir ve çevik alternatifleri kullanarak üstesinden gelmeyi ele alan bir çalışma Rodriguez ve Demsak (2006) tarafından yürütülmüştür.

Bir SOA projesinin başarılı olabilmesi için teknik ve temel prensipler öneren Erl (2005), Servis Odaklı Mimari yönetimi ile çeşitli seviyelerdeki SOA kararlarının doğru yönetilmesini sağlamıştır.

Erl'ün çalışmalarına benzer olarak, O'Brien (2009) çalışmasında başarılı bir SOA projesi ortaya konabilmesi için bazı teknik ve temel prensipleri ortaya çıkarmıştır. Bunları Servis Odaklı Mimari Yönetimi kavramı altında toplamıştır. Avustralya Yazılım Mühendisliği Konferansında sunduğu çalışmasında SOA projelerinde kapsam, maliyet, efor tahmini oluşturmaya çalışmıştır. Bu tahmini oluştururken teknik, sosyal, kültürel ve kurumsal açıların yanında organizasyonun tecrübesi ile birlikte olgunluk seviyesini de inceleyerek, birbirinden farklı SOA projelerinin tiplerini göz önüne almıştır.

Günümüzde dağıtık sistemlerin nesneye dayalı olarak modellenmesinde en çok kullanılan standartlardan olan CORBA ve Java RMI farklı bakış açılarından karşılaştıran Türksever ve Erdur (2000) performans, dağıtık programlama ve sistem düzeyinde sunulan servisleri ele almışlardır.

Daha önce yaptıkları çalışmaya ek olarak Türksever ve Erdur (2003) yeniden kullanılabilir yazılım bileşenlerine web üzerinden erişim için CORBA temelli bir mimari ortaya koymuşlardır. Daha sonra aynı ortam için Java 2 platformu teknolojileri

kullanımına dayanan alternatif bir mimari verilmiş ve önerilen CORBA temelli mimari ile karşılaştırılmıştır.

Arsanjani (2004) çalışmasında SOA metodolojisinin önemli noktalarına değinmiş, ideal SOA ortamı oluşturulması için gerekli temel analiz ve tasarım faaliyetleri sunmuştur. Ayrıca bu çalışmada, SOA kavramsal meta-modeli *Unified Modelling Language* (UML) diyagramları şeklinde oluşturulmuştur. Mimari stil ve prensipler üzerinde durulmuş, bir SOA şablonu oluşturulmaya çalışılmıştır. SOA kavramı katmanlar seklinde modellenerek bunların tanımlanması için gerekli faaliyetler belirlenmiştir. Son olarak servis odaklı modelleme ve mimarisini meydana getiren bir süreç oluşturulmuştur.

Brown (2008) çalışmasında servis odaklı tasarım ve geliştirme için deneysel bir metodoloji tanımlamıştır. Tanımlanan metodoloji, tasarım ve geliştirme temellerini anlatır. Bu çalışmada web servisleri ve iş süreçleri paralel uygulanır. Ayrıca Brown metodolojinin dayandığı referans modellere vurgu yaparak bir kaç servis senaryosunu göz önüne alır. Servis tasarımı sırasında servislerin fonksiyonel gereksinimleri yanında, güvenlik, haberleşme özellikleri ve politikaları gibi genel konuları da kapsamaktadır.

Krogdahl, Luef ve Steindl (2005), SOA'nın günümüz hızlı değişen ve gelişen iş gereksinimlerini karşılayan yönü ele alındığında, çevik yazılım geliştirmenin (*agile software development*) en uygun yaklaşım olduğunu öne sürmüşlerdir. Çevik yazılım geliştirmenin temelleri incelenmiş, SOA tabanlı sistemlere uygunluğu irdelenmiştir. Uygulama içi servisler için çevik yazılım geliştirme yaklaşımını çalışmışlardır.

SOA kavramının web servislerin kullanılmaya başlaması ile her geçen gün öneminin arttığından bahseden Erl (2006) çalışmasında SOA'nın gelişimsel sürecini de aktarmıştır.

Özellikle büyük ölçekli bir SOA projesinin System Dynamics modelleme tekniği ile daha başarılı olduğunu savunan bir çalışma da Jeng ve An (2007) tarafından ortaya konmuştur. Çalışmalarında SOA uygulamalarının bu metodolojiden faydalanmasını hedeflemişlerdir. System Dynamics modellemeye dayanan birçok SOA senaryo

analizleri de yapılmıştır. Sonuç olarak SOA'nın iki önemli özelliği, hızlı geliştirme ve farklı çözüm sistemleri ve geliştirmelerde yüksek derecede yeniden-kullanılabilirlik olarak belirlenmiştir.

2.2 TEZİN KONUSUNA İLİŞKİN SAPTANAN OLASI SORUNLAR

Çok katmanlı bir mimari yapıda çalışma yapılacak olması tez konusunca geliştirilecek yazılımın karmaşıklığını arttırmaktadır. Yazılımın geliştirmesi ve test süresince alınacak hataların izlenmesi çok katmanlı bir ortamda güçleşmektedir.

Günümüzde SOA kullanan teknolojilerin yönetiminin kolaylaştırılması bir ihtiyaç haline gelmiştir. Bu da basitleştirilmiş (*lightweight*) SOA aracı ihtiyacını doğurmaktadır. Bu soruna yönelik olarak tez boyunca kullanıcı dostu bir ara yüze sahip, karmaşık olmayan bir yazılım üretilmesine ve bu yazılımın insan - bilgisayar etkileşimine dikkat edilmelidir.

3. VERİ VE YÖNTEM

3.1 *SERVICE ORIENTED ARCHITECTURE* – HİZMET ODAKLI MİMARİ (SOA)

SOA bir bilgi işlem stratejisi olup, kurumsal yazılımlar içindeki farklı fonksiyonları karşılıklı çalışabilecek, standartlara dayanan, iş ihtiyaçlarını karşılamak üzere kolaylıkla tekrar kullanılabilen ve entegre edilebilen hizmetler haline sokmaktır. Hizmetler, aslında mevcutta var olan fonksiyonların ya da yeni oluşturulan fonksiyonların, belli prensipler göz önünde bulundurularak hizmet halinde sunulmasıdır (Özdemir, 2011)

SOA bir takım teknolojileri tanımlayan bir kavram olmaktan ziyade, teknolojilerden bağımsız, tamamıyla işe yönelmiş bir mimari modeldir (Akyüz, 2007).

SOA yaklaşımında izlenmesi gereken süreci özetlemek gerekirse;

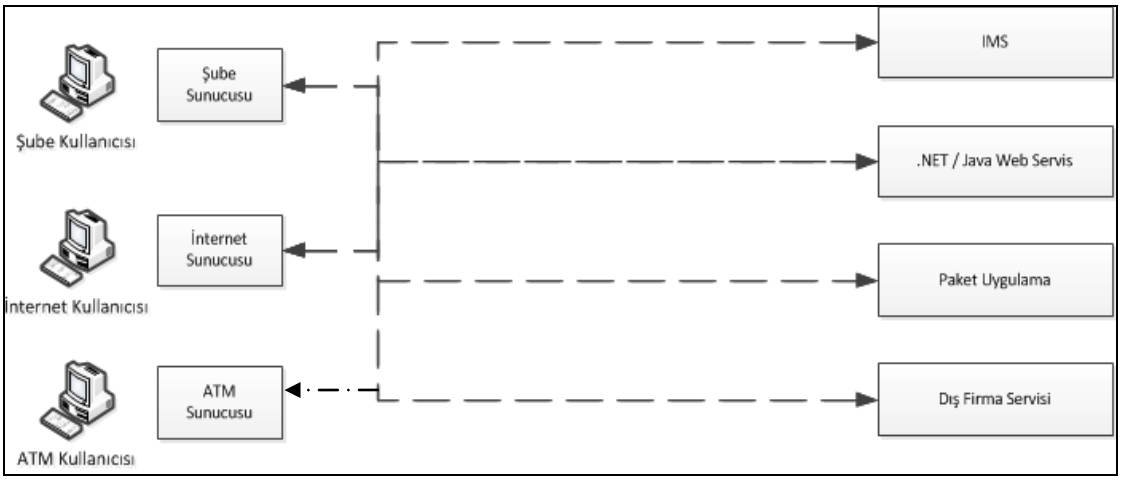
i. Bu mimari modelde iş süreçleri mevcut mimari yapı dahilinde ele alınır ve büyük resim çizilir, bunun sonucunda süreç uygulaması veya senaryolar üretilir.
ii. Bu süreçlerde gerekecek servisler ve bu servislerin ihtiyaçlar doğrultusunda etkileşimleri tasarlanır.
iii. Bu servisleri kategoriler altında düşünmek mümkündür.(Finans Servisleri, Kimlik doğrulama servisleri, Destek Servisleri vb.) Bu servislere ve servislerin kullanımına ilişkin kısıtlamaları belirten politikaların hazırlanması ve test senaryolarının oluşturulmasından sonra servislerin teknik implementasyonu yapılır (Akyüz, 2007).

Kurumsal yazılımlar içindeki farklı fonksiyonları, entegre edilebilen hizmetler haline sokma ihtiyacını bir örnek ile açıklayacak olursak;

Bir bankanın alternatif dağıtım kanallarının önyüzünden çağrılan öyle bir işlem olsun ki; bu işlem hem *mainframe* veritabanlarında bir değişikliğe, hem açık sistem veritabanlarında bir değişikliğe, hem de dış firmadan bir servis çağrılmasına ihtiyaç

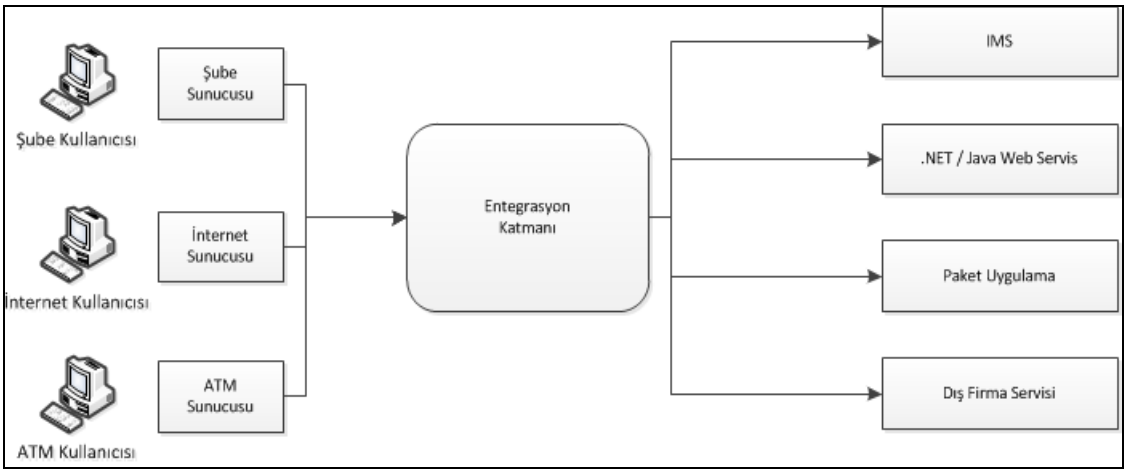
duysun. SOA metodolojisini gerçekleştiren bir entegrasyon katmanı kullanılmadığında her önyüz, *backend*'lerde yapılacak işlerin mantığını ve erişim mekanizmalarını ayrı ayrı kodlamak zorundadır bu durum Şekil 3.1'de de gösterilmiştir.

Şekil 3.1: Entegrasyon katmanı olmadan haberleşme



Önyüz uygulamaları ile *backend* uygulamalar arasına eklenen bir entegrasyon ara katmanı sayesinde ortak işler bu katmanda yapılarak önyüzlerde kod *duplikasyonu* önlenmiş olur. Bakınız Şekil 3.2.

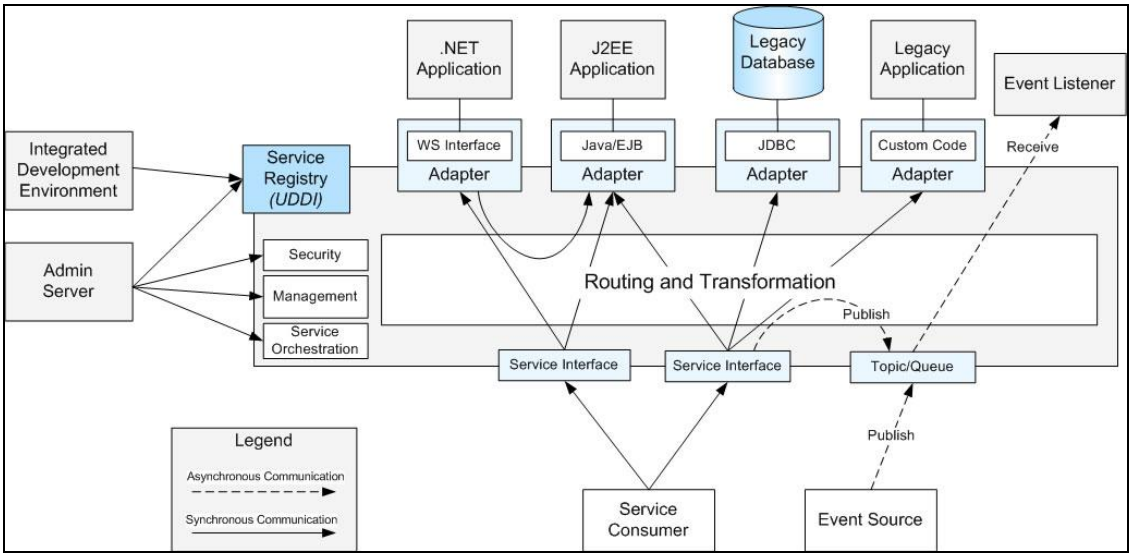
Şekil 3.2: SOA *implementasyonu* ile entegrasyon katmanı



3.2 *ENTERPRISE SERVICE BUS* – KURUMSAL HİZMET VERİYOLU (ESB*)*

Enterprise Service Bus (ESB), SOA uygulanması için bir şablon (*pattern*) görevini gören, çeşitli altyapısal hizmetleri sunan, heterojen ortamlarda dahi çalışabilen, böylelikle entegrasyonu kolaylaştıran bir altyapı mimarisidir. Şekil 3.3'te bir ESB şablonu örneği verilmiştir. Bu altyapı, yeni bir ürün kategorisi olarak ortaya çıkmakta veya mevcut ürünlerin ESB özellikleri ön plana çıkartılarak pazarlanmaktadır. ESB ile birlikte kurumlar mevcutta var olan uygulamaları arasındaki entegrasyon yapılarını yeniden şekillendirmektedir. Buna göre "*Legacy*" adını verdiğimiz, kurum içerisindeki eski uygulamalara yatırım kaybı olarak bakılmamaktadır. Bu uygulamalardan modüler servisler üretilerek bir yandan uygulamalar arasındaki entegrasyon ihtiyaçları giderilirken diğer yandan servis odaklı yeni bir hizmet mantığına geçilmektedir. "*Peer to Peer*" olarak bilinen, her uygulamanın bir diğer uygulama ile entegrasyonu yöntemi terk edilirken, bunun yerine bütün uygulamaların üzerinde yeni bir katman olarak "ESB" konumlandırılmaktadır (Özdemir, 2011).

Şekil 3.3: *Enterprise Service Bus* şablonu



Kaynak: https://enterprisearchitecture.nih.gov/Pages/EnterpriseServiceBusPattern.aspx
Erişim tarihi: 10.10.2013

Özetle ESB'nin Görevleri:

i. ESB'ler dağıtık servisleri birleştirmeye, çalıştırmaya ve yönetmeye yararlar.
ii. ESB'ler dağıtık çalışan, standartlara dayanan entegrasyon ve kurumsal bir altyapı sunar.
iii. Geleneksel sistemler sıkı semantik bağlarla kurulmuştur. Eğer bu semantik zincirinin bir halkası kırılırsa bütün zincir kırılmış olur. Fakat ESB'ler semantik dönüşümlerini mümkün kılarlar.
iv. ESB fiziksel bağlantıları soyutlaştırır ve böylelikle konum bağımsızlığı kazanılır.
v. ESB içinde iş süreçlerini ya da iş mantığını uygulayabilecek araçlar bulunabilir.
vi. ESB, içeriğe göre akıllı mesaj yönlendirmesi ya da filtreleme bulundurabilir.
vii. ESB'ler farklı protokollerle mesajlaşmayı destekler. Bir protokolden diğerine dönüşümü kolayca sağlarlar.
viii. Sadece SOAP mesajları değil, farklı mesaj tipleriyle de çalışabilir, örneğin, REST, XML.
ix. Servislere "güvenlik hizmetleri" sağlayarak onları koruyabilir.
x. Hizmet Seviyesi Anlaşmalarına (Service Level Agreement - SLA) dayanarak servisleri gözlemleyebilir ve alarmlar oluşturabilir (Özdemir, 2011).

Özetle ESB nin Faydaları:

i. Kurumsal Süreç otomasyonu için altyapının oluşturulması,
ii. Kurumda var olan uygulamaların kolayca servis haline dönüştürülmesi,
iii. Servislerin kolaylıkla farklı protokollere açılımının sağlanması (Protokoller arası geçiş) (Özdemir, 2011).

3.3 *ENTERPRISE SERVICE BUS* ARAÇLARI

SOA modelini kullanan bu platformlar aşağıdaki şekilde sıralanabilir:

i. TIBCO Platformu
ii. IBM WebSphere Enterprise Service Bus
iii. Microsoft BizTalk Server
iv. Oracle Enterprice Service Bus

3.3.1 TIBCO Platformu Üzerinde Entegrasyon Servisi Geliştirmek

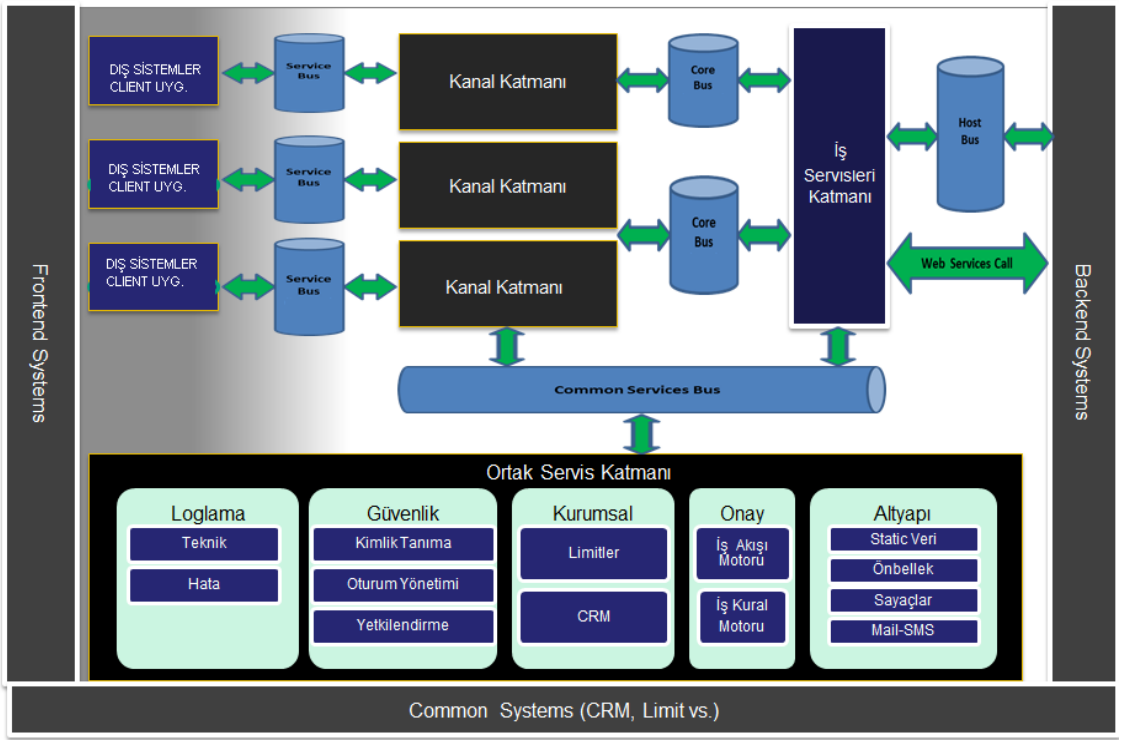
3.3.1.1 TIBCO platformuna genel bakış

3.3.1.1.1 *Entegrasyon ortamı ve mimari yapı*

Entegrasyon katmanı TIBCO ürünü olan *Enterprise Message Server* (EMS) ve *Business Works* (BW)'den oluşmaktadır. Entegrasyon katmanı mimarisi Şekil 3.4'de verilen diyagramla özetlenebilir.

i. EMS: *Queue* ve *Topic* tanımlanan, mesajlaşma *Server'*ıdır.
ii. BW: Tüm orkestrasyonların yapıldığı ortamdır. EMS ile konuşabildiği gibi, HTTP protokollerini de desteklemektedir. *Java Development Kit* (JDK) 1.5 ortamında çalışmaktadır.

Şekil 3.4: Entegrasyon katmanı mimarisi özeti



Mimariyi oluşturan parçalara ait bilgilere şu şekildedir;

Enterprise Message Server (EMS) :

i. *Service Bus* (EMS): *Client*'ın bildiği *interaction* noktasıdır. *Client* operasyon *buffer*'ını (XML), buradaki önceden belirlenmiş *queue*'lara bırakır.

ii. *CommonLogging Bus* (EMS): Asenkron yapıda çalışır. Loglama amaçlı tüm mesajlar buradaki *queue*'larda biriktirilir.

iii. *CommonBusiness Bus* (EMS): Kanallara arası ortak iş mantıkları buradan sunulur. Ortak *Authentication, Authorization, Approval* vb.

iv. *CoreService Bus* (EMS): Kanal BW *engine*'ler ile *Core* BW *engine*'ler arasındaki mesajlaşmayı sağlayan *queue*'ların yer aldığı *bus*'dır.

v. *Host Bus* (EMS): Sadece *Mainframe* ile iletişimde kullanılır. *Mainframe*'deki *Substation* ürünü, bu EMS'i dinler.

