EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

KURUMSAL SERVİS KANALI TABANLI VERİ BÜTÜNLEŞTİRME MİMARİSİNİN GERÇEKLEŞTİRİMİ

Beril KAVAKLI

Tez Danışmanı: Doç. Dr. Murat Osman ÜNALIR

Bilgisayar Mühendisliği Anabilim Dalı

Sunuş Tarihi: 04.07.2017

Bornova-İZMİR

2017

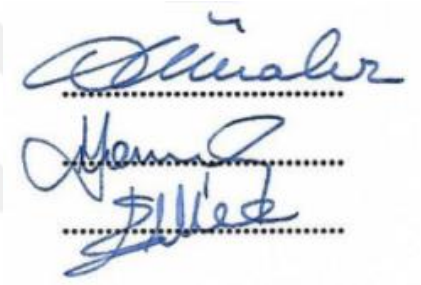
Beril KAVAKLI tarafından Yüksek Lisans tezi olarak sunulan "**Kurumsal Servis Kanalı Tabanlı Veri Bütünleştirme Mimarisinin Gerçekleştirimi**" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 04.07.2017 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri: **İmza**

Jüri Başkanı : Doç. Dr. Murat Osman ÜNALIR

Raportör Üye: Doç. Dr. Murat KOMESLİ

Üye : Yrd. Doç. Dr. Emine SEZER



EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi olarak sunduğum "**Kurumsal Servis Kanalı Tabanlı Veri Bütünleştirme Mimarisinin Gerçekleştirimi**" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

04/07/2017

Beril KAVAKLI

ÖZET

KURUMSAL SERVİS KANALI TABANLI VERİ BÜTÜNLEŞTİRME MİMARİSİNİN GERÇEKLEŞTİRİMİ

KAVAKLI, Beril

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı
Tez Yöneticisi: Doç. Dr. Murat Osman ÜNALIR
Temmuz 2017, 65 sayfa

Günümüz iş dünyasında birçok farklı alanda süreçlerin takibi, o alana özel geliştirilmiş yazılım ürünleri kullanılarak yapılmaktadır. Şirket yapıları ele alındığında birçok farklı birim içerdikleri gözlemlenebilir. Birimler belli oranda kendi içlerinde bağımsız olsalar da birbirleriyle iletişim ihtiyacı duymaktadırlar. Bu iletişim ihtiyacı birimlerde kullanılan yazılım ürünleri için de son derece önemlidir. Örneğin; müşteri ilişkileri biriminde ilgili yazılım uygulaması üzerinde oluşturulan bir müşteri kaydının, muhasebe biriminde kullanılan yazılım uygulamasında da otomatik oluşturulması beklenmektedir. Bu kapsamda birçok ortak kavram için bütünleştirme gereksinimi karşımıza çıkmaktadır.

Bu gereksinimler göz önünde bulundurularak bu tez kapsamında bir yazılım şirketi bünyesindeki yazılım ürünlerinin birbirileriyle olan iletişim ihtiyacı ve mevcut koşullarda kullanılan bütünleştirme yapısı incelenmiş ve eksik noktaları araştırılmıştır. Mevcut bütünleştirme sistemlerinin getirdiği zorlukları ortadan kaldırmak için kurumsal servis kanalı mimarisinde bir model önerilmiştir. Önerilen model, yazılımsal olarak da prototip haline getirilerek gerçeklenmiştir.

Anahtar sözcükler: Kurumsal Servis Kanalı, Servis Odaklı Mimari, Veri Bütünleştirme, Veri Senkronizasyonu

ABSTRACT

IMPLEMENTATION OF

ENTERPRISE SERVICE BUS BASED

DATA INTEGRATION ARCHITECTURE

KAVAKLI, Beril

MSc. in Computer Engineering
Supervisor: Doç. Dr. Murat Osman ÜNALIR
July 2017, 65 pages

In today's world, business processes on various fields are handled by taking the advantages of custom software products which are specially developed for that fields. When corporate hierarchies are considered, it is possible to observe that there are many different departments. Although they are independent, these departments need to communicate with each other in order to achieve corporate's common goals. This communication need is also very important for software products used in departments. For example; A customer record, which is created by the software of customer-relations department, must automatically be accessible by the software of accounting department. In this context, the need for integration for many common concepts emerges.