BW (*Business Works*):

i. *Channel* Orkestrasyonu BW: Kanal akışları işletilir. *Service Bus'*dan mesajı alır ve ilgili akışı işletir.
ii. *Common Logging* BW: *Commonlogging Bus'*dan mesajı alır ve ilgili tablolara *insert* eder. Asenkron çalışır.
iii. *Common Business* BW: Ortak iş akışları buradan yönlendirilir. Örneğin Business Process Management (BPM) onay istekleri.
iv. *Core Services* BW: *Core* servislerin *interface'*lerinin ve akışlarının tanımlı olduğu ortamdır. *Service Composition* bu ortamda yapılır.

3.3.1.1.2 *Entegrasyon katmanı genel standartları*

Tüm EMS' lerde (*Service Bus*, *Common Business Bus*, *Core Bus* ve *Host Bus*) aşağıdaki standartlar geçerlidir.

i. Tüm mesajlar *Non-Persistent* tanımlıdır
ii. *Pooling Interval'*ler 10 saniye olarak belirlenmiştir.
iii. *Bus*'lardaki Mesajlaşma *Request/ Reply* şeklindedir. İşlemin cevabı alınmaktadır.
iv. *Request* gönderimi daha önceden belirlenmiş *queue*'lara yapılmaktadır. *Response queue*'lar ise *engine* isimlerine göre (her *engine* için bir tane olacak şekilde) belirlenmiştir.
v. *Client'*tan gelen istekler *Service Bus'*tan karşılanmaktadır. Burada *Response queue*'lar *Temprory queue'*lardır. Her *Presentation* sunucusunun kendine ait *Temprory queue*'su bulunmaktadır.
vi. *Servis Bus'*ta *timeout* süreleri *connector* aracılığı ile ems.xml dosyasındaki *Pool* tanımlarından belirlenmektedir. Ems.xml dosyasında uzunluklarına göre seviye seviye kategorize edilmiş süreler *set* edilmektedir. Hangi operasyonun hangi kategoriden çalışacağı bilgisi veritabanındaki ilgili tabloda, ilgili alana atanmıştır.

Örnek Ems.xml dosyasının içeriği aşağıdaki gibidir:

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
<poolCount>2</poolCount>
<connectionPoolControlAttemptInterval>300000</connectionPoolControlAttemptInter
val>
<connectionPoolControlAttemptDelay>1000</connectionPoolControlAttemptDelay>
<poolDataList>
<poolData>
 <poolName>SEVIYE1</poolName>
 <emsURL>tcp://xxx.yy.com.tr:1010 tcp://mcauap2.xxx.com.tr:1011</emsURL>
 <emsUserid>R5AE061C2C31194C</emsUserid>
 <emsPassword>08090F65081D6BF7</emsPassword>
 <connectionCount>3</connectionCount>
 <sessionCount>2</sessionCount>
 <receiveTimeOut>75</receiveTimeOut>
 </poolData>
<poolData>
 <poolName> SEVIYE2</poolName>
 <emsURL>tcp://zzz.yy.com.tr:1010 tcp://zzz.xxx.com.tr:1011</emsURL>
 <emsUserid>R5AE061C2C31194C</emsUserid>
 <emsPassword>08090F65081D6BF7</emsPassword>
 <connectionCount>3</connectionCount>
 <sessionCount>2</sessionCount>
 <receiveTimeOut>135</receiveTimeOut>
 </poolData>
 </poolDataList>
</root>
```

vii. *Core Bus* ve *Host Bus timeout* değerleri dinamik olacak şekilde düzenlenmiştir. *Timeout* değerleri veritabanındaki ilgili tablodaki ilgili alana atanmıştır. Bu tablo statik data olarak kanal ve *core* projelere yüklenmektedir. Statik data olarak tutulan *timeout* değerleri, *CoreBus*'a bağlanmayı sağlayan *Call Core*

Services process'i ve *Host Bus'*a bağlanmayı sağlayan *Call Substation process'i* içindeki *Wait for JMS Message* aktivitesinin *process Timeout input*'una set edilmiştir.

3.3.1.1.3 *Entegrasyon katmanı mesaj yapıları*

Entegrasyon Katmanı mesajlaşma yapısı üzerine kuruludur. Kanallar ve Entegrasyon Katmanı arasında hangi formatta mesajların gidip geleceğini belirleyen .xsd uzantılı dosyalar vardır. Entegrasyon Katmanı kanal sahiplerince servis ihtiyaçlarına göre hazırlanan ve uygulama katmanında tutulan .xsd uzantılı dosyanın aynısı Entegrasyon Katmanı projelerinde de yer alır. Örneğin; CustomerOrder isimli .xsd dosyası örnek oluşturması açısından [http://msdn.microsoft.com/en-us/library/bb675181.aspx] kaynağından alınmıştır ve EK 3'te verilmiştir.

3.3.1.2 TIBCO ile entegrasyon servisi geliştirme prensipleri

Entegrasyon alt yapısı kullanan sistemlerin, ihtiyaç duydukları entegrasyon servisinin *backend* servislerini temel olarak ikiye ayırabiliriz.

i. Standart servisler (*Mainframe*'de çalışacak servisler, Webservisler, vb.)
   Kanal akışına göre alınmak istenen bilgiler ya da yapılması istenen iş, tek bir servis ile sağlandığı durumlarda standart bir servis geliştirmesi yapılır. *Backend* servis *mainframe* programı ise *copybook* detayı, webservis ise web servis WSDL bilgisi ile bu servislerin *core* projeleri için geliştirilme aşamasına gelinir.

ii. Birden fazla *backend* servisin çağrıldığı kompozit servisler
   Kanalın ihtiyaç duyduğu servis *backend* de birden fazla programa bağlanma durumunda ve *mainframe'*de başlatılması istenen *transaction* sayısı üçü geçmiyor ise entegrasyon katmanında kompozit servis tasarlanır.

Recurrent çağrılır gibi iteratif olarak *mainframe*'e gidişi tetikleyen servislerin, hem işlem bütünlüğü sağlamak hem de sistem kaynaklarını verimli kullanmak adına

entegrasyon ihtiyacı olan sistemler tarafından bir programda toplanması ve entegrasyon katmanının bu servisi çağırması doğrudur.

Örneğin, fatura ödeme servisinde önce web servisden borç sorgulaması yapıp ardından fatura ödeme işlemi yapılması entegrasyon katmanında bir kompozit servis olarak tasarlanabilir.

Hesap listesini *input* olarak alan bir servisin, her bir hesap için tek tek mevduat sistemi programından bilgi sorgulaması entegrasyon katmanında *composition* yapılmaması gereken bir örnektir. *Input'*ta kaç tane eleman geleceği bilinmediği için iteratif bir şekilde *transaction* başlatarak *backend* sistem kaynaklarını olumsuz yönde etkileme söz konusu olabilir.

Yeni entegrasyon servisi için aşağıdaki içerikleri ihtiyaç duyulur:

i. *Client* .xsd dosyası,
ii. *Backend* servis bilgileri (*Mainframe* programı ise *copybook* detayı, webservis ise WSDL bilgisi ile),
iii. *Backend* servisine gönderilecek bilgilerin detay *mapping* bilgileri,
iv. Servisin çalışacağı fonksiyon bilgisi (Yeni fonksiyon/Mevcut Fonksiyon):

Yeni fonksiyon için aşağıdaki tanımların yapılması gerekir

i. Fonksiyon tanımı,
ii. Operasyon tanımı,

Mevcut fonksiyon için,

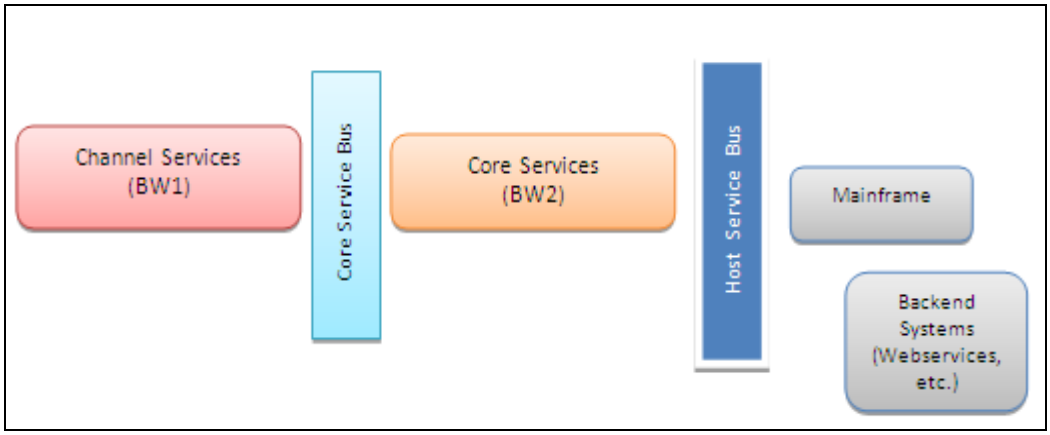
Operasyon tanımı:

Entegrasyon altyapısı, fonksiyon / operasyon tanımlarını baz alarak parametrik bir yapıda çalışır durumdadır. Bu tanımların doğru yapılması, servisin analizini yapan entegrasyon alt yapısı kullanan sistemlerin sorumluluğundadır. Servisin analistinin 4. Maddedeki tanımları *User Acceptance Testing* (UAT) ortamında hazırlamış olması beklenir.

3.3.1.2.1 *Core servis oluşturma*

En temel anlamda TIBCO ile entegrasyon mimarisinde önyüzler *Service Bus* aracılığı ile *Channel Layer'*a mesaj bırakırlar, *Channel Layer'*da kanalın işlem akışına göre ilgili kısımlarda, *Core Bus* aracılığıyla *Core* Servis projelerine erişilir. *Core* projeler *Host Bus* üzerinden *mainframe*'e, SOAP ya da HTTP bağlantısı ile diğer *backend* sistemlere ulaşabilirler. Şekil 3.4'te entegrasyon katmanında TIBCO entegrasyon servislerinin konumlandırılması şematize edilmiştir.

Şekil 3.5: Entegrasyon katmanında TIBCO entegrasyon servislerinin (BW) konumlandırılması



3.3.1.2.2 *Mainframe ile entegre olan core servis oluşturma*

Mainframe (*host*) ile entegre olan bir *core* servisin yapısını şu şekilde özetleyebiliriz:

i. *CopyBook*
ii. XML *Schema Definition* (XSD)
iii. *Implemantasyon* (*process* dosyası)

3.3.1.2.3 *CopyBook hazırlama*

i. Yeni servisin analizini yapan entegrasyon alt yapısı kullanan sistemlerin ya da uzman sistem ekiplerinin *copybook* içeriğini iletmeleri gerekmektedir.
ii. İletilen *copybook*'taki yorum satırları silinir, soldan 01 seviyesine kadar 7 karakter temizlenir, sağdan 8 karakter olarak yer alan numaralar silinir.

iii. *Mainframe* programının veri yapısını içeren CoreServices/!*SUBLOB*!/ Copybooks dizini altında hazırladığımız *copybook* !*HOSTCPYADI*!.cpy isim formatıyla oluşturulur.
iv. Daha sonra TIBCO Designer uygulaması ile açılır, sağ tarafta yer alan *Character Set* açılır kutusunda Cp1254 seçilir. *Apply* ve *Save* edilir.
v. Sağ tarafta sol alttaki Test *Parsing* butonu tıklanır, sonucun başarılı olması beklenir.

3.3.1.2.4 *XSD hazırlama*

i. Yukarıda bahsedilen *copybook*'tan türetilmiş *copybook*'taki alanları .xml uzantılı olarak ifade eden .xsd dosyası CoreServices/!*SUBLOB*!/Schemas dizinine core-\!*SUBLOB*!-!*HOSTPRGADI*!.xsd isim formatıyla oluşturulur,
ii. Bu dosya TIBCO Designer uygulaması ile açılır,
iii. *Toolbar*daki *Validate Resource* (tek *check*) ikonu tıklanır, sonucun başarılı olması beklenir.

3.3.1.2.5 *Transaction ID (Trnxid) tanımlama*

Transaction ID (Trnxid) dört karakterden oluşur *core* servis içinde kullanılır. Benzersiz bir *Trnxid core* servise verilir.

3.3.1.2.6 *Trnxid için substation tanımlanması*

Belirlenen *Trnxid* ID'nin *mainframe* uygulaması ile ilişkilendirilmesi için *substation* tanımı yapılmalıdır.

3.3.1.2.7 *Core servis implemantasyonu oluşturmak ve servisin validate edilmesi*

Bu ana başlık altında bir *mainframe* programı ile entegre oluduğunu varsaydığımız *Core* servisin implemantasyonunu oluşturmak için gerekli adımlar aşağıdaki gibidir:

i. Servis implementasyon *process* dosyası ilgili entegrasyon projesi içinde aşağıdaki dizin ve formatta oluşturulur

ii. /CoreServices/!*SUBLOB*!/Implementation/!*MAINFRAMEPROGRAMADI*!.process

iii. *Process* dosyası TIBCO Designer Uygulaması ile açılır. *Render* aktivitesi tıklanır, *input tabı* açılır. Sol taraftan start/!*HOSTPRGADI*!RequestMsg/ !*HOSTCPYADI*! sağ taraftaki !*HOSTCPYADI*! na bağlanır. *Apply* ve *save* edilir.

iv. Oluşturulan *process* dosyası içinde servis tipi *Main Frame* olacak şekilde belirtilir,

v. Servis *sublob*'u (Örn:Members) ve servis versiyonu ((YYYY-AA):2014-05) girilir,

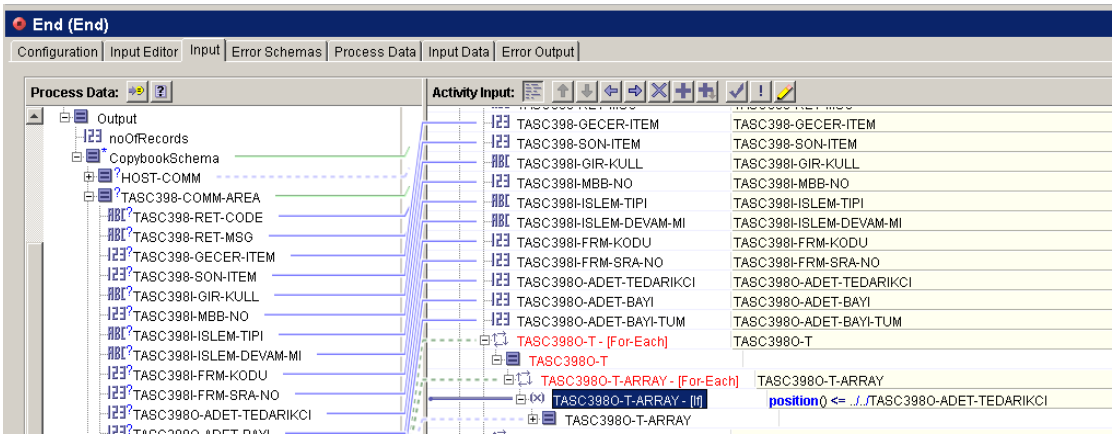
vi. Servisi başlatan *CoreBus Queue*, daha önceden tanımlanan *Trnxid*, *HostBus Queue*, daha önce hazırlanan *copybook* içeriği girilir,

vii. *Mainframe* servisinin döndüğü işlem statüsünü belirten *return code*'un ve işlem statü mesajını belirten *return* mesajının *copybook* içersindeki yeri yazılır.

viii. *End* aktivitesi tıklanır, *input tabı* açılır. sol taraftan Parse-!*HOSTCPYADI*!/Output/CopybookSchema/!*HOSTCPYADI*! sağ taraftaki !*HOSTPRGADI*!ResponseMsg/!*HOSTCPYADI*! na bağlanır. *Apply* ve *save* edilir.

ix. Sol menüden !*HOSTPRGADI*!.process tıklanır *toolbar*daki *Validate Resource* (tek *check*) ikonu tıklanır. Başarılı olması beklenir.

Şekil 3.6: Örnek bir TIBCO Business Works *mapping* ekranı



Şekil 3.6'da örnek bir *mapping* ekranı gösterilmiştir. Burada *Copybook*'taki alanlar ile oluşturulan .xsd'deki alanlar birbiri ile eşleniyor.

3.3.1.3 Web servisleri ile entegre olan *core* servis oluşturma

Web servis ile entegre olan bir *core* servisin yapısını şu şekilde özetleyebiliriz:

i. *Web* servisin veri yapısını içeren WSDL dosyası
ii. Varsa WSDL'ın içerdiği .xsd dosyaları
iii. *Implemantasyon* (*process* dosyası)

3.3.1.3.1 *Core servis implemantasyonu oluşturmak ve servisin validate edilmesi*

Core servis eğer bir web servis ile entegre oluyorsa *Core* servisi oluşturmak için gerekli adımlar aşağıdaki gibidir:

i. Servis *implementasyon process* dosyası ilgili entegrasyon projesi içinde aşağıdaki dizin ve formatta oluşturulur /CoreServices/!*SUBLOB*!/Implementation/ !*WEBSERVİSADI*!.process
ii. Eğer bir *web* servis ile entegrayon sağlanacaksa *Endpoint* URL bilgisi girilir,
iii. *Web* servisin döndüğü işlem statüsünü belirten *return code*'un ve işlem statü mesajını belirten *return* mesajının WSDL içersindeki yeri yazılır.

3.3.1.3.2 *Servisin Core Bus Starter'ının eklenmesi*

Core Servisin çalışması için ilgili veritabanında ilgili tabloya yeni *implementation process*'inin ismi ve *core* projede çalışacağı *stater* kuyruğu eklenmelidir.

3.3.1.4 *Library* oluşturma

Library, *core* projede yapılan geliştirmelerin kanal tarafında kullanılması için oluşturulan bir design elementidir. Kanaldan *core* programa giderken iki projenin hangi *core* desende konuşacağı burada belirtilir.

Başka entegrasyon servisi geliştiren ekiplerce de aynı proje dosyalarının kullanılabilir olması için versiyonlama aracı Team Foundation Server (TFS), ClearCase, Apache Subversion Client (SVN) gibi kullanılmalıdır.

i. Servisin olduğu *container*'ın son hali alınır. (*Update* alınma esnasında TIBCO Designer'da proje açık ise, F5 tuşuna basılarak TIBCO Designer'daki dosyalar *refresh* edilir ya da TIBCO Designer kapatılır ve yeniden açılır.)

ii. Dosyaların *checkout* edilmesi
    a. Çalışılan *core* projedeki *Library* dizini altındaki *libbuilder* dosyası *checkout* edilir. (İlgili_Proje!*ilgilicoreservisprojeadı*!.projlib)
    b. *Shared Libraries* altındaki ilgili *projlib* dosyası *checkout* edilir. (İlgili_Proje!*ilgilicoreservisprojeadı*!.projlib)

iii. *Library* nin *build* edilmesi
    a. TIBCO'da *Core* servisin olduğu ilgili *container* proje açılır
    b. Sol menüde *Library* altındaki İlgili_Proje !*ilgilicoreservisprojeadı*! seçilir.
    c. Sağ alttaki *Resources* tabına açılır. Burada yeni servisin şemasını eklemek için sağdaki dürbün ikonu tıklanır.
    d. Açılan *popup* da CoreServices/!*SUBLOB*!/Schemas/ altındaki yeni eklediğimiz servisin core-\!*SUBLOB*!-!*HOSTPRGADI*!.xsd isimli xsd dosyası seçilip ok denir.
    e. *Apply* butonuna basılır.
    f. Aynı tabda sağ alt köşedeki *BuildLibrary* butonuna basılır. *Save* edilir.

iv. Dosyaların *Checkin* Edilmesi
    a. *Library*'si *build* edilecek ilgili projenin *projlib* dosyası *checkin* edilir. İlgili_Proje!*ilgilicoreservisprojeadı*!.projlib
    b. Yeni bir servis *library*'e eklendi ise aşağıdaki dosya *checkin* edilir, mevcut servisin şeması değiştiği için *library build* ediliyorsa, *libbuilder* dosyası *undo checkout* edilir.

Library'e eklenen dosyalar yeni oluşturulmuş ise versiyon kontrol aracı vasıtasıyla proje arşivine eklenmelidir, mevcut dosyalar değişti ise, *library checkin* edilirken dosyalar da *checkin* edilmelidir.

XSD versiyonunu sileceğimiz durumlarda öncelikle *libbuilder*'dan çıkarılıp, ardından silinmelidir.

3.3.1.5 *Enterprise Archive* (EAR) oluşturulması

Enterprise Archive (EAR), yaptığımız geliştirmeleri UAT, *Pre-production* (PREPROD) ya da *Production* (PROD) gibi herhangi bir ortama *deploy* etmek için oluşturulur. *Archive* dosyasına *core* projelerde *implementation process*ler, kanal tarafında *client* ve *core wrapper* dosyaları eklenir.

i. Versiyon kontrol aracından geliştirme yapılan ortamın tamamı için *update* alınır. (EAR oluşturulmadan önce Shared Library, EAR ve çıktığımız projeyi *update* alınmalıdır.)
ii. *Update* aldıktan sonra projeyi tekrar açmak ya da *refresh* ve *hide* / *show* library yapmak gerekir.
iii. Dosyaların *checkout* edilmesi
    a. Çalışılan core projedeki Deployment dizini altındaki archive dosyası checkout edilir. (İlgili_Proje!*ilgilicoreservisprojeadı*!.archive)
    b. Deployments/EAR altındaki ilgili EAR dosyası checkout edilir.
    c. (İlgili_Proje!*ilgilicoreservisprojeadı*!.ear)
iv. EAR nin build edilmesi
    a. TIBCO'da Core servisin olduğu container açılır (İlgili_Proje vb).
    b. Sol menüde Deployment altıncaki İlgili_Proje tıklanarak projenin archive dosyası seçilir ve ekranın sağ aşağıdaki tab'larından Process seçilir.
    c. Process Archive dosyasında Process tabına geçildikten sonra sağ üst ortadaki dürbüne tıklanarak yeni servisin implementation dosyasını eklemek gerekir. EAR'a yeni bir starter ekleneceği durumda sağ üst soldaki yeşik ok olan dürbüne tıklanarak, starter dosyası eklenir.
    d. Apply seçilip, save butonuna basılır.
    e. Aynı tabda sağ alt köşedeki Build Archive butonuna basılır. Save edilir.
v. Dosyaların Checkin Edilmesi

a. EAR'ların bulunduğu dizinindeki şu ilgili dosya sağ tıklanıp clearcase'den checkin edilir. (İlgili_Proje!*ilgilicoreservisprojeadı*!.ear)
    i. EAR checkin edilirken, servis yeni ise dosyalar versiyonlama aracına eklenmelidir, mevcut servis değişti ise değişen dosyalar da earla birlikte checkin edilmelidir.
b. Yeni bir servis EAR'a eklendi ise aşağıdaki dosya checkin edilir, mevcut servislerde değişiklik olduğu için EAR çıkılıyor ise aşağıdaki dosya undo checkout edilir. (İlgili_Proje!*ilgilicoreservisprojeadı*!.archive)

3.3.1.6 Kanal (Client) servislerinin oluşturulması

3.3.1.6.1 *Fonksiyon operasyon kayıtlarının tanımlanması*

Servis analizi iletildiği anda fonksiyon / operasyon tanımları yapılmamış ise, *database*'den benzer fonksiyon ve / veya operasyon bilgileri kopyalanarak ya da aşağıdaki gibi önyüz ekranlarından bilgi girişi yapılarak tanımlar oluşturulabilir.
Fonksiyon Operasyon tanımları için aşağıdaki bilgiler veritabanında ilgili tabloya girilip kaydedilir. Yapılan Fonksiyon Operasyon tanımlarının *server cache*'lerinde aktif olması için veritabanında *commit* işlemi yapılmalıdır.

3.3.1.6.2 *Client ve core wrapper dosyalarını üretme, .xsd dosyasını kopyalama*

Client wrapper dosyalarının map edilmesi
TIBCO Designer'da operasyonun bulunduğu kanal *container* açılır. İlgili_Kanal_Proje!**!Services\InternetCorporateChannel\!*SUBLOB*!\Wrappers\Client *wrapper* dosyasında, *client* xsd'nin alanlarınından raporlamalara yansıması istenen alan var ise, PostMasterLog aktivitesinde *info* alanına *tag name* verilerek *map*'lenir. Raporlama ihtiyacı yok ise, dosya *validate* edildikten sonra versiyonlama aracına eklenir.

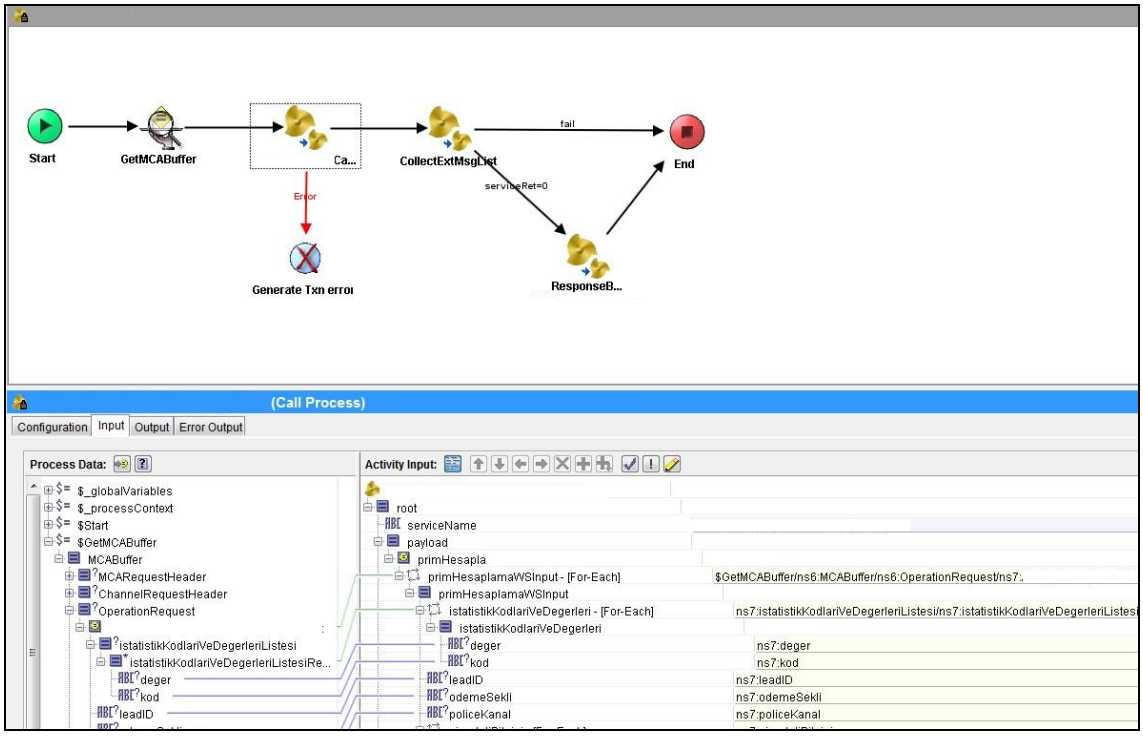
Core wrapper dosyalarının map edilmesi

Core wrapper dosyasından, kanallardan gelen *input* parametrelerinin *backend* servislerine gönderilmesi ve servislerin outputlarının kanallara geri gönderilmesi için *output mappingleri* yapılır. Dosyalar *validate* edildikten versiyonlama aracına eklenir ya da mevcut servis ise *checkin* edilir.

Input Mapping

Core Wrapper açılır,!*CORESERVİSADI*! *input* tabı açılır. Sol taraftaki EntegrasyonBuffer/OperationRequest altındaki *input* alanlar sağ taraftaki !* CORESERVİSADI *!RequestMsg altındaki *input* alanlara tek tek bağlanır. Bakınız Şekil 3.7. Burada gerekli *condition* ya da formatlama ihtiyacı var ise yapılır.(*if*, *when*, *substring* ve *for-each* benzeri fonksiyonlar kullanılanılabilir.). *Apply* ve *save* edilir.

Şekil 3.7: *Core* projeye önyüzden alınan *input*'ların gönderildiği *mapping*



Output Mapping

ResponseBuffer Aktivitesinin *input* tabı açılır. Sol taraftaki CORESERVİSADI *!/!* CORESERVİSADI *!RequestMsg altındaki *output* alanlar sağ taraftaki OperationResponse/!*OperationName*!Response altındaki *output* alanlara tek tek bağlanır. Burada gerekli *condition* ya da formatlama ihtiyacı var ise yapılır. (*if*, *when*, *substring* ve *for-each* benzeri fonksiyonlar kullanılanılabilir). *Apply* ve *save* edilir.

Process validate edilir.

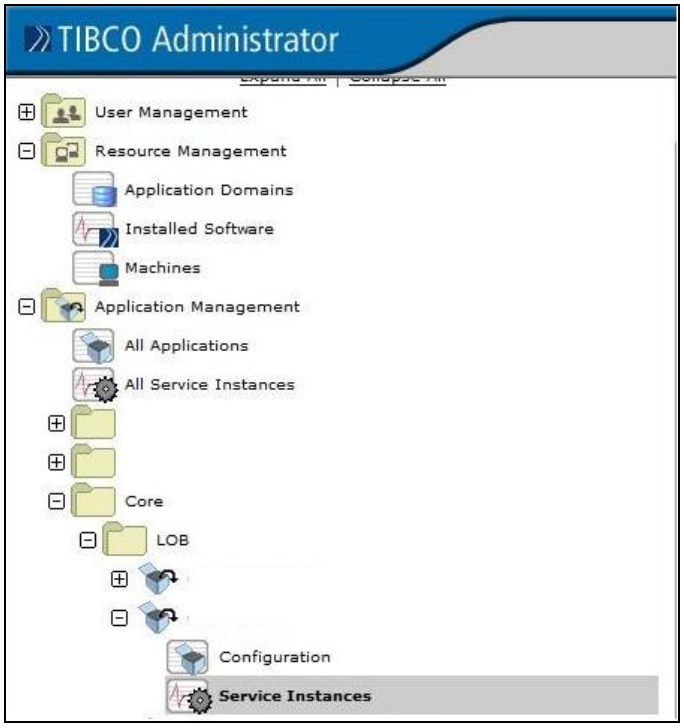
3.3.1.7 User acceptance testing (UAT) deployment

UAT ve Preprod ortamlarındaki *server*lara hazırlanan entegrasyon servislerinin projelerinin *deployment*ları *shell script*ler ile hazırlanmış menüler aracılığı ile entegrasyon geliştiriciler tarafından yapılabilir.

i. Filezilla gibi File Transfer Protocol (FTP) programları aracılığı ile *build* edilen .ear uzantılı proje dosyaları *deployment* yapılacak ortama ait *server*'a kopyalanır.
ii. Putty programı aracılığı ile yine bu *server*'a kullanıcı adı ve şifresi ile bağlanılır.
iii. "sudo -u tibcoadm menu.sh" yazılarak menüye ulaşılır.
iv. Açılan menüden "*Deployment* işlemleri" altından *deployment* başlatılır.

Deployment tamamlandığında TIBCO Administrator panelinden, panel Şekil 3.8'de gösterilmiştir, server üzerine kurulmuş olan projeler ve projelerin içindeki entegrasyon servisleri görüntülenebilir. Ayrıca TIBCO Administrator panelinde hangi entegrasyon servisinin kaç kere çağırıldığı, bu çağrılarının kaçının başarılı yada başarısız olduğu gibi istatistiki bilgilere de ulaşılabilinir. Şekil 3.9'da TIBCO Administrator Panel'i ekran görüntüsünde, entegrasyon katmanı üzerinde çalışan servislerin durumları ile ilgili bilgilerin görünümü verilmiştir.

Şekil 3.8: TIBCO Administrator Panel'i ekran görüntüsü



Şekil 3.9: TIBCO Administrator Panel'i ekran görüntüsü- Entegrasyon katmanıüzerinde çalışan servislerin durumları

Execution Count	Elapsed Time (ms)	CPU Time (ms)	Errors	Status
11	1	1	0	OK
11	0	0	0	OK
11	2	2	0	OK
11	0	0	0	OK
11	793	273	0	OK
7	1188	189	0	OK
7	508	277	0	OK
7	1	1	0	OK
11	54	50	0	OK
11	1	1	0	DEAD
4	1	1	0	OK

3.3.2 IBM *WebSphere Enterprise Service Bus* (ESB) Mimarisi Üzerinde Entegrasyon Servisi Geliştirmek

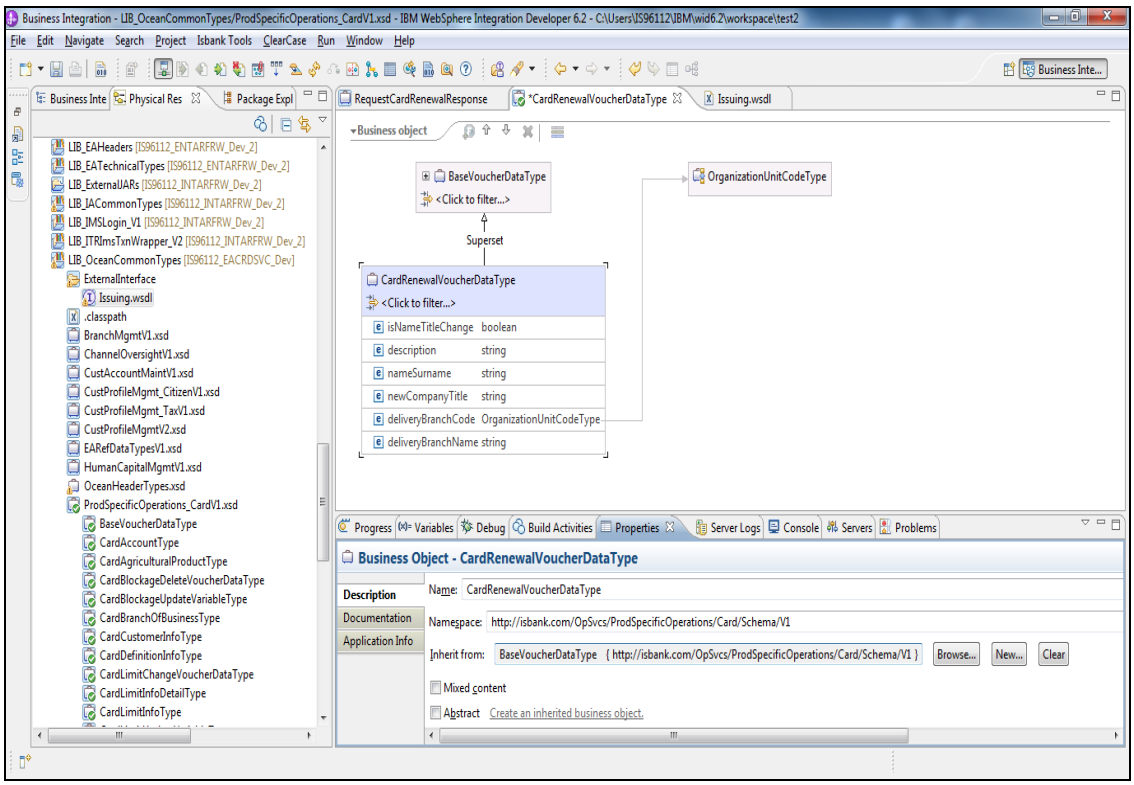
Doğası gereği IBM ESB ürünü üzerinde entegrasyon servisi geliştirme süreci TIBCO Platfotmu ürünü üzerinde entegrasyon servisi geliştirme ile çok benzerdir.

IBM'in ESB servis geliştirme platformu olan WebSphere Integration Developer (WID) ile entegrasyon servislerini geliştirmek için entegre edilecek servislerin türüne göre

WSDL, XSD, *copybook*'ları entegrasyon projesine eklenmelidir. Bu işlemden sonra ilgili servislerin fonksiyonlarını, fonksiyonların *input*/*output*'larını (*request*/*response*) ve *input*/*output*'larda yer alan alanların veri tiplerini görebiliriz. Şekil 3.10'da WID üzerinde isteğe uyarlanmış (*custom*) veri tiplerinin entegrasyon servisinde kullanıldığı gösterilmektedir. Daha sonra hangi servisi entegre etmek istiyorsak, bu servislerin *input* ve *output* alanlarını birbirlerine eşleme (*mapping*) yapılmalıdır. *Mapping* sırasında veri tipi dönüşüm işlemleri gerekebilir.

Servisler arasında yapılan iletişimde entegrasyon servisi aracılığı ile loglama, veri formatlama, veri tipi dönüşümü, hata yakalama gibi birçok genel ihtiyaç karşılanabilir.

Şekil 3.10: WID üzerinde isteğe uyarlanmış (*custom*) veri tipilerinin entegrasyon servislerinde kullanımı



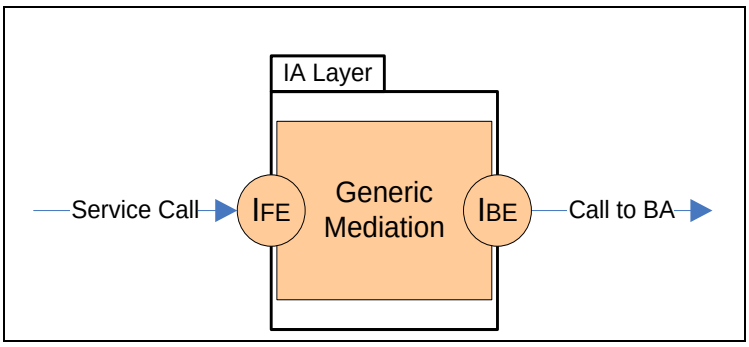
3.3.2.1 Mediation

Mediation, iki farklı sistem arasındaki farklı veri tiplerini eşleştirdiğimiz dönüştürdüğümüz adımdır.

3.3.2.1.1 *Generic mediation*

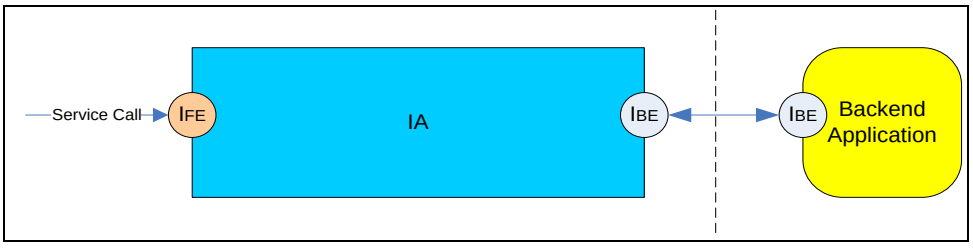
Generic Mediation görevi entegrasyon mimarisinde çeşitli kontrolleri ve denetimleri gerçekleştirmek için kullanılır. Şekil 3.11'de basit bir entegrasyon katmanı gösterilmiştir.

Şekil 3.11: Basit bir entegrasyon uygulaması katmanı



Ana ve en basit entegrasyon mimarisi çözümü, bir *backend* uygulamasının entegrasyon mimarisi üzerinden geçmesidir. *Backend* servisi bir operesyonel fonksiyonaliteyi destekler. *ChannelService, CustomerService, CitizenService, WorkstationService* veya *mainframe* üzerindeki XYZ *transaction*'ı *backend* servislerine örnek olabilir. Bir *backend* servisi *web* servis olmak zorunda değildir. *Backend* uygulamasının entegrasyon mimarisi üzerinden geçirilmesi genel anlamda Şekil 3.12'de özetlenmiştir.

Şekil 3.12: *Backend* uygulamasının entegrasyon mimarisi üzerinden geçirilmesi

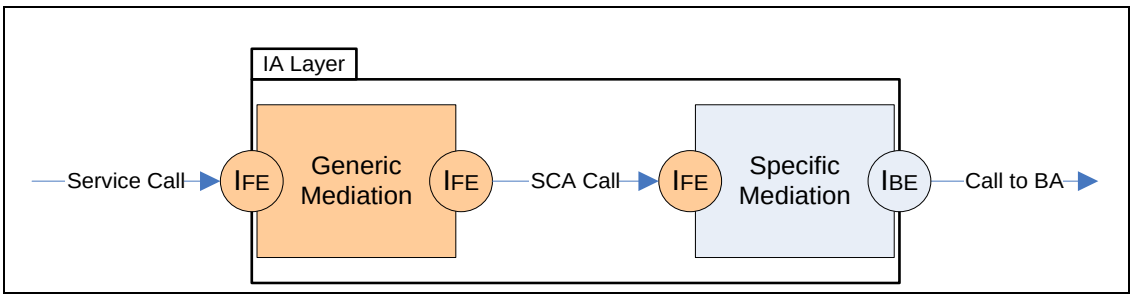


Generic Mediation, Specific Mediation'ın *request*'ini ve *response*'unu karşı taraftaki sisteme aktarır.

3.3.2.1.2 *Specific mediation*

Bu senaryoda *frontend* arabirimi ile *backend* arabirimi birbirinden farklıdır. Geliştirilen entegrasyon uygulamasının iki görevi vardır; bunlardan birincisi genel kontrolleri ve denetimleri gerçekleştirmektir. İkinci görev ise *frontend* ile *backend* arabirimi arasında *interface* dönüşümünü sağlamaktır. Entegrasyon katmanı bu noktada isimleri *generic* ve *specifis mediation* olan iki uygulama içerir. Bakınız Şekil 3.13.

Şekil 3.13: *Frontend* arabirimi ile *backend* arabirimi birbirinden farklı olduğunda entegrasyon katmanınında yer alan uygulamalar



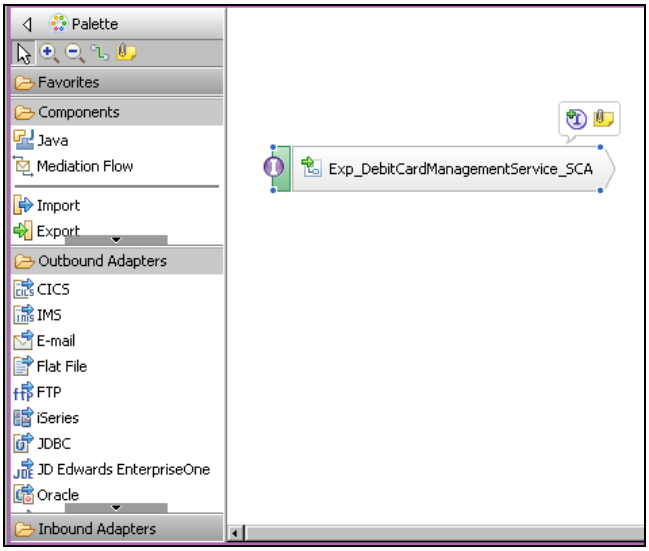
Specific mediation oluşturma adımları şunlarıdır:

i. *New- New Mediation Module* seçilir,
ii. İsimlendirme: MM_Servisadı_VersiyonNo
iii. *Next, Library* olarak oluşturulan wsdl seçilir, esb *Server* seçilir,
iv. *Finish*

Medition module interface atamak için;

i. SCA.module dosyası açılır, ekran Şekil 3.14'te gösterilmiştir.

Şekil 3.14: SCA modül



ii. Oluşan kutucuğa sağ tıklanır, *Add interface* seçilir.

iii. Oluşturulan WSDL *interface* i seçilir, kutunu üzerinde I harfi belirir.

Mediation flow eklemek için;

i. *Mediation flow* SCA.module dosyası üzerine sürüklenir.

Export Oluşturmak için;

i. Oluşan kutucuğa sağ tıklayıp, *Genarate export* seçilir,

ii. *Web service binding* seçilir,

iii. SOAP...using Java API for XML Web Services (JAX)... seçilir,

iv. OK butonuna tıklanır.

İsimlendirme**:** İlk kutucuğa Exp_ServiceName_WS şeklinde yeni isim verilir.

IBM Information Management System (IMS) Outbound Adaptors eklemek için;

i. *Outbound Adaptors*'den IMS *Mediation* alanının üzerine sürüklenir,

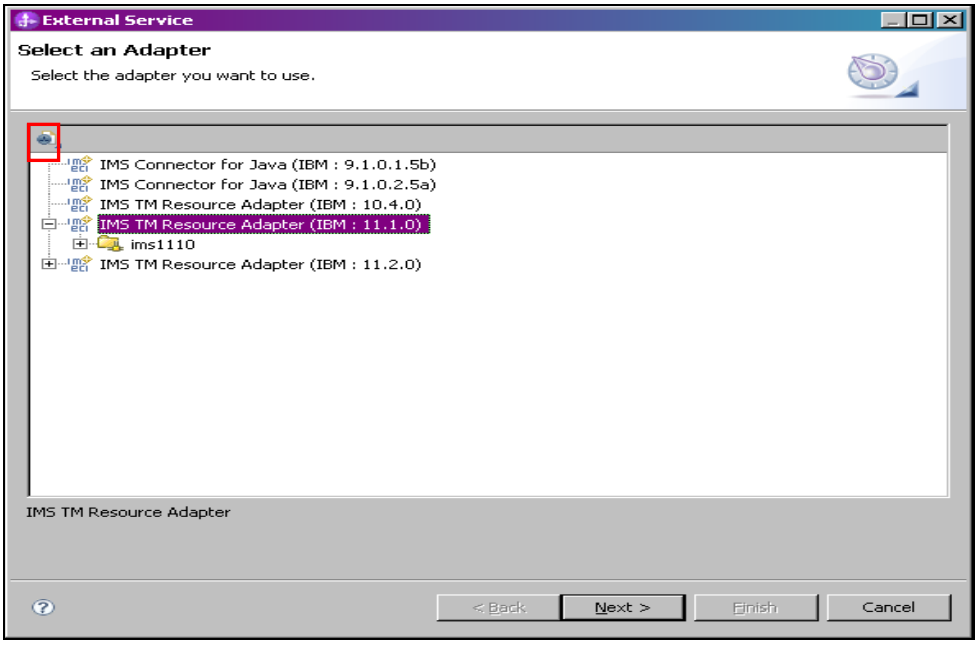
ii. IMS *Resource adapter* 11.1 seçilir,

iii. ims1110 seçilir,

iv. eis/IMS,

Aşağıdaki gibi IMS TM *Resource Adapter* (IBM: 11.1.0) görüntülenmezse sol üst köşedeki butona tıklanarak ims1110.rar dosyası kullanılarak tanımlanır. Şekil 3.15'te *external* servis seçimi gösterilmiştir.

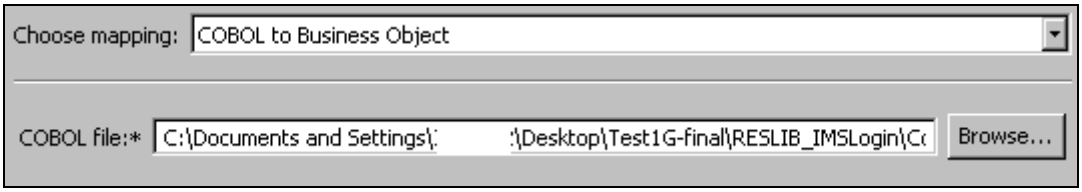
Şekil 3.15: *External* servis seçimi



Not: *Admin* panleden bakıldığında IMS1110 olduğu halde *mediation* sırasında bunu tekrar eklemenin sebebi; *Runtime enviroment* ile *development enviroment*'ı eşitlemektir. İki ortam birbiri ile senkronize olmalıdır.

i. *Authentication name* girilir: IMSAuthAlias
ii. Java Naming and Directory Interface (JNDI) *Name*: eis/IMS tanımlıysa otomatik gelir tanımsızsa *New* butonuna tıklanır.
iii. *Next - Add* - seçilir,
iv. Operasyon ismi girilir,
v. *New – Browse* seçilir,
vi. *Cobol* to Business Objects (BO) seçilir, Şekil 3.16'da gösterilmiştir,

Şekil: 3.16: Mapping tipinin seçimi



vii. *Input copybook* seçilir *main copybook'*a alt Program (PGM)'ın *copybook*'u özel data alanı şeklinde eklenir

viii. *Input* ve *output copybook'*ların alınacağı dizinlerde Türkçe karakter olmamalıdır, dizin çok uzun olmamalıdır.

ix. *Encoding z/os 1026* seçilir,

x. *Find* seçilip, *copybook parse* edilir,

xi. TRX-INPUT seçilir, *preserve case of names* seçilerek *copy*book'daki alan isimleri korunur,

xii. Yukarıdaki işlemin aynısı *Output copybook* için de yapılır, *finish* denir.

Gelen yeni ekranda;

i. *Time out* 10000 girilir bu 10 saniye anlamına gelmektedir.

ii. *Commit* moda COMMIT THEN SEND seçilir.

iii. *Name* için _Transaction *name* yazılır.

iv. *Join global transaction* kaldırılır. Eğer kaldırılmazsa *mediation response'*da talep edilmeyen başka *transaction'*ların *response*'u görülebilinir.

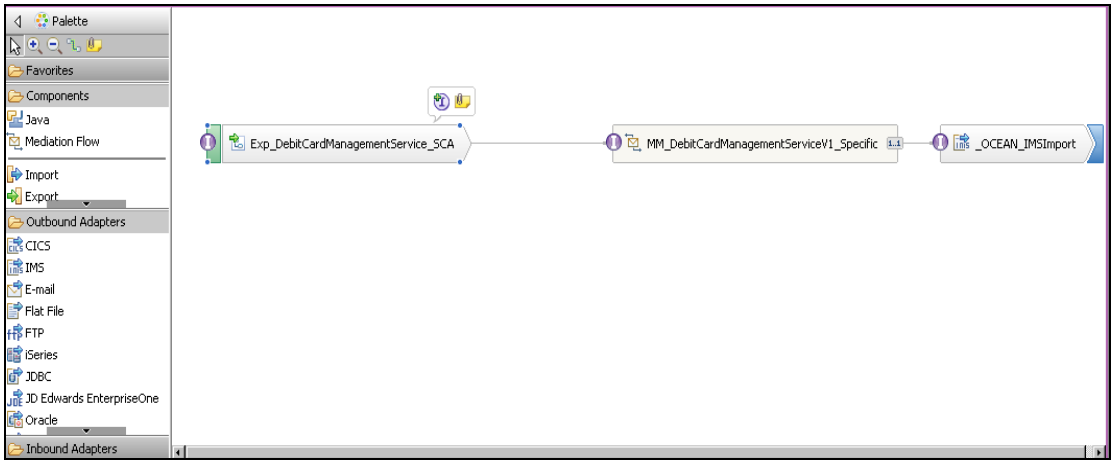
Import oluşturmak için;

i. *Components*'den *Import Mediation* ekranına fare sürüklenerek *import* yaratılır, bakınız Şekil 3.17.

Şekil 3.17: *Import primitive*'i

ii. *Mediation import'*a bağlanır.

Şekil 3.18: Mediation ile import primitive'inin ilişkilendirilmesi



Service Component Architecture (SCA) *export* eklemek için;

i. *Mediation*'ın üzerinde sağ tıklanır,

ii. *Generate export* seçilir, SCA *binding* seçilir. Uyarı gelirse, *export* üzerinde sağ tıklanıp "*Regenarete Binding*" yapılır.

Specific mediation işlemleri

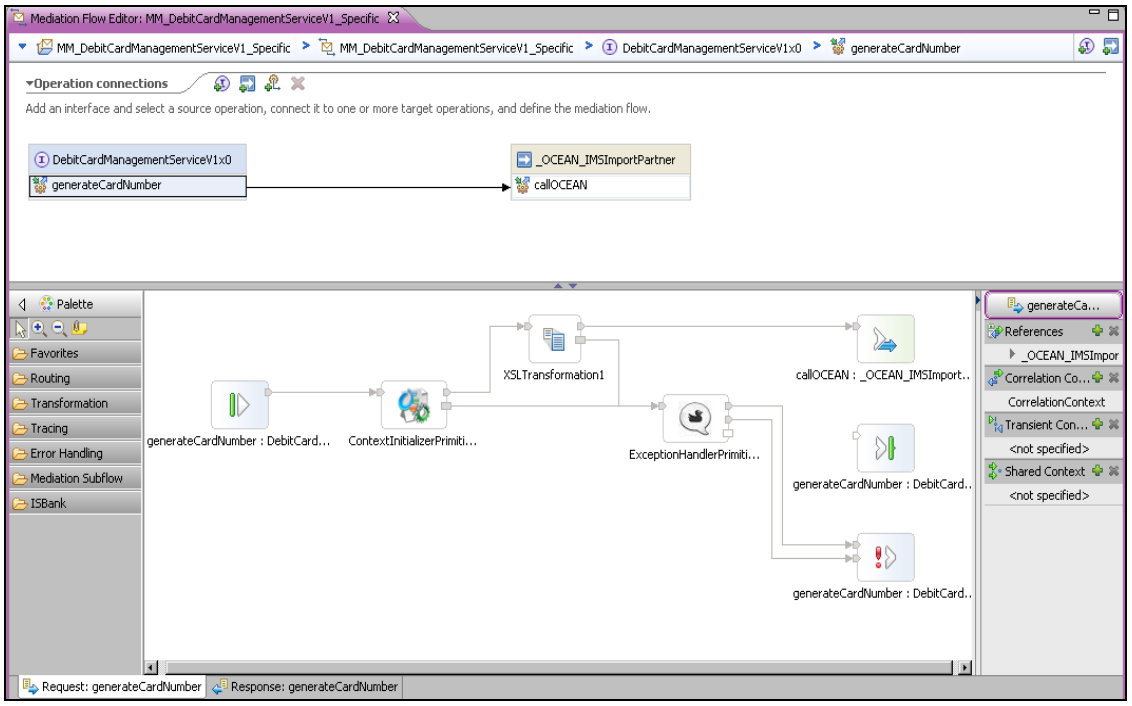
Mapping işleminden önce;

i. *Specific Mediation*'ın dosyasında "*Mediation flow* " kutucuğuna çift tıklanır, kutucuk Şekil 3.19'da gösterilmiştir.

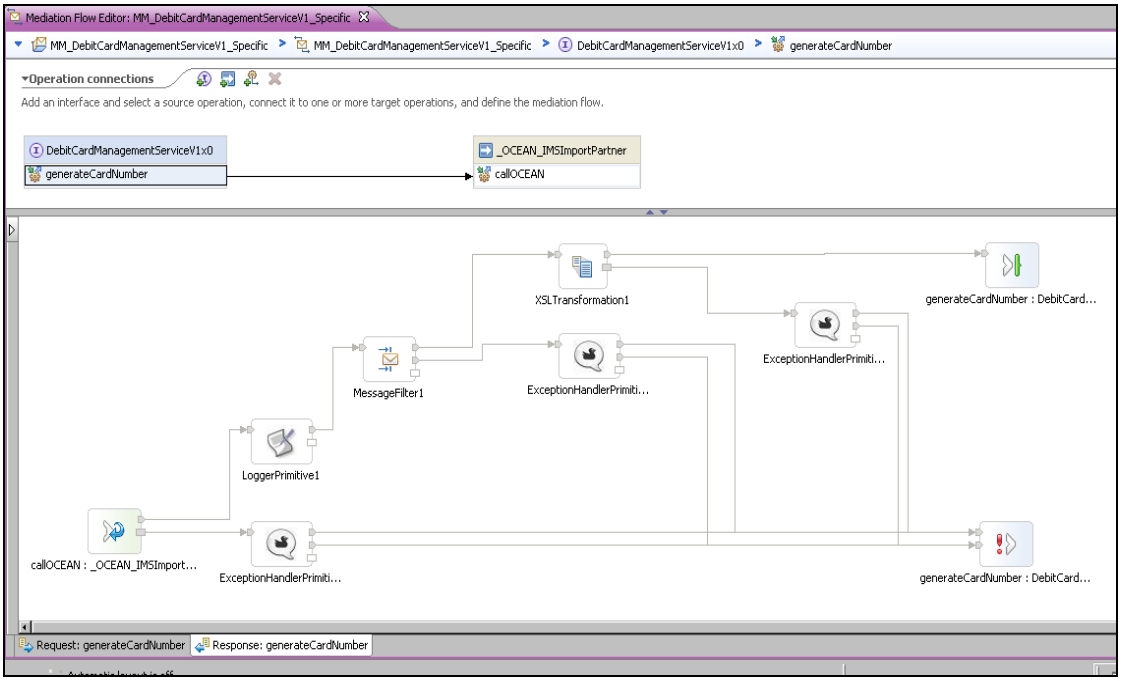
Şekil 3.19: Mediation flow

ii. *Operations Connetions* penceresindeki bağlantı yapıldıktan sonar bakınız Şekil 3.20, *Response* ve *Request flow'*lar *primitive'*ler kullanılarak düzenlenmelidir bu da şekil 3.21'de gösterilmiştir.

Şekil 3.20: IBM WID üzerinde geliştirilmiş özel orta katman servisi (*specific mediation*) örneği, *request flow*



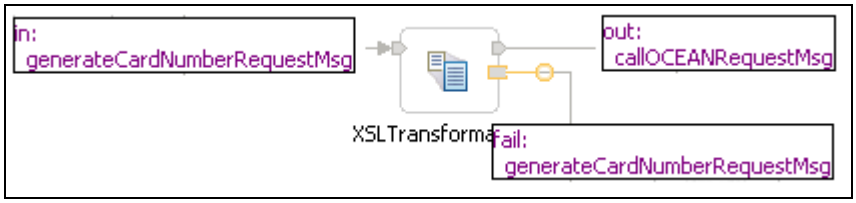
Şekil 3.21: IBM WID üzerinde geliştirilmiş özel orta katman servisi (*specific mediation*) örneği, *response flow*



XSL Transformation Primitive'i Kullanımı:

i. *Mediation*da kullanılan XSL *Transformation primitive*'inin *input* terminalinin *type*'ı ile *fail* terminalinin *type*'ı aynı olmalıdır. Bu terminal grafiksel olarak Şekil 3.22'de gösterilmiştir.

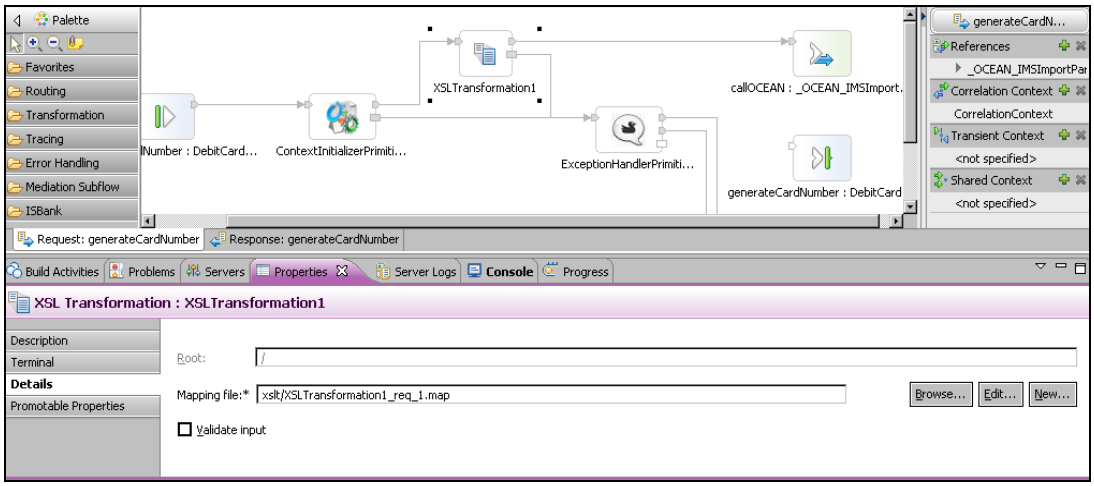
Şekil 3.22: Bir XSLT *primitive*'inin terminalleri (*in, out, fail*)



ii. *Output* terminalinin *type*'i ise bu *primitive*de transformation yapıldığından farklı (IMS *Outbound adapter* oluşturulurken verilen "*Operation Name*") olmalıdır.

iii. *Header* ve *context* bölümlerinin *mediation*da gelmesi için *mediation* oluşturulurken "/*body*" yerine "/" seçilmelidir, bakınız Şekil 3.23.

Şekil 3.23: XSL *transformation*'da *root*'un belirlenmesi



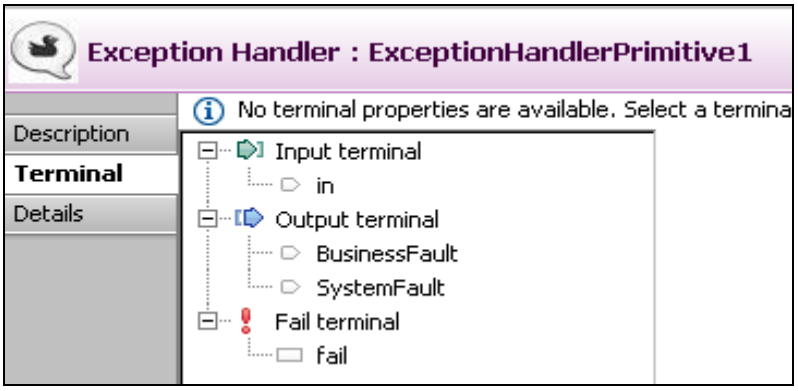
BO Mapper Kullanımı:

Karmaşık *mediation* işlemlerinde XSL *Transformation* yerine BO *Mapper* kullanılır. Örneğin *request* olarak bir *array* alındığında ve *response* olarak bir matris döndüğünde bu BO *mapper* ile çözümlenebilinir.

Exception Handler Primitive'i Kullanımı:

ESB entegrasyon katmanında hata kodundan servis ve dile göre farklı hata mesajları üretilir, detay alanında yazılımcıya yönelik detaylı hata verilir, mesaj alanında ise genel hata mesajı vardır. *Exception Handler Primitive*'inin iki *output* terminali vardır, Şekil 3.24'te de gösterilmiştir, bunlar: (yanlış girilen bilgilerden alınan hatalar için) *Business Fault* ve (sistem hataları için) *System Fault*'tur.

Şekil 3.24: *Exception Handler Primitive*'inin terminalleri

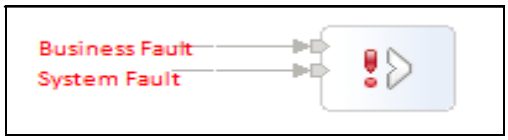