Taking these requirements into consideration, in this thesis, inter-product communication needs and existing integration methodology of a software development company are investigated and problematic parts are identified. In order to eliminate the difficulties of existing integration methodology; a model, which is based on enterprise service bus architecture, is proposed and implemented.

Keywods: Enterprise Service Bus, Service Oriented Architecture, Data Integration, Data Synchronization

TEŞEKKÜR

Lisans eğitim yıllarımdan başlamak üzere deneyimleri ve önerileriyle her zaman desteğini gördüğüm, bu tez çalışması süresince de elinden gelen yardımı sağlayan danışmanım Ege Üniversitesi Bilgisayar Mühendisliği öğretim üyesi Sayın Doç. Dr. Murat Osman ÜNALIR'a desteği ve hoşgörüsü için teşekkür ederim.

Tezin bitiş aşamasında bana verdiği değerli katkıları için Yaşar Üniversitesi Yazılım Mühendisliği öğretim üyesi Sayın Doç. Dr. Murat KOMESLİ'ye teşekkür ederim.

Lisans eğitimim sırasında tanıdığım ve yıllar içinde her türlü konuda destek ve yardımını benden esirgemeyen Ege Üniversitesi Bilgisayar Mühendisliği öğretim üyesi Sayın Yrd. Doç. Dr. Emine SEZER'e teşekkür ederim.

Yıllar içinde bütün güzel günlerde olduğu gibi zor günlerimde de elimi hiç bırakmayan, desteğini her zaman kalbimde hissettiğim en iyi arkadaşım, meslektaşım ve sevgili eşim Tunç GÜLTEKİN'e varlığı ve her zaman yanımda olduğu için teşekkür ederim.

Bu günlere gelmemdeki en önemli yapıtaşlarından, her zaman bana güvenerek destekleyen sevgili annem Cahide KAVAKLI'ya ve sevgili babam Ahmet KAVAKLI'ya çok teşekkür ederim.

Yüksek lisans eğitimim boyunca göstermiş oldukları hoşgörü için Logo Yazılım'a ve sevgili ekip liderim Utku CEVRE'ye teşekkür ederim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

ÇİZELGELER DİZİNİ

Çizelge Sayfa

KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
API	Application Programming Interface
B2B	Business to Business
CORBA	Common Object Request Broker Architecture
CRM	Customer Relationship Management
DIM	Data Integration Model
EAI	Enterprise Application Integration
EDA	Event Driven Development
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
ETL	Export Transform Load
HRM	Human Resources Management
HTTP	Hyper-Text Transfer Protocol
JSON	Javascript Object Notation
MVC	Model View Controller
REST	Representational State Transfer
RMM	Relationship Management Methodology
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol

KISALTMALAR DİZİNİ (devam)

Kısaltmalar	Açıklama
UDDI	Universal Description Discovery and Integration
URI	Uniform Resource Identifier
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

1. GİRİŞ

Birçok alanda iş süreçlerinin yazılım uygulamalarıyla takip edilmeye başlanmasıyla birlikte iş dünyasında yazılımlara ayrılan bütçe ve gösterilen ilgi gitgide artmaktadır. Öncesinde manuel olarak kağıtlarda, tablolama dosyalarında, birbirinden bağımsız formlarda takip edilen süreçler hızla kendini modern yazılımlara bırakmaktadır.

Bir işletme çok yönlü olarak incelenecek olursa, birçok farklı birimden oluştuğu görülmektedir. İşletmedeki çalışanların bilgileri, izin hak edişleri, maaş bordroları vs. takip edilmesi gereken verilerdir. Üretim, satış, kaynak planlama, muhasebe gibi bölümler işletmenin bel kemiğini oluşturacak bilgileri barındırmaktadır. Bunların yanında müşteri bilgileri de ciddi önem taşımaktadır. Müşterinin iletişim bilgileri, adresi, daha önceki satışlar, potansiyel fırsatlar firmanın takip etmek isteyeceği bilgilerdir. Bahsedilen her alana özel bugüne kadar pek çok uygulama geliştirilmiştir. Ancak uygulamalar arasında birbirinden bağımsız veri çoklanması problemleri ciddi sıkıntılar yaratabilmektedir.

İşletmenin bölümlerini insan beyninin bölümlerine benzetirsek, birbiriyle iletişime geçmeden işlerin yürümesi imkansızdır. Bu bağlamda uygulamalar arası iletişim ve veri paylaşımı da üst düzeyde önem arz etmektedir. Bu tez kapsamında da gerçek hayatta karşılaşılan uygulamalar arası iletişimsizlik probleminin kurumsal servis kanalı mimarisi kullanılarak çözülmesi amaçlanmıştır.

1.1 Tez Kapsamı

Tezin ikinci bölümünde, bu çalışma süresince yararlanılan ve temel alınan kavramların açıklamalarına yer verilmektedir. Üçüncü bölümde, ilgili çalışmalara değinilmektedir. Dördüncü bölümde, bu çalışmanın yapılmasına sebep olan problem detaylı şekilde açıklanmaktadır. Beşinci bölümde, problemin çözümü için önerilen mimari model açıklanmakta, yararlanılan ve kullanılan mimari desenlerden bahsedilmektedir. Altıncı bölümde, tez kapsamında yazılmış olan uygulamanın detaylarına değinilmekte, kullanılan teknolojiler belirtilmektedir. Yedinci bölümde, önerilen bütünleştirme modeli ile iki alternatif bütünleştirme modeli karşılaştırılarak, avantaj ve dezavantajları tartışılmaktadır. Sekizinci bölümde, bu çalışma sonrasında odaklanılabilecek ve çalışmayı geliştirebilecek noktalar açıklanmaktadır. Dokuzuncu ve son bölümde, çalışmadan elde edilen sonuçlara yer verilmektedir.

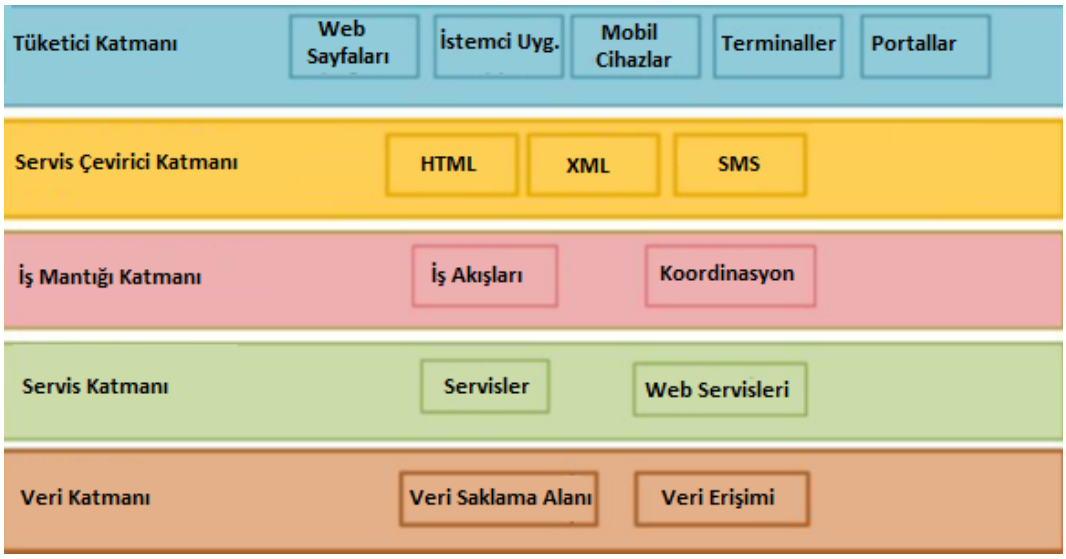
2. TEMEL KAVRAMLAR

Bu tez kapsamında kullanılacak olan kurumsal servis kanalı mimarisi, farklı uygulamaların birbiri ile iletişim kurabilmesi için kullanılan bir uygulama mimarisidir. Kurumsal servis kanalı, altyapısında servis odaklı yazılım mimarisini kullanmaktadır. Bu bölümde tez boyunca yararlanılacak kavramlar olan servis odaklı mimari ve kurumsal servis kanalı kavramlarının açıklamalarına yer verilmiştir.