Herhangi bir *primitive*'in *output*'u *Exception Handler Primitive*'ine *input* olabilir. Şekil 3.25'te *Exception Handler Primitive'inin* terminal dağılımı özetlenmiştir.

Şekil 3.25: Exception handler primitive'inin terminal dağılımı

Input Fault Primitive'inin ilgili *Input* terminallerine *Exception Handler Primitive*'inin *Business Fault* ve *System Fault* terminalleri bağlanır. Şekil 3.26'da *Input Fault Primitive*'ine hataların atanması gösterilmiştir.

Şekil 3.26: Input Fault Primitive'ine Hataların Atanması

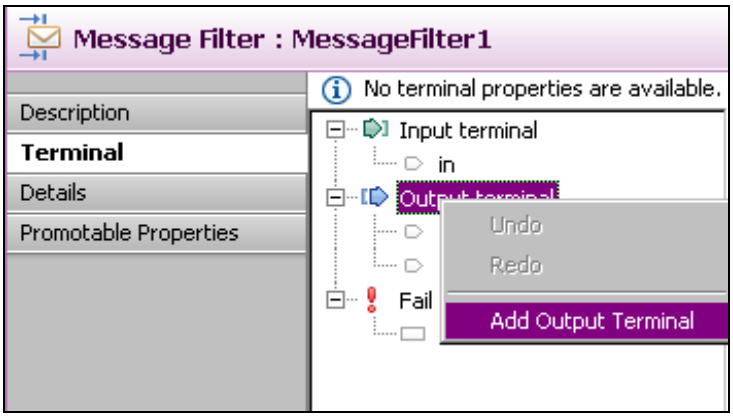


MessageFilter Primitive'i Kullanımı:

Message Filter buradan dönen *Return Code*'u kontrol ederek doğruysa *mapping* işleminin yapıldığı *primitive*'e (XSL *Transformation*), yanlışsa *Exception Handler Primitive*'ine yönlendirir.

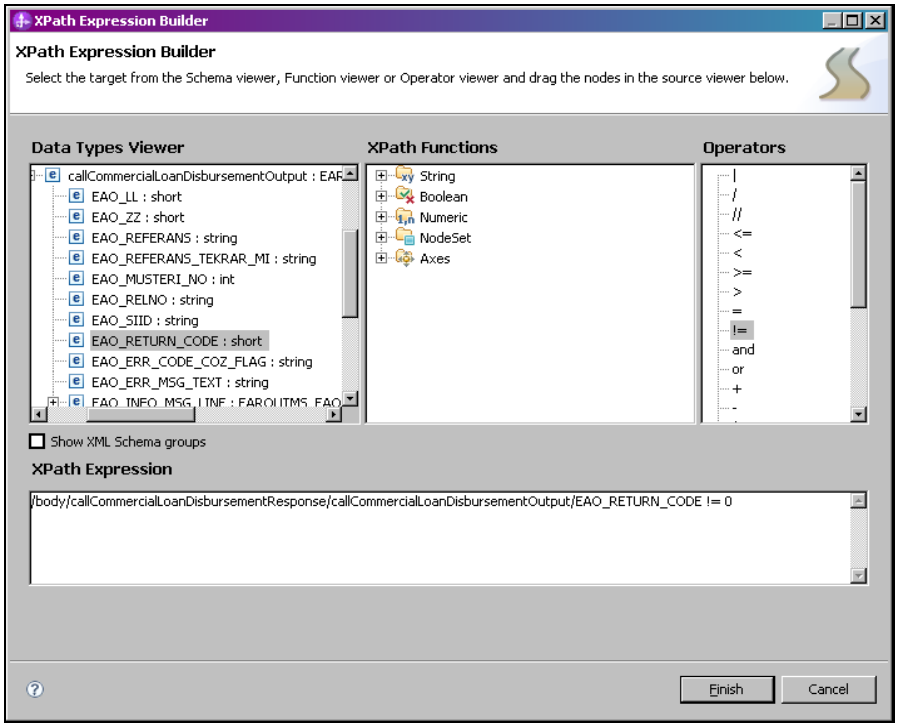
Message Filter'a yeni bir *output* terminal eklenir, terminalin eklenme penceresi Şekil 3.27'de gösterilmiştir. *Error Match* olarak isimlendirilir:

Şekil 3.27: *Message Filter*'a *output* terminali ekleme



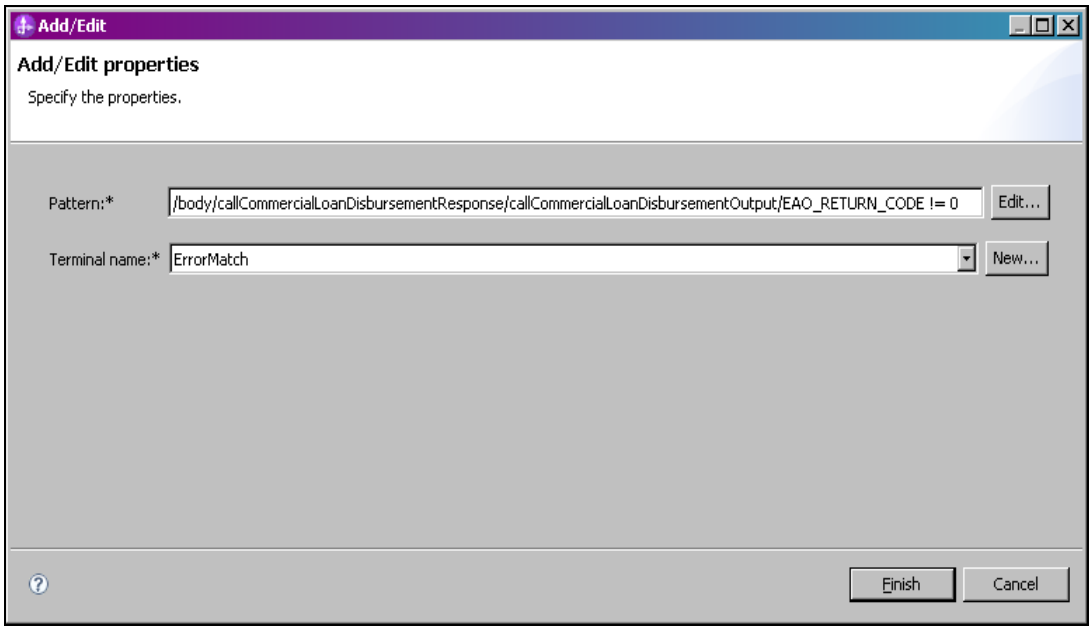
Bu *primitive*'in *Properties*'inden *Details* penceresini açıp, *edit* seçeneğine tıklanarak Şekil 3.28'de gösterilen pencereye ulaşılır, istenen koşul girilir:

Şekil 3.28: XML Path Language (XPATH) ile koşul oluşturmak

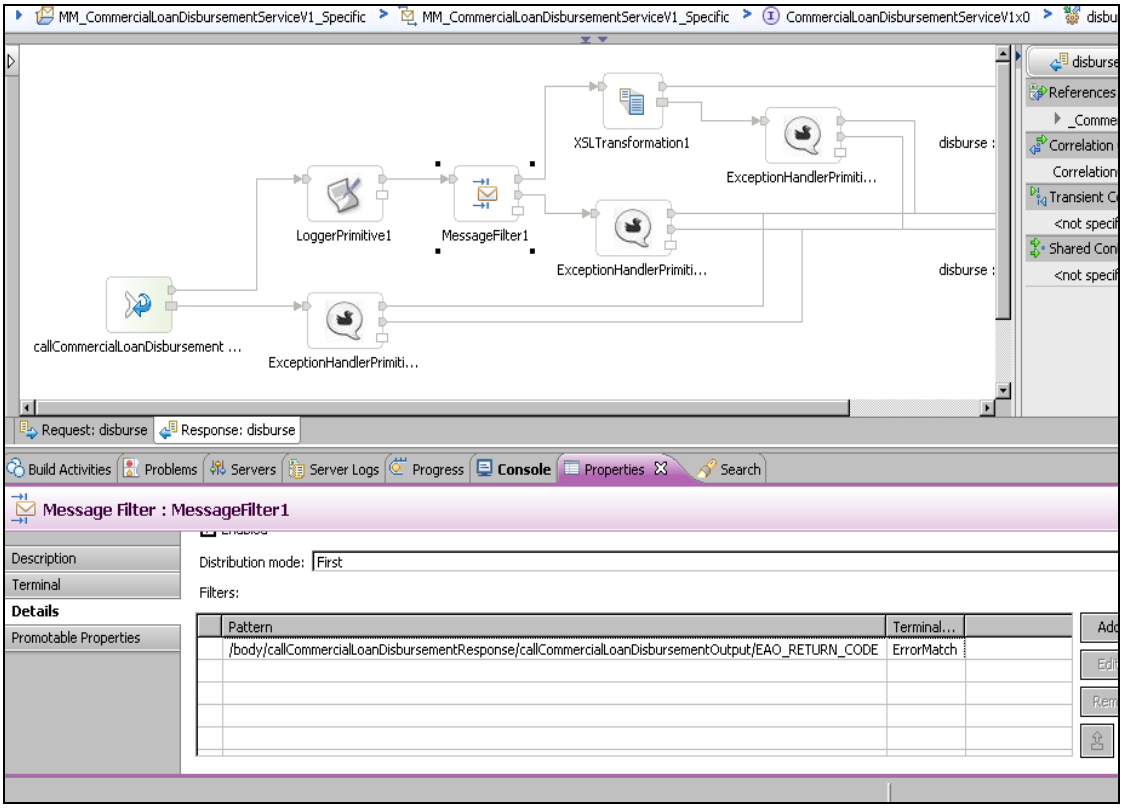


Finish seçildiğinde aşağıdaki ekran gelir. *Terminal name* olarak tanımladığımız *ErrorMatch* terminali seçilir. Seçimin yapıldığı ekran görüntüsü Şekil 3.29 ve Şekil 3.30'da verilmiştir.

Şekil 3.29: XPATH ile oluşturulan koşul gerçekleştiğinde akışın belirleneceği terminali seçmek

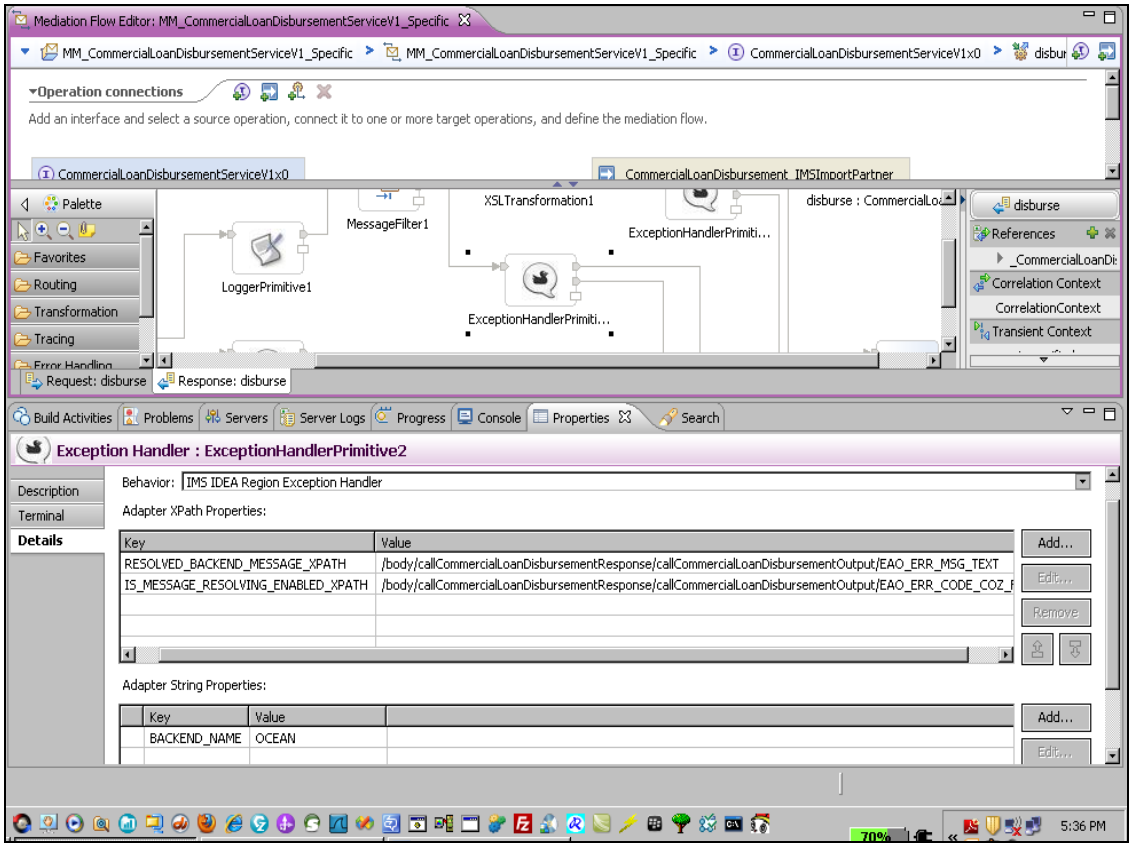


Şekil 3. 30: Uygulanan filtrenin listelenmesi



Hata mesajlarının görüntülenmesi için *ErrorMatch* çıkışının bağlı olduğu *Exception Handler Primitive*'inde aşağıdaki tanımlar yapılmalıdır. Ayrıca servisin response'unda yer alan hata mesajı ve kodunun alınması Şekil 3.31 ile gösterilmiştir.

Şekil 3.31: Servisin *response*'unda yer alan hata mesajı ve kodunun alınması



Mapping

XSL *Transformation* üzerine çift tıklanarak *mapping*'e başlanır.

Hazır Java kütüphaneleri kullanarak *response* olarak gelen datayı formatlayabilmek için;

i. *Custom mapping* seçilir *Properties*'den Java seçilir,

ii. *Class* olarak: IMSParameterConverter seçilir,

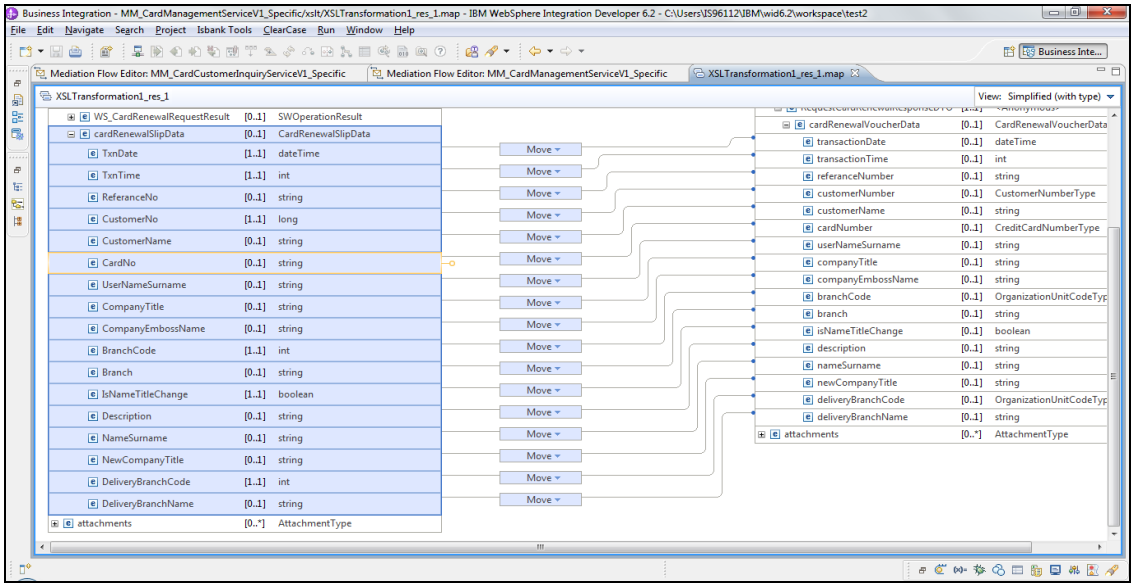
iii. Method olarak: getDay(string) seçilir,

iv. Parametre olarak *birthdateX* seçilir buna ulaşmak için: *Custom Xpath parameter*'dan CTRL + Boşluk tuşlarına basılıp metot aranır. Örneğin; 20111312 şeklinde gelen değer yyyy aa gg şeklinde formatlanabilir.

Spesifik işlemler için Java kodu yazılacaksa, bu bir Java projesi oluşturularak gerçekleştirilebilinir. Bunun için *mediation* klasörüne oluşturulan Java proje dosyasını *dependecy* olarak tanımlamak gerekir.

Java kodu yazılmak istenen proje altında, Java dosyası oluşturmak için *mediation* (MM_...._VX) klasörü seçiliyken *New*- Java seçenekleri ile Java *class* oluşturulur, ancak bu dosya *Bussiness Integration view* da gözükmeyecektir, *Physical Resources View*'da Java dosyası da görülebilinir. Şekil 3.32'de bir *specific mediation*'ın içerdiği XSL *transformation primitive*'ine ait WID ekran görüntüsü verilmiştir. Entegrasyonu sağlanan servislerin alanları arasında yapılmış *mapping* işlemi burada görülebilir.

Şekil 3.32: Bir *specific mediation*'ın içerdiği XSL *transformation primitive*'ine ait WID ekran görüntüsü

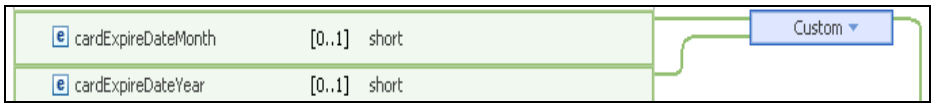


Mapping sırasında iki alanı birleştirmek için;

i. Sürükle bırak ile iki alanı temsil eden kutu aynı kutuya işaretlenir,

ii. *Mediation* tipi olarak *Custom* seçilir, Şekil 3.33'te bu işlemin nasıl yapıldığı gösterilmiştir.

Şekil 3.33: *Custom mediation* yaratmak

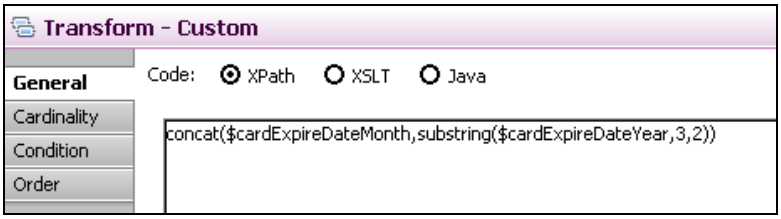


iii. İki alanın hangi değerlerinin alınacağı bilgisi ise aşağıdaki gibi *Custom* nesnesinin *Properties, General, Xpath* seçenekleri seçilerek *concat* komutu ile yapılır. Bu işlem Şekil 3.34'te gösterilmiştir.

a. *Properties General*'a :

concat($cardExpireDateMonth,substring($cardExpireDateYear,3,2))

Şekil 3.34: Xpath ile concat işlemi

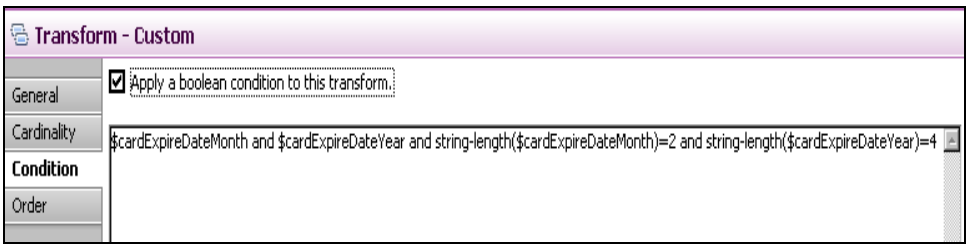


b. *Properties Condition'a* :

$cardExpireDateMonth and $cardExpireDateYear and string-length($cardExpireDateMonth) = 2 and string-length($cardExpireDateYear) = 4

Şekil 3.35'te Xpath ile koşulun nasıl yaratıldığı gösterilmiştir.

Şekil 3.35: Xpath ile koşul yaratmak



LL Değerinin Bulunması İçin:

LL değeri *copybook*'ta yer alan alanlar ve *header* bilgilerinin toplam uzunluğu ya da *copybook*'un maksimum boyudur.

Body'de eşleyecek hiçbir şey olmadığı zaman yani *copybook*'ta bir alana *default* bir değer atanması gerekiyorsa *LocalMap* kullanılır.

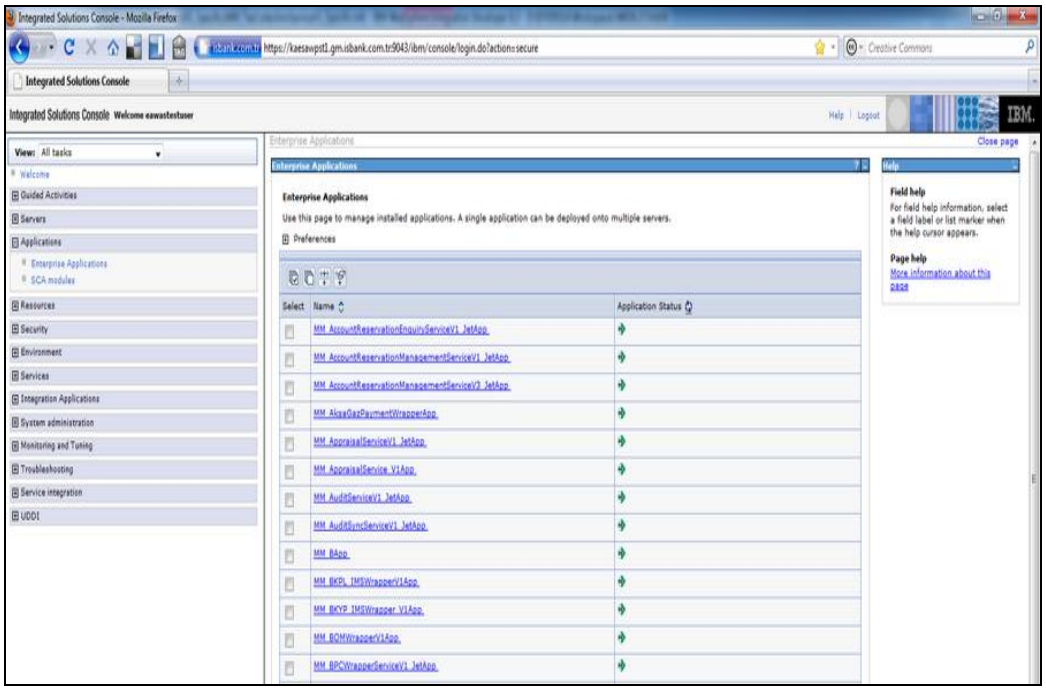
Deployment İşlemleri

Çalışır durumdaki ESB Server üzerinde;

i. Sağ tıklanır, *Add - Remove Projects* seçilir,

ii. Açılan pencerede proje sağ tarafa taşınarak çalışılan lokalde projey *deploy* edilmiş olunur. IBM ESB servisinin yüklü olduğu sunucu yönetim ekranı örneği Şekil 3.36'da verilmiştir.

Şekil 3.36: IBM ESB servislerinin yüklü olduğu sunucu yönetim ekranı



4. BULGULAR

4.1 MEVCUT ENTEGRASYON UYGULAMALARININ DEZAVANTAJLARI

Entegrasyon servislerinin geliştirme ortamını ve süreçlerini anlattıldığı yukarıdaki platformlarda, yazılım servislerini entegre etmek için geliştirilen entegrasyon servislerinin maliyeti gittikçe hantallaşan ve yönetimi zorlaşan bu yapılarda oldukça fazladır. Örneğin; basit sayılabilecek iki *web* servisin haberleşmesi için geliştirilen bir entegrasyon servisinin üç adam/gün geliştirme, test ve kurulum maliyeti olduğu hesaplanmaktadır. Daha fazla servis içeren daha karmaşık entegrasyon servislerinde bu maliyet artmaktadır.

Birden fazla geliştirici ekip tarafından geliştirilen entegrasyon servislerinde standardizasyonun sağlanması ve versiyon kontrolü zorlaşmaktadır.

Ayrıca, entegre edilmek istenen servislerde yapılacak yeni değişiklikler, *frontend* uygulama katmanı, *backend* uygulama katmanı ve entegrasyon katmanlarında koordinasyonu gerektirir.

Entegre edilmek istenen servislerde olası muhtemel değişiklikler şu şekilde sıralanabilir:

i. Servislerin *request*/*response*'larına yeni alan eklemesi / çıkarılması,
ii. Servislerin *request*/*response*'larında var olan alanların meta-data'sının değiştirilmesi,
iii. Servislerin *request*/*response*'larında var olan alanların sırasının değişmesi,
iv. Servislerin adreslerinin değişmesi,
v. Servislerde *authentication* kullanılmaya başlanması.

Yukarıdaki değişikliklerin birebir entegrasyon servisine uygulanması gerektiğinden, entegrasyon servislerinin bakımı ve yönetimi zorlaşmaktadır.

Küçük ve orta ölçekli işletmeler maliyet açısından SOA modelini kullanan teknolojileri satın almakta zorlanmaktadırlar.

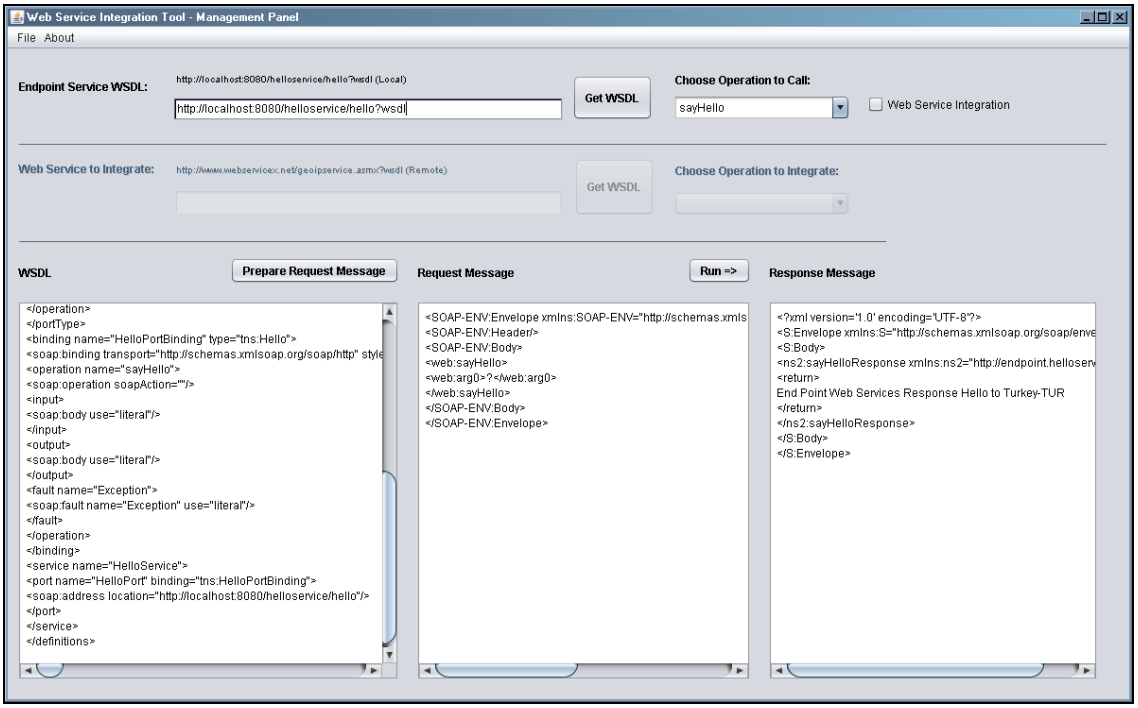
4.2 İHTİYAÇ: *LIGHTWEIGHT* SOA

Tez konusu kapsamında, SOA metadolojisini kullanan entegrasyon araçlarıyla (Bakınız Bölüm 3.3) entegrasyon servisi geliştirilmesi ve bu servislerin yönetimi incelenmiştir. Yapılan incelemeler sonucunda, entegrasyon servisi geliştirme ve yönetiminin kolaylaştırılması gereği ortaya koyulmuştur. Ayrıca, küçük ve orta ölçekli işletmelerin maliyet açısından SOA modelini kullanan teknolojileri satın almakta zorlanmaları yine basitleştirilmiş (*lightweight*) SOA ihtiyacını doğurmaktadır. Bu sebeple *lightweight* SOA modelleri geliştirilmelidir. Tez konusu kapsamında, *Lightweight* SOA modeline örnek olabilecek aşağıda detaylı olarak anlatılacak "Web Service Test ve Entegrasyon Aracı" geliştirilmiştir.

4.3 *WEB SERVICE* TEST ve ENTEGRASYON ARACI (*WEB SERVICE TEST* and *INTEGRATION* TOOL)

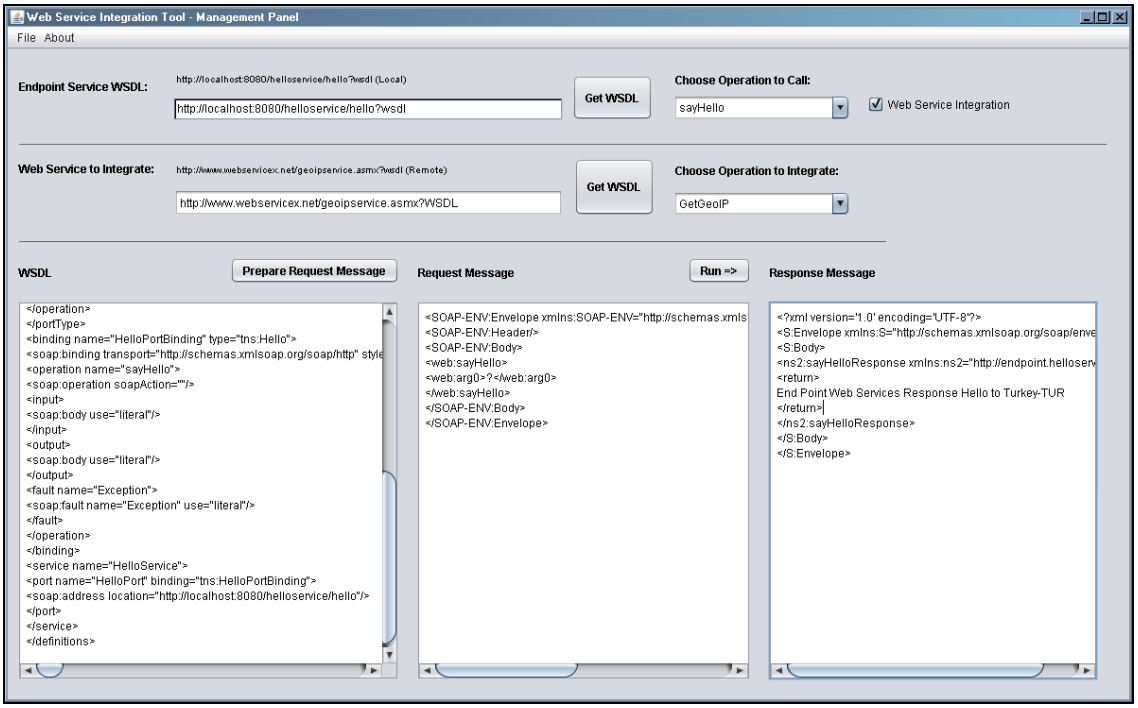
Bölüm 4.2'de bahsedilen *Lightweight* SOA ihtiyacı doğrultusunda geliştirilen yazılımın iki temel kipi bulunmaktadır. Bunlardan ilki *web* servislerin test edilmesi amaçlıdır. Bu fonksiyon amacı itibariyle SOAPUI programı ile kıyaslanabilinir. Şekil 4.1'de *web service* test ve entegrasyon aracı test kipi ekran görüntüsü verilmiştir.

Şekil 4.1: *Web service* test ve entegrasyon aracı test kipi



Yazılımın ikinci kipi ise web servislerin birbirleri ile entegrasyonunu *Lightweight* SOA kapsamında gerçekleştirmektir. Bu amaçla *Web Service* Test ve Entegrasyon Aracı'nın bir yönetim paneli bulunmaktadır. Yönetim panelinin görüntüsü Şekil 4.2'deki gibidir.

Şekil 4.2: *Web service* test ve entegrasyon aracı yönetim paneli



Yönetim panelinden entegre edilecek servislere ait WSDL adresleri, entegre edilecek operasyon bilgileri gibi bilgiler girilir. Bu işlemden sonra arka planda daha sonra entegre edilecek serviste kullanmak üzere parameter.xml dosyası oluşur.

Servis tarafında ise parameter.xml dosyasını kullanan bir Java paketi mevcuttur. *Client* (İstemci) tarafı, web servisi çağırdığında web servis önce entegrasyon katmanı görevi gören *Integrator* isimli Java paketini çalıştırır. Bu paket vasıtasıyla entegre olunacak diğer web servise gidilir, bu diğer servisten yanıt alınır ve tekrar ilk servise yanıt dönülür. İlk çağırılan servis de bu yanıtı *client* uygulamaya geri döner. Böylelikle iki web servis birbirleriyle *Integrator* olarak adlandırılan Java yazılımı ile ortak bir konuşma yapısı geliştirmiş olurlar.

4.3.1 Genel Mimari

Geliştirilen *Web Service* Test ve Entegrasyon Aracı'nın kullanım algoritmasını şu şekilde özetleyebiliriz:

i. *Web Service* Test ve Entegrasyon Aracı ekranından; *Web* servisin WSDL adresi *Endpoint* Servis WSDL kutucuğuna girilip, *Get* WSDL butonuna basıldığında WSDL C:\WebServiceIntegration\endpoint.wsdl adresine indirilir,

ii. C:\WebServiceIntegration\endpoint.wsdl dosyasından okuma yapılarak, *web* servisin WSDL'ında tanımlı olan tüm operasyonlar belirlenir ve açılır kutu içinde listelenir.

iii. C:\WebServiceIntegration\endpoint.wsdl dosyası okunarak WSDL sahasında görüntülenir.

iv. Açılır kutudan birinci *web* servisin hangi operasyonu entegre edilmek isteniyorsa bu operasyon seçilir.

v. *Web Service Integration* seçeneği işaretlenir. Bu seçenek işaretlendiğinde entegre edilmek istenen servisin WSDL adres bilgilerinin girileceği alanlar aktif olur,

vi. İkinci *web* servisin WSDL adresi *Service to Integrate* kutucuğuna girilir,

vii. Bu servis için ikinci *Get* WSDL butonuna basıldığında WSDL C:\WebServiceIntegration\integrated.wsdl adresine indirilir,

viii. C:\WebServiceIntegration\integrated.wsdl dosyasından okuma yapılarak, 2. *web* servisin WSDL'ında tanımlı olan tüm operasyonlar, *Choose Operation to Integrate* açılır kutusu içinde listelenir,

ix. *Prepare Request Message* butonuna basılarak hangi operasyon üzerinden entegrasyon sağlanacaksa bu operasyon için *request message* oluşturulur ve *Request Message* sahasında görüntülenir.

x. Ayrıca entegre edilmek istenen servisler ile ilgili C:\WebServiceIntegration\parameters.xml dizininde bir parametre dosyası oluşur. Bu dosyanın deseni aşağıdaki gibidir:

   a. Entegre Edilecek Servis URL'i@Entegre Edilcek Servis targetNamespace'i
   b. Oluşturulan dosya entegrasyon katmanında kullanılacaktır.

xi. Eğer parametrik bir request hazırlanması isteniyorsa *Web Service* Test ve Entegrasyon Aracı ekranından, *request'*in içine eklenmesi gereken parametreler eklenir,

xii. *Run* butonuna basılarak *web* servisin seçilen operasyonu hazırlanan *request* ile çağırılır, bu adımda *Web Service* Test ve Entegrasyon Aracı'nı istemci (*client side*) uygulama olarak da düşünebiliriz.

xiii. *Client* tarafından gelen istek *web* servise geldiğinde *web* servisin içinde daha önceden tanımlanmış Integrator isimli Java paketi içinde yer alan SOAPcaller(); metodu ile *web* servis başka bir entegrasyon servisine entegre edilebilir hale gelir.

xiv. Toplu olarak *web* servislerin SOAPcaller(); methodunu çağıracak şekilde modifiye edilmesi gerekiyorsa bunun için özel olarak hazırlanmış bir script kullanılabilir.

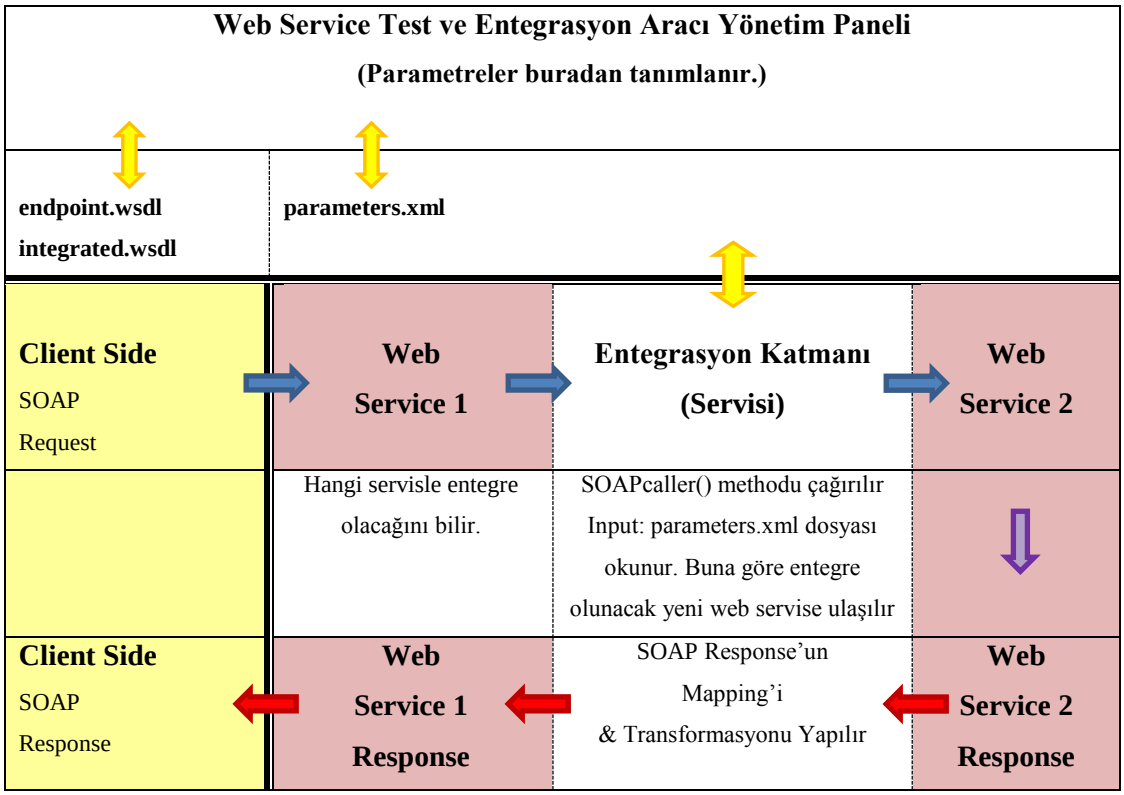
xv. SOAPcaller(); metodu daha önceden bahsedilen entegre olunacak servisin bilgilerini içeren parameters.xml dosyasını okur. Bu bilgilere göre entegre olunacak servisin hangi operasyonunu nasıl bir *request* ile çağıracağını bilir.

xvi. SOAPcaller(); methodu entegre olunacak servisin hangi *response*'u döneceğini de bilir, bu *response*'u *client* tarafından ilk çağırılan *servisin response*'una ekleyerek *client* tarafa geri döner.

xvii. Gelen yanıt *Response Message* sahasında görüntülenir.

Web servis entegrasyon aracının genel mimarisi ve katmanların birbirleri ile olan iletişimi genel olarak Tablo 4.1'de aktarılmıştır. Tabloda yer alan beyaz kutular entegrasyon alt yapısını oluşturmak için yazılın bölümlerini içermektedir.

Tablo 4.1: Web Servis Entegrasyon Aracı'nın genel mimarisi, katmanların birbirleri ile olan iletişimi



4.3.1.1 Kullanılan teknolojiler

Web Service Test ve Entegrasyon Aracı'nı geliştirilirken; NetBeans Integrated Development Enviroment 7.4, yazılım geliştirme dili olarak Java *Development Kit (JDK)* 1.7, JDK 1.7'yi destekleyen tek uygulama sunucusu olan GlassFish Server 4.0 ve *Graphical User Interface* olarak ise Swing kütüphanesi kullanılmıştır. Ayrıca mesajlaşma yapısının kurulmasında SOAP kütüphanelerinden yararlanılmıştır. Yönetim panelinin geliştirilmesinde kullanılan tüm kütüphaneler aşağıdaki gibidir:

i. javax.xml.soap.MessageFactory
ii. javax.xml.soap.SOAPBody
iii. javax.xml.soap.SOAPConnection
iv. javax.xml.soap.SOAPConnectionFactory
v. javax.xml.soap.SOAPElement
vi. javax.xml.soap.SOAPEnvelope
vii. javax.xml.soap.SOAPException

viii. javax.xml.soap.SOAPMessage
ix. javax.xml.soap.SOAPPart
x. javax.xml.transform.Source
xi. javax.xml.transform.Transformer
xii. javax.xml.transform.TransformerFactory
xiii. javax.xml.transform.stream.StreamResult
xiv. java.awt.FlowLayout
xv. java.io.*
xvi. java.io.ByteArrayOutputStream
xvii. java.net.HttpURLConnection
xviii. java.net.URL
xix. java.util.ArrayList
xx. java.util.HashMap
xxi. java.util.List
xxii. java.util.Scanner
xxiii. java.util.logging.Level
xxiv. java.util.logging.Logger
xxv. javax.swing.JFrame
xxvi. javax.swing.JprogressBar

Entegre olunacak servisin çağrılmasını sağlayan, gerekli transformasyon işlemlerini yapan entegrasyon katmanı (servisi) görevi gören yazılımın herhangi bir ara yüzü yoktur. Gerekli parametreleri yönetim panelinin oluşturulduğu parameter.xml dosyasından alır. Bu modülün geliştirilmesinde kullanılan kütüphaneler aşağıdaki gibidir:

i. java.io.BufferedReader
ii. java.io.ByteArrayInputStream
iii. java.io.ByteArrayOutputStream
iv. java.io.DataInputStream
v. java.io.File
vi. java.io.FileInputStream
vii. java.io.IOException

viii. java.io.InputStream
ix. java.io.InputStreamReader
x. java.util.HashMap
xi. java.util.Scanner
xii. javax.xml.soap.MessageFactory
xiii. javax.xml.soap.SOAPBody
xiv. javax.xml.soap.SOAPConnection
xv. javax.xml.soap.SOAPConnectionFactory
xvi. javax.xml.soap.SOAPElement
xvii. javax.xml.soap.SOAPEnvelope
xviii. javax.xml.soap.SOAPException
xix. javax.xml.soap.SOAPMessage
xx. javax.xml.soap.SOAPPart

4.3.2 Kullanım Kipleri

4.3.2.1 Test kipi

Geliştirilen *Web* Service Test ve Entegrasyon Aracı ile *web* servisler tıpkı SOAPUI programındaki gibi test edilebilir. Test kipi ekran görüntüsü Şekil 4.3 ile verilmiştir.

Bunun için;

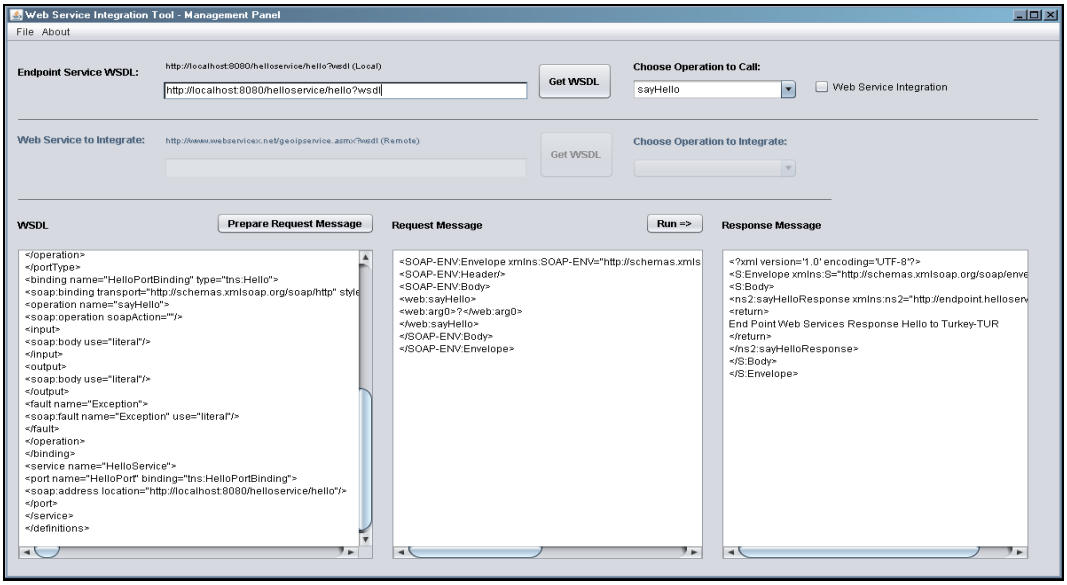
i. *Web* servisin WSDL adresi *Endpoint* Servis WSDL kutucuğuna girilir.
ii. *Get* WSDL butonuna basılır. Geliştirilen yazılım sayesinde *web* servisin WSDL'ında tanımlı olan tüm operasyonlar, açılır kutu içinde listelenir. Ayrıca *Get* WSDL butonuna basıldığında WSDL indirilerek WSDL kutucuğunda görüntülenir.
iii. Açılır kutudan *web* servisin hangi operasyonu test edilmek isteniyorsa bu operasyon seçilir. *Prepare Request Message* butonuna basılarak hangi operasyon çağırılacaksa bu operasyon için *request message* oluşturulur ve *Request Message* sahasında görüntülenir.

iv. Eğer parametrik bir *request* hazırlanmak isteniyorsa, *request'*in içine eklenmesi gereken parametreler eklenir,

v. *Run* butonuna basılarak web servisin seçilen operasyonu hazırlanan *request* ile çağırılır,

vi. *Web* servisten gelen yanıt *Response Message* sahasında görüntülenir.

Şekil 4.3: *Web service* test ve entegrasyon aracı test kipi



Yazılımın ikinci fonksiyonu ise Service *Oriented Architecture* (SOA) yaklaşımı kapsamında sıklıkla ihtiyaç duyduğumuz *web* servislerin birbirleri ile haberleşmelerini sağlayan entegrasyonunu içerir.

4.3.2.2 *Web* servis entegrasyon kipi

Geliştirilen *Web Service* Test ve Entegrasyon Aracı ile *web* servisler birbirine entegre edilebilir.

Bunun için;

i. Birinci *web* servisin WSDL adresi *Endpoint* Servis WSDL kutucuğuna girilir,

ii. *Get* WSDL butonuna basılır. Geliştirilen yazılım sayesinde *web* servisin WSDL'ında tanımlı olan tüm operasyonlar, açılır kutu içinde listelenir. Ayrıca

Get WSDL butonuna basıldığında WSDL indirilerek WSDL kutucuğunda görüntülenir.

iii. Açılır kutudan birinci *web* servisin hangi operasyonu entegre edilmek isteniyorsa bu operasyon seçilir.

iv. *Web Service Integration* seçeneği işaretlenir. Bu seçenek işaretlendiğinde entegre edilmek istenen servisin WSDL adres bilgileri girilebilir.

v. İkinci *web* servisin WSDL adresi *Service to Integrate* kutucuğuna girilir,

vi. *Get* WSDL butonuna basılır. Gelişitirilen yazılım sayesinde ikinici *web* servisin WSDL'ında tanımlı olan tüm operasyonlar, *Choose Operation to Integrate* açılır kutusu içinde listelenir.

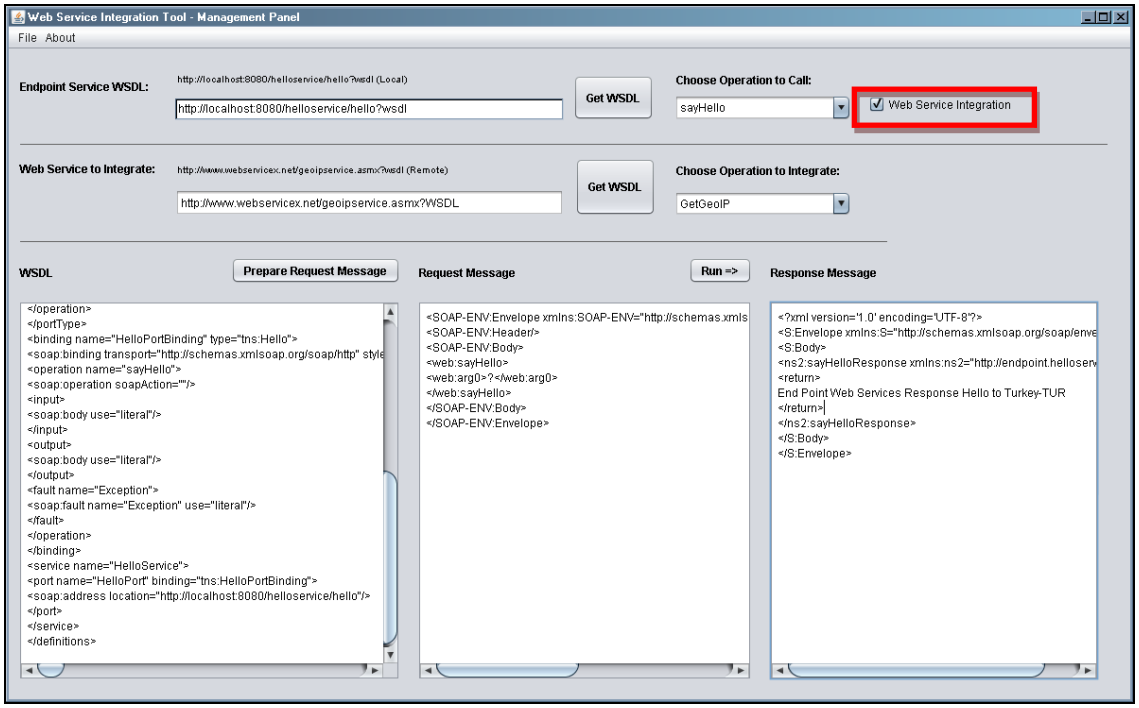
vii. *Prepare Request Message* butonuna basılarak hangi operasyon üzerinden entegrasyon sağlanacaksa bu operasyon için *request message* oluşturulur ve *Request Message* sahasında görüntülenir.

viii. Eğer parametrik bir *request* hazırlanmak isteniyorsa, *request*in içine eklenmesi gereken parametreler eklenir,

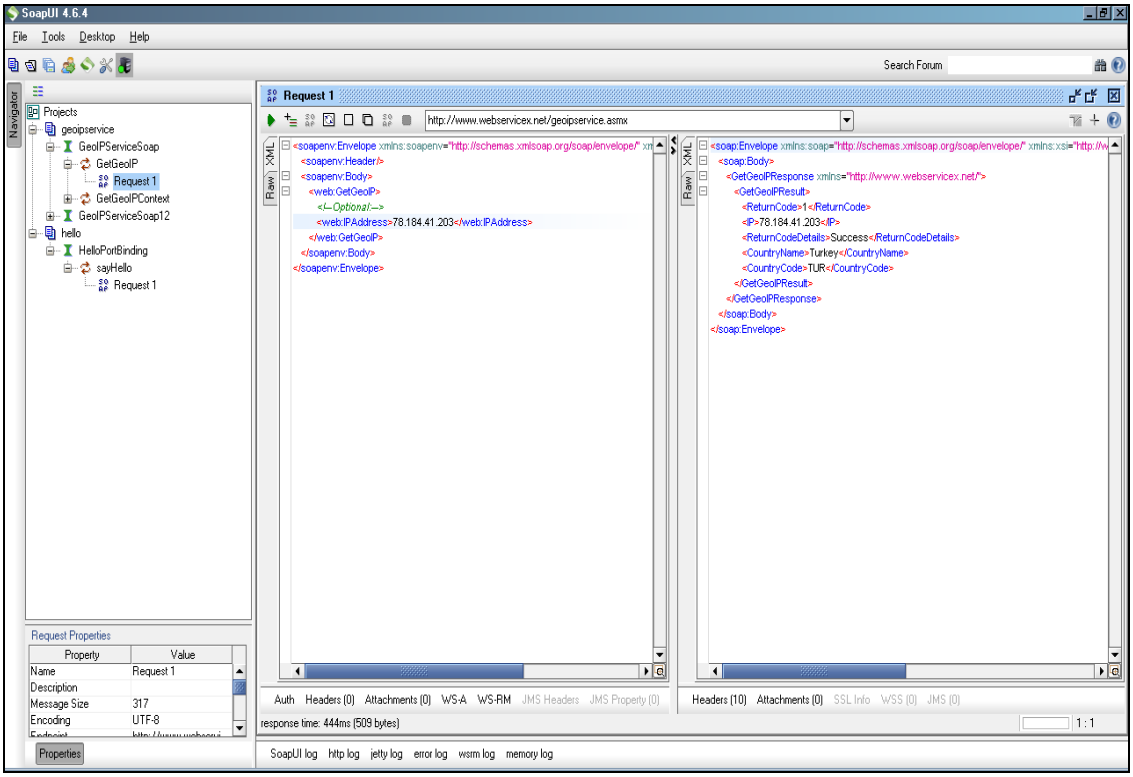
ix. *Run* butonuna basılarak *web* servisin seçilen operasyonu hazırlanan *request* ile çağırılır,

x. *Web* servisten gelen yanıt *Response Message* sahasında görüntülenir. Şekil 4.4'te web service test ve entegrasyon aracı, entegrasyon kipi ekran görüntüsü bulunmaktadır. Ayrıca, Şekil 4.5 ve Şekil 4.6'da gösterildiği gibi web servislerden dönen mesajın sağlaması SOAPUI programı ile de yapılabilinir.

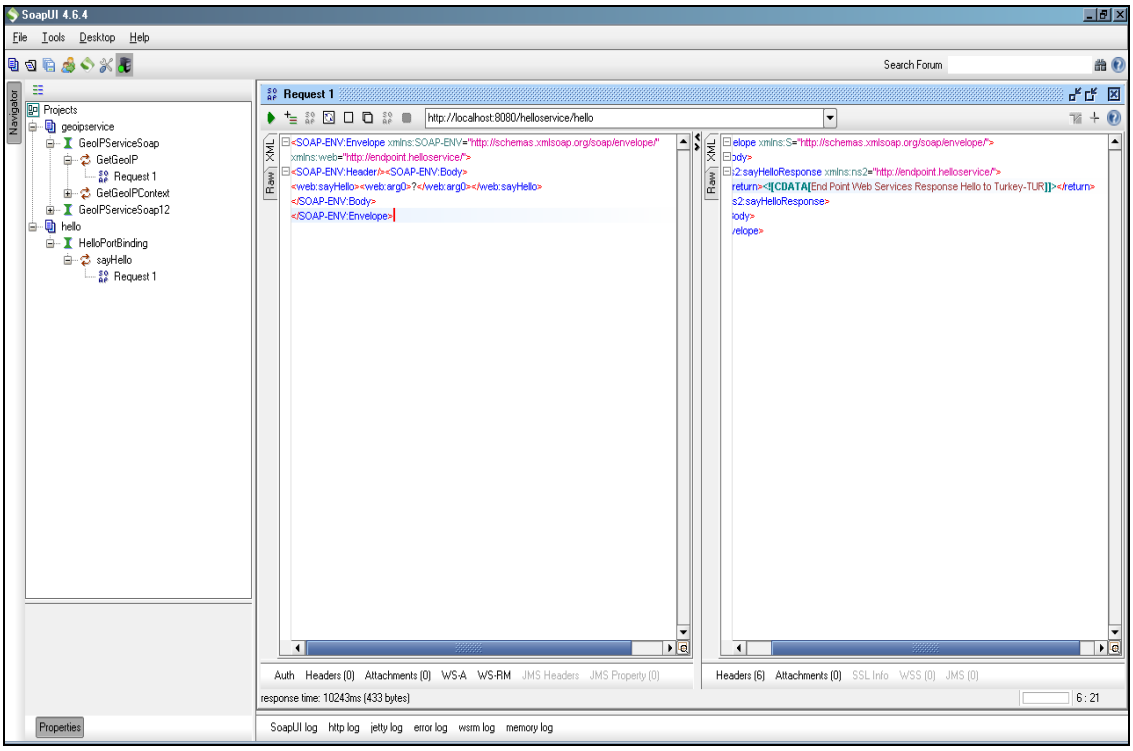
Şekil 4.4: *Web service* test ve entegrasyon aracı entegrasyon kipi



Şekil 4.5: Entegre olunan servisin entegrasyon servisi dışında SOAPUI ile çağırılması ve servisin döndüğü *response*



Şekil 4.6: SOAPUI ile *Lightweight* SOA mimarisi kapsamında geliştirilmiş entegrasyon servisinini çağırılması ve servisin döndüğü *response*



5. SONUÇ VE ÖNERİLER

Her geçen gün artan yazılım ihtiyacı ile büyüyen proje geliştirme maliyetleri, birçok uygulamanın birlikte çalışarak tekrar kullanılabilir olmasını ve dolayısıyla uygulamaların birbirleri ile entegrasyonunu gerekli kılmıştır. Günümüzde kurumlar entegrasyon uygulamalarını yönetmek için çeşitli *middleware* platformlar kullanmaktadırlar.

Entegrasyon platformları üzerinde, yazılım uygulamalarının servislerini entegre etmek için hazırlanan servislerinin geliştirme ve yönetim maliyeti büyüyen sistemlerde gittikçe artmaktadır.

Mevcut entegrasyon platformları ile entegrasyon ihtiyacını karşılamak küçük ve orta ölçekli işletmelerin sağlayamayacağı maliyet boyutundadır. Bunun yanı sıra büyük ölçekli işletmelerde de, birden fazla geliştirici ekip tarafından geliştirilen entegrasyon servislerinde standardizasyonun sağlanması ve versiyon kontrolü zorlaşmaktadır. Ayrıca, entegre edilmek istenen servislerde yapılacak yeni değişiklikler, *frontend* uygulama katmanı, *backend* uygulama katmanı ve entegrasyon katmanlarında koordinasyonu gerektirmektedir. Yukarıdaki değişikliklerin birebir entegrasyon servisine uygulanması gerektiğinden, entegrasyon servislerinin bakımı ve yönetimi zorlaşmaktadır.

Tüm bunların sonucu olarak SOA kullanan entegrasyon platformlarının yönetiminin ve entegrasyon servislerinin geliştirilmesinin kolaylaştırılması bir ihtiyaç haline gelmektedir. Bu ihtiyacı karşılamak için yeni metadolojiler geliştirmek gereği aşikardır.

Entegrasyon servisi geliştirme ve yönetiminin *Lightweight* SOA modeli ile kolaylaştırılabileceği ortaya konmuştur. Bu bağlamda *Lightweight* SOA modeline örnek olabilecek bir "Web *Service* Test ve Entegrasyon Aracı" geliştirilmiştir. Geliştirilen bu yazılımda *web* servislerin test edilmesi ve web servislerin birbirleri ile entegrasyonunu *Lightweight* SOA modeli ile gerçekleştirilmiştir.

Tez konusu olan "SOAP ile Entegrasyon Servislerinin Geliştirilmesi"ni sağlayan entegrasyon katmanı görevi gören yazılımın input olarak kullandığı WSDL versiyonu, *World Wide Web Consortium* (W3C) standartlarından olan SOAP 1.1 ve SOAP 1.2'nin her ikisini de destekler hale getirilmesi sağlanabilir. Ayrıca, entegre olan servislerin *request* veya *response* alanlarının eşleşmesini *mapping* işlemi öncesi gösteren bir ara yüz tasarlanabilir. Bu ara yüz ile *mapping* işlemi yapılmadan kontrol edilme ve olası yanlış *mapping*'leri düzeltme imkanı yaratılabilir.

Ayrıca, bazı web servislerden kullandığı parçalı WSDL (Bakınız Şekil1.2) dökümanları ile ilgili de gelişme sağlanabilir. Parçalı WSDL dokümanlarında görülen, WSDL dokümanının içine import edilen, WSDL'da tanımlı operasyonların *request*/*response* alanlarını, karmaşık veri tiplerini tanımlayan XSD dosyalarının entegrasyon katmanında (Web Servis Test ve Entegrasyon Aracı) *mapping*'lerinin otomatikleştirerek yapılması yönünde çalışmalar yapılabilir.

KAYNAKÇA

Kitaplar

Brown, P. C., 2008. *Implementing soa: total architecture in practice.* Massachusetts: Addison Wesley Professional.

Cerami, E., 2002. *Web services essentials distributed applications with xml-rpc, soap, uddi & wsdl*. California: O'Reilly.

Davis, J., 2009. *Open source soa.* Connecticut: Manning Publications Co.

Erl, T., 2009. *Service-oriented architecture: concepts, technology and design.* 9.Baskı. Indiana: Prentice Hall.

Grosso, W., 2002. *Java rmi.* California: O'Reilly.

Süreli Yayınlar

Erdur, C ve Türksever, M., 2003. Yeniden kullanılabilir yazılım bileşenlerine web üzerinden erişim için corba temelli bir mimari. *Mühendislik Bilimleri Dergisi*. **9** (1), ss.47-54.

Türksever, M. ve Erdur, C., 2001. Farklı bakış açılarından java rmi ve corba'nın, karşılaştırılması. *Mühendislik Bilimleri Dergisi*. **7** (1), ss.63-69.

Diğer Yayınlar

Akyüz, M., 2007. Soa'nın hakimi olmak (soa yönetişimi) [online] http://soa-tr.com/2007/07/soanin-hakimi-olmak-soa-yonetisimi/ [Erişim tarihi 13 Ekim 2013]

Arsanjani, A., 2004. Service-oriented modeling and architecture. *IBM SOA/Web Services Center of Excellence, Software Group,* [online] November 2004, https://www.ibm.com/developerworks/library/ws-soa-design1/ws-soa-design1-pdf.pdf [Erişim tarihi 4 Eylül 2013]

Aydın, G., 2008. Web services. Fırat Üniversitesi [online] 2008 http://web.firat.edu.tr/bilmuh/gaydin/dersler/0809/bmu401/ppt/webservices.doc [Erişim tarihi 13 Ekim 2013]

Baldwin, R.G., 1999. Advanced java programming tutorial remote method invocation. [online] 1999 http://www.dickbaldwin.com/java/Java600.htm [Erişim tarihi 13 Ekim 2013]

Clark, K.J., 2009. Designing an esb gateway in websphere process server and websphere esb. *IBM,* [online] 2009 http://www.ibm.com/developerworks/websphere/library/techarticles/0908_clark/0908_clark-pdf.pdf [Erişim Tarihi 4 Eylül 2013]

Cobol Unit, https://code.google.com/p/cobolunit/source/browse/trunk/cobolunit+--username+hvaujour/COBOLUnit/CPY/CBUC0002.cpy?r=178 [Erişim tarihi 13 Ekim 2013]

Erl, T., 2006. A brief history of soa. [online] http://searchsoa.techtarget.com/answer/A-brief-history-of-SOA [Erişim tarihi 7 Eylül 2013]

IBM, The COBOL Copybook, http://pic.dhe.ibm.com/infocenter/dmanager/v7r5/index.jsp?topic=%2Fcom.ibm.dserver.rulestudio%2FContent%2FBusiness_Rules%2F_pubskel%2FInfocenter_Primary%2Fps_DS_Rule_Designer772.html [Erişim tarihi 20 Ekim 2013]

İmre, M., 2012. Web servis nedir nerelerde kullanılır [online] http://muratimre.blogspot.com.tr/2012/06/web-servis-nedir-nerelerde-kullanlr.html [Erişim tarihi 20 Ekim 2013]

Jeng, JJ., An, L., 2007. System dynamics modeling for soa project management. *IEEE International Conference on Service-Oriented Computing and Applications (SOCA'07).* 19-20 June 2007 California, IEEE, pp.286-294.

Krogdahl, P., Luef, G., Steindl, C., 2005. Service-oriented agility: an initial analysis for the use of agile methods for soa development. *IEEE International Conference on Services Computing*. 11-15 July 2005 Florida, IEEE, pp.93-100.

National Institutes of Health, Enterprise service bus (esb) pattern, 2006, https://enterprisearchitecture.nih.gov/Pages/EnterpriseServiceBusPattern.aspx [Erişim tarihi 7 Eylül 2013]

O'Brien, L., 2009. A framework for scope, cost and effort estimation for service oriented architecture (soa) projects. *Australian Software Engineering Conference.* 14-17 April 2009 Queensland, IEEE, pp. 101-110.

OMG, Integrating, Distributing applications via CORBA, Semaphore, http://www.omg.org/news/whitepapers/sema4corba.pdf [Erişim tarihi 13 Ekim 2013]

Oracle, Right from the Start: SOA Lifecycle Governance, An Oracle White Paper, 2012, http://www.oracle.com/us/technologies/soa/right-from-the-start-soa-1848279.pdf [Erişim tarihi 7 Eylül 2013]

Oracle, Oracle Service Bus Examples and Tutorials, 2010, https://java.net/downloads/oraclesoasuite11g/OSB/osb_examples_tutorials_111130.pdf [Erişim tarihi 7 Eylül 2013]

Oracle, Client Application Developer's Guide, Invoking Data Services Through Web Services, http://docs.oracle.com/cd/E13162_01/odsi/docs10gr3/appdev/wsdlclt.html [Erişim tarihi 7 Eylül 2013]

Oracle, Getting Started With WebLogic Web Services Using JAX-WS http://docs.oracle.com/cd/E15051_01/wls/docs103/webserv/index.html [Erişim tarihi 7 Eylül 2013]

Oracle, Client Application Developer's Guide, Invoking Data Services from Java Clients http://docs.oracle.com/cd/E13162_01/odsi/docs10gr3/appdev/ejbclt.html [Erişim tarihi 7 Eylül 2013]

Oracle, Client Application Developer's Guide, Data Programming Model and Update Framework http://docs.oracle.com/cd/E13162_01/odsi/docs10gr3/appdev/sdo.html [Erişim tarihi 7 Eylül 2013]

Özdemir, M., 2011. Soa ile ilgili çeşitli kavramlar. [online] http://ozdemirmurat.wordpress.com/2011/06/03/soa-ile-ilgili-cesitli-kavramlar/ [Erişim tarihi 14 Ekim 2013]

Rodriguez, J., ve Demsak, D., 2009. Lightweight soas: exploring patterns and principles of a new generation of soa solutions. *The Architecture Journal*, [online] December 2009, http://msdn.microsoft.com/de-de/bb426891.aspx [Erişim tarihi 5 Eylül 2013]

Sample XSD File: Customers and Orders. http://msdn.microsoft.com/en-us/library/bb675181.aspx [Erişim tarihi 9 Eylül 2013]

Smartbear, What is SoapUI. http://www.soapui.org/About-soapUI/what-is-soapui.html [Erişim tarihi 20 Ekim 2013]

TIBCO, TIBCO ActiveMatrix Service Bus Tutorial, 2013 https://docs.tibco.com/pub/activematrix_service_bus/3.3.0_september_2013/doc/pdf/tib_amx_administration_tutorials/tib_amx_administration_tutorials.pdf [Erişim tarihi 13 Ekim 2013]

TIBCO, TIBCO ActiveMatrix Service Bus Development Tutorials, 2012 https://docs.tibco.com/pub/activematrix_service_bus/3.2.0-august-2012/doc/pdf/tib_amx_development_tutorials/tib_amx_development_tutorials.pdf [Erişim tarihi 13 Ekim 2013]

EKLER

EK 1: Örnek COBOL Copybook

Örnek COBOL Copybook'u aşağıdaki kaynaktan alınmıştır.

https://code.google.com/p/cobolunit/source/browse/trunk/cobolunit+--username+hvaujour/COBOLUnit/CPY/CBUC0002.cpy?r=178

```
* COBOL UNIT CALLING INTERFACE FOR ASSERT PHASE
* FILE : CBUC0002[.CPY]
* THIS COPYBOOK DESCRIBES THE INFORMATION PROVIDED BY A USER
* PROGRAM TO COBOLUNIT ASSERT PROGRAM
01 CBU-ASSERT-CALL-INTERFACE.
* INPUT DATA
05 CBU-ASSERT PIC X(8) VALUE "CBUP0002".
05 CBU-ASSERT-NAME PIC X(20) VALUE SPACE.
05 CBU-ASSERT-TYPE PIC X(3) VALUE SPACE.
        88 CBU-ASSERT-EQUAL VALUE "=".
        88 CBU-ASSERT-NOT-EQUAL VALUE "<>".
        88 CBU-ASSERT-GT VALUE ">".
        88 CBU-ASSERT-GE VALUE ">=".
        88 CBU-ASSERT-LT VALUE "<".
        88 CBU-ASSERT-LE VALUE "<=".

05 CBU-ASSERT-DATA-TYPE PIC X(3).
        88 CBU-ASSERT-INT VALUE "INT".
        88 CBU-ASSERT-STRING VALUE "STR".
        88 CBU-ASSERT-DECIMAL VALUE "DEC".
        88 CBU-ASSERT-BOOLEAN VALUE "BOO".

05 CBU-ASSERT-EXPECTED-VAL PIC X(1024) VALUE SPACE.
05 CBU-ASSERT-EXPECTED-INT
REDEFINES CBU-ASSERT-EXPECTED-VAL PIC S9(9) COMP.
05 CBU-ASSERT-EXPECTED-DEC
REDEFINES CBU-ASSERT-EXPECTED-VAL PIC S9(9)V9(5) COMP-3.
```

```
05 CBU-ASSERT-EXPECTED-BOO
REDEFINES CBU-ASSERT-EXPECTED-VAL PIC X(5).
        88 CBU-ASSERT-EXPECTED-TRUE VALUE "TRUE".
        88 CBU-ASSERT-EXPECTED-FALSE VALUE "FALSE".

05 CBU-ASSERT-ACTUAL-VAL PIC X(1024) VALUE SPACE.
05 CBU-ASSERT-ACTUAL-INT
REDEFINES CBU-ASSERT-ACTUAL-VAL PIC S9(9) COMP.
05 CBU-ASSERT-ACTUAL-DEC
REDEFINES CBU-ASSERT-ACTUAL-VAL PIC S9(9)V9(5) COMP-3.
05 CBU-ASSERT-ACTUAL-BOO
REDEFINES CBU-ASSERT-ACTUAL-VAL PIC X(5).
        88 CBU-ASSERT-ACTUAL-TRUE VALUE "TRUE".
        88 CBU-ASSERT-ACTUAL-FALSE VALUE "FALSE".
* OUTPUT DATA
05 CBU-ASSERT-RETURN-CODE PIC X(1) VALUE SPACE.
        88 CBU-ASSERT-NO-ERROR VALUE SPACE.
        88 CBU-ASSERT-WARNING VALUE "W".
        88 CBU-ASSERT-ERROR VALUE "E".
        88 CBU-ASSERT-SEVERE VALUE "S".
05 CBU-ASSERT-MESSAGE PIC X(100) VALUE SPACE.
```

EK 2: Parçalı WSDL Örneği

(http://www.webservicex.net/geoipservice.asmx)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitionsxmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www.webservicex.net/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://www.webservicex.net/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault = "qualified" targetNamespace =
"http://www.webservicex.net/">
      <s:element name="GetGeoIP">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs = "0" maxOccurs = "1" name = "IPAddress" type = "s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name = "GetGeoIPResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs = "0" maxOccurs = "1" name = "GetGeoIPResult" type =
"tns:GeoIP"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="GeoIP">
        <s:sequence>
```

<s:element minOccurs="1" maxOccurs="1" name="ReturnCode" type="s:int"/>
<s:element minOccurs="0" maxOccurs="1" name="IP" type="s:string"/>
<s:element minOccurs = "0" maxOccurs = "1" name = "ReturnCodeDetails" type = "s:string"/>
<s:element minOccurs = "0" maxOccurs = "1" name = "CountryName" type = "s:string"/>
<s:element minOccurs = "0" maxOccurs = "1" name = "CountryCode" type = "s:string"/>
</s:sequence>
</s:complexType>
<s:element name="GetGeoIPContext">
<s:complexType/>
</s:element>
<s:element name="GetGeoIPContextResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs = "0" maxOccurs = "1" name = "GetGeoIPContextResult" type = "tns:GeoIP"/>
</s:sequence>
</s:complexType>
</s:element>
<s:element name="GeoIP" nillable="true" type="tns:GeoIP"/>
</s:schema>
</wsdl:types>
<wsdl:message name="GetGeoIPSoapIn">
<wsdl:part name="parameters" element="tns:GetGeoIP"/>
</wsdl:message>
<wsdl:message name="GetGeoIPSoapOut">
<wsdl:part name="parameters" element="tns:GetGeoIPResponse"/>
</wsdl:message>
<wsdl:message name="GetGeoIPContextSoapIn">
<wsdl:part name="parameters" element="tns:GetGeoIPContext"/>
</wsdl:message>

<wsdl:message name="GetGeoIPContextSoapOut">
<wsdl:part name="parameters" element="tns:GetGeoIPContextResponse"/>
</wsdl:message>
<wsdl:message name="GetGeoIPHttpGetIn">
<wsdl:part name="IPAddress" type="s:string"/>
</wsdl:message>
<wsdl:message name="GetGeoIPHttpGetOut">
<wsdl:part name="Body" element="tns:GeoIP"/>
</wsdl:message>
<wsdl:message name="GetGeoIPContextHttpGetIn"/>
<wsdl:message name="GetGeoIPContextHttpGetOut">
<wsdl:part name="Body" element="tns:GeoIP"/>
</wsdl:message>
<wsdl:message name="GetGeoIPHttpPostIn">
<wsdl:part name="IPAddress" type="s:string"/>
</wsdl:message>
<wsdl:message name="GetGeoIPHttpPostOut">
<wsdl:part name="Body" element="tns:GeoIP"/>
</wsdl:message>
<wsdl:message name="GetGeoIPContextHttpPostIn"/>
<wsdl:message name="GetGeoIPContextHttpPostOut">
<wsdl:part name="Body" element="tns:GeoIP"/>
</wsdl:message>
<wsdl:portType name="GeoIPServiceSoap">
<wsdl:operation name="GetGeoIP">
<wsdl:documentation xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"> GeoIPService - GetGeoIP enables you to easily look up countries by IP addresses </wsdl:documentation>
<wsdl:input message="tns:GetGeoIPSoapIn"/>
<wsdl:output message="tns:GetGeoIPSoapOut"/>
</wsdl:operation>
<wsdl:operation name="GetGeoIPContext">
<wsdl:documentation xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"> GeoIPService - GetGeoIPContext enables you to easily look up countries by Context </wsdl:documentation>

<wsdl:input message="tns:GetGeoIPContextSoapIn"/>
<wsdl:output message="tns:GetGeoIPContextSoapOut"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="GeoIPServiceHttpGet">
<wsdl:operation name="GetGeoIP">
<wsdl:documentation xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"> GeoIPService - GetGeoIP enables you to easily look up countries by IP addresses </wsdl:documentation>
<wsdl:input message="tns:GetGeoIPHttpGetIn"/>
<wsdl:output message="tns:GetGeoIPHttpGetOut"/>
</wsdl:operation>
<wsdl:operation name="GetGeoIPContext">
<wsdl:documentation xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"> GeoIPService - GetGeoIPContext enables you to easily look up countries by Context </wsdl:documentation>
<wsdl:input message="tns:GetGeoIPContextHttpGetIn"/>
<wsdl:output message="tns:GetGeoIPContextHttpGetOut"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="GeoIPServiceHttpPost">
<wsdl:operation name="GetGeoIP">
<wsdl:documentation xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"> GeoIPService - GetGeoIP enables you to easily look up countries by IP addresses </wsdl:documentation>
<wsdl:input message="tns:GetGeoIPHttpPostIn"/>
<wsdl:output message="tns:GetGeoIPHttpPostOut"/>
</wsdl:operation>
<wsdl:operation name="GetGeoIPContext">
<wsdl:documentation xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"> GeoIPService - GetGeoIPContext enables you to easily look up countries by Context </wsdl:documentation>
<wsdl:input message="tns:GetGeoIPContextHttpPostIn"/>
<wsdl:output message="tns:GetGeoIPContextHttpPostOut"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="GeoIPServiceSoap" type="tns:GeoIPServiceSoap">

<soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="GetGeoIP">
<soap:operation soapAction = "http://www.webservicex.net/GetGeoIP" style = "document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetGeoIPContext">
<soap:operation soapAction = "http://www.webservicex.net/GetGeoIPContext" style = "document"/>
<wsdl:input>
<soap:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:binding name="GeoIPServiceSoap12" type="tns:GeoIPServiceSoap">
<soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="GetGeoIP">
<soap12:operation soapAction = "http://www.webservicex.net/GetGeoIP" style = "document"/>
<wsdl:input>
<soap12:body use="literal"/>
</wsdl:input>
<wsdl:output>
<soap12:body use="literal"/>
</wsdl:output>

```
    </wsdl:operation>
    <wsdl:operation name="GetGeoIPContext">
      <soap12:operation soapAction = "http://www.webservicex.net/GetGeoIPContext" style =
"document"/>
      <wsdl:input>
        <soap12:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name= "GeoIPServiceHttpGet" type= "tns:GeoIPServiceHttpGet">
    <http:binding verb="GET"/>
    <wsdl:operation name="GetGeoIP">
      <http:operation location="/GetGeoIP"/>
      <wsdl:input>
        <http:urlEncoded/>
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetGeoIPContext">
      <http:operation location="/GetGeoIPContext"/>
      <wsdl:input>
        <http:urlEncoded/>
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
```

```
<wsdl:binding name= "GeoIPServiceHttpPost" type= "tns:GeoIPServiceHttpPost">
  <http:binding verb="POST"/>
  <wsdl:operation name="GetGeoIP">
    <http:operation location="/GetGeoIP"/>
    <wsdl:input>
      <mime:content type="application/x-www-form-urlencoded"/>
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetGeoIPContext">
    <http:operation location="/GetGeoIPContext"/>
    <wsdl:input>
      <mime:content type="application/x-www-form-urlencoded"/>
    </wsdl:input>
    <wsdl:output>
      <mime:mimeXml part="Body"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="GeoIPService">
  <wsdl:port name="GeoIPServiceSoap" binding="tns:GeoIPServiceSoap">
    <soap:address location="http://www.webservicex.net/geoipservice.asmx"/>
  </wsdl:port>
  <wsdl:port name="GeoIPServiceSoap12" binding="tns:GeoIPServiceSoap12">
    <soap12:address location="http://www.webservicex.net/geoipservice.asmx"/>
  </wsdl:port>
  <wsdl:port name="GeoIPServiceHttpGet" binding="tns:GeoIPServiceHttpGet">
    <http:address location="http://www.webservicex.net/geoipservice.asmx"/>
  </wsdl:port>
  <wsdl:port name="GeoIPServiceHttpPost" binding="tns:GeoIPServiceHttpPost">
    <http:address location="http://www.webservicex.net/geoipservice.asmx"/>
```

```
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

EK 3: CustomerOrder Xsd Dosyası Örneği

[http://msdn.microsoft.com/en-us/library/bb675181.aspx]

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name='Root'>
    <xs:complexType>
      <xs:sequence>
        <xs:element name='Customers'>
          <xs:complexType>
            <xs:sequence>
              <xs:element name = 'Customer' type = 'CustomerType' minOccurs = '0' maxOccurs = 'unbounded' />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name='Orders'>
          <xs:complexType>
            <xs:sequence>
              <xs:element name = 'Order' type = 'OrderType' minOccurs = '0' maxOccurs = 'unbounded' />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
```

```
  <xs:key name='CustomerIDKey'>
    <xs:selector xpath='Customers/Customer'/>
    <xs:field xpath='@CustomerID'/>
  </xs:key>
  <xs:keyref name='CustomerIDKeyRef' refer='CustomerIDKey'>
    <xs:selector xpath='Orders/Order'/>
    <xs:field xpath='CustomerID'/>
  </xs:keyref>
</xs:element>
<xs:complexType name='CustomerType'>
  <xs:sequence>
    <xs:element name='CompanyName' type='xs:string'/>
    <xs:element name='ContactName' type='xs:string'/>
    <xs:element name='ContactTitle' type='xs:string'/>
    <xs:element name='Phone' type='xs:string'/>
    <xs:element name='Fax' minOccurs='0' type='xs:string'/>
    <xs:element name='FullAddress' type='AddressType'/>
  </xs:sequence>
  <xs:attribute name='CustomerID' type='xs:token'/>
</xs:complexType>
<xs:complexType name='AddressType'>
  <xs:sequence>
    <xs:element name='Address' type='xs:string'/>
    <xs:element name='City' type='xs:string'/>
    <xs:element name='Region' type='xs:string'/>
```

```
    <xs:element name='PostalCode' type='xs:string' />
    <xs:element name='Country' type='xs:string'/>
  </xs:sequence>
  <xs:attribute name='CustomerID' type='xs:token'/>
</xs:complexType>
<xs:complexType name='OrderType'>
  <xs:sequence>
    <xs:element name='CustomerID' type='xs:token'/>
    <xs:element name='EmployeeID' type='xs:token'/>
    <xs:element name='OrderDate' type='xs:dateTime'/>
    <xs:element name='RequiredDate' type='xs:dateTime'/>
    <xs:element name='ShipInfo' type='ShipInfoType'/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name='ShipInfoType'>
  <xs:sequence>
    <xs:element name='ShipVia' type='xs:integer'/>
    <xs:element name='Freight' type='xs:decimal'/>
    <xs:element name='ShipName' type='xs:string'/>
    <xs:element name='ShipAddress' type='xs:string'/>
    <xs:element name='ShipCity' type='xs:string'/>
    <xs:element name='ShipRegion' type='xs:string'/>
    <xs:element name='ShipPostalCode' type='xs:string'/>
    <xs:element name='ShipCountry' type='xs:string'/>
  </xs:sequence>
```

```
    <xs:attribute name='ShippedDate' type='xs:dateTime'/>
  </xs:complexType>
</xs:schema>
```

ÖZGEÇMİŞ

Adı Soyadı : Okan PULUKÇU

Sürekli Adresi : Cihangir Mah. Deryadil Sok. Efe Apt. No:29/6 Avcılar-İstanbul

Doğum Yeri ve Yılı : Bakırköy/1985

Yabancı Dili : İngilizce

İlk Öğretim : Avcılar Abdülkadir Uztürk İlköğretim Okulu - 1999

Orta Öğretim : Küçükçekmece Halkalı Anadolu TML - 2003

Lisans : İstanbul Kültür Üniversitesi/Bilgisayar Mühendisliği - 2009

Yüksek Lisans : Bahçeşehir Üniversitesi - 2014

Enstitü Adı : Fen Bilimleri Enstitüsü

Program Adı : Bilgi Teknolojileri (Türkçe, Tezli)

Çalışma Hayatı :

Akbank (2012 - ...)

Görevi: Yazılım Uzmanı
Açıklama:
BT Altyapı Uygulamaları, Çoklu Kanal Mimarisi ekibinde yazılım uzmanı olarak TIBCO EMS, TIBCO BusinessWorks Designer, TIBCO Administrator ürünleri kullanarak entegrasyon servislerinin geliştirilmesi, mevcut servislerin bakım işleri ve Facebook Bankacılığı ve Web Bankacılığı gibi yeni alternatif dağıtım kanalları projelerinde entegrasyon katmanının altyapısının tasarlanmasından sorumluyum.

Ayrıca; custom Java class'larının geliştirilmesi ve bu kodların genel mimaride yer alan TIBCO BW servislerinde kullanmasından, Oracle DB üzerinde günlük ve arşiv loglama işlerinin takip edilmesinden, Çoklu Kanal Mimarisi için birçok Java uygulaması geliştirerek entegrasyon servisi geliştirme yaşam döngüsüne katkıda bulunmaktan sorumluyum.

Çoklu Kanal Mimarisi framework'ünde kullanılan veya proje bazlı olarak geliştirilmesine ihtiyaç duyulan, Java teknolojileri ile geliştirdiğim yazılımlar arasında;
Tüm servis requestlerinde kart numaralarının maskelenerek loglanmasını sağlayan yazılım, büyük çaptaki projelerin parçalanarak bölünmesini sağlayan yazılım, MBB Yasal Zorunluluk Projesi kapsamında 250 civarı process'de yapılacak değişikliklerin toplu olarak yapılmasını sağlayan yazılım sayılabilir.

Çoklu Kanal Mimarisi Yönetim ekranlarını RAD, JSF, JSP kullanarak geliştirmekte ve mevcut ekranların bakımlarını yapmaktayım.

Kullandığım teknolojiler arasında; TIBCO EMS, TIBCO BW Designer, TIBCO Administrator, Java, Eclipse, Oracle 11g, SQL, IBM ClearCase, Team Foundation Server (TFS) 2013 sayılabilinir.

Şekerbank T.A.Ş. (2011 - 2012)

Görevi: Analist Yazılımcı
Açıklama:
Şekerbank'daki analist yazılımcı görevimde; Aurora Framework'ü üzerinde, JavaEE, Swing, EBML, PomFactory, iBatis, jBoss ve Oracle 11g kullanarak; kart basım ve kurye modüllerinde yazılım geliştirilmesinden, bu modüllerde olan bakım işlerinin yapılmasından sorumluydum.

Yer aldığım projelerden; GMS migration projesinde, Garanti Bankası tarafından yönetilen POS veritabanının in-house'a çekilmesi amacıyla PL-SQL ile stored procedureler yazdım. Bankalararası Kart Merkezi'nin projesi olan Merkezi POS Veritabanı projesi için, MS-SQL ve Oracle veritabanlarından gerekli verileri alan ve istenen datawarehouse'u oluşturan yazılımı Java ile geliştirdim.

Ayrıca, talep yönetim sistemi olarak JIRA, süreç yönetim sistemi olarak Sesis, versiyonlama aracı olarak da SVN kullandım.

SoftTech (İş Bankası) (2010 - 2011)

Görevi: Yazılım Mühendisi
Açıklama:
Bilgi Teknolojileri Çözüm Geliştirme departmanı Kredi Kartları proje ekibinde Yazılım Uzmanı olarak, COBOL ile IBM Mainframe üzerinde IMS programlarının yazılmasından ve üye işyerleri için günlük ekstre üreten ve FTP yapan programların Java ile programlanmasından ve schedule edilmesinden sorumluydum.

SoftTech A.Ş.'de çalıştığım bir yılın ardından aldığım terfi sonrasında Yazılım Mühendisi ünvanına getirildim. Görevlendirildiğim debit/credit kart paketi projesi kapsamında; elektronik journal, printing servislerinin sağlanması ve IMS ile .Net platformunun (Ocean) entegrasyonu amacıyla; entegrasyon katmanında WebSphere Integration Developer aracı ile IBM Enterprise Service Bus mediation servislerinin WSDL'larının tasarlanmasından, çeşitli primitiveler, Java, XML, Xpath ve custom XSLT'ler kullanarak generic ve specific mediationların geliştirilmesinden, printing servisleri ile birlikte JasperReports ile dekont hazırlanmasından, SoapUI ile birim ve mock testlerinin yapılmasından, ayrıca UAT ve entegrasyon testlerinin yapılmasından ve servislerin ilgili ortamlara kurulumunun yapılmasından sorumluydum.

IBM'in versiyonlama ve sürümleme ürünleri olan ClearCase ve ClearQuest'i, raporlama ürünü olan RPM'i, talep yönetim sistemi olan MAXIMO'yu ve HP'nin test süreç yönetim aracı olan Quality Center'ı kullandım.