2.1 Servis Odaklı Mimari (Service Oriented Architecture - SOA)

Servis odaklı mimari, servis tabanlı birleştirici bir mimaridir. Servisler bir araya gelerek iş akışını oluşturur ve böylece uygulamalar gerçekleştirilmiş olur. Bu mimaride servisler, yeniden kullanılabilir ve aynı zamanda farklı uygulamalar tarafından çağırılabilecek iş parçacıklarıdır. Her bir küçük iş parçacığının birbirinden bağımsız olması, ana sistemi etkilemeden parçaların kolayca değiştirilebilmesine olanak sağlamaktadır. Servis odaklı mimari, dağıtık sistemlerin ortaya çıkışıyla gelişmiş bir mimari olduğundan, istenirse her bir servis farklı sunucular üzerinde konumlandırılabilir.

Servis odaklı mimari, uygulama katmanlarını farklı servislere bölüp, sunucular üzerinden kullanmaya imkan sağlayan bir mimaridir. Şekil 2.1'de farklı katmanların servislere ayrılması modellenmiştir.



Şekil 2.1 Servis Odaklı Katmanlı Yapı

Yinelenen işlerin servis olarak gerçekleştirilmesi, günlük hayattaki birçok işte teknoloji bağımlılığından kurtulmak anlamına gelmektedir. Örneğin banka şubelerinde müşteri sorgulama işlemi servis mantığına çok uygundur. Temelde müşteri kimlik numarasını alıp, cevap olarak müşteriyle ilgili detaylı bilgiyi getiren atomik bir yapıdadır.

Servis odaklılık, iş uygulamalarının küçük iş servislerinin bir araya getirilerek oluşturulmasıdır. Bu noktada önemli olan kullanılacak olan teknoloji değil, iş mantığını kurmaktır. Servis odaklı mimari, servis odaklı iş mantığının gerçeğe uygulanmasını sağlayan bilgi teknolojileri mimarisidir.

Servis odaklı mimaride, iş fonksiyonlarının birlikte çalışmayı destekleyecek esneklikte, yeniden kullanılabilir şekilde, iyi tanımlanmış ve gevşek bağlı bileşenler halinde oluşturulması amaçlanmaktadır.

İş süreçleri açısından bakıldığında, süreçler ve ihtiyaçlar değişime açık yapılardır. İhtiyaçlar değiştiğinde programın, yeni ihtiyaçları karşılayacak şekilde değişmesi gerekmektedir. Bu da belirli bir zaman ve maliyet demektir. Bu açıdan bakıldığında iş uygulamalarının yeni ihtiyaçlara hızlı cevap verebilecek esneklikte olması gerekir. Bu servis odaklı mimari ile oluşturulabilir.

2.1.1 Servis odaklı mimari özellikleri

Mimarinin genel olarak başlıca özellikleri aşağıdaki şekildedir:

Esneklik: İş fonksiyonlarının küçük servisler halinde yazılması yapıya büyük bir esneklik sağlamaktadır. Değişiklik gerektiğinde sadece ilgili servisler düzenlenerek ihtiyaç giderilmiş olacaktır.

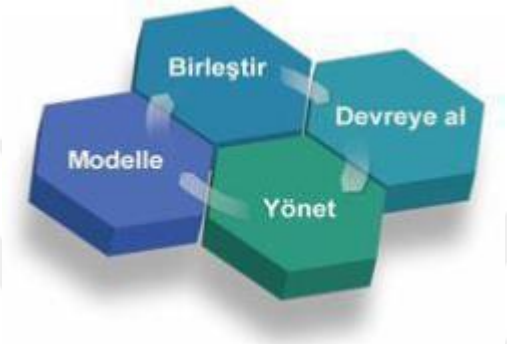
Yeniden Kullanılabilirlik: Oluşturulan küçük servisler istenen uygulama üzerinden kullanılabildiği için yeniden kullanılabilirlik üst seviyelere taşınmış olmaktadır.

İyi Tanımlama: Her servisin iyi tanımlanmış bir arayüzü bulunmaktadır ve bu arayüz sayesinde servisin ne içerdiği, nasıl kullanılacağı kolayca anlaşılabilmektedir. Arayüzler belli bir standarta sahip olduğundan bütün platform ve sistemler tarafından kullanılabilir. WSDL (Web Services Description Language) buna örnek olarak gösterilebilir.

Gevşek Bağlılık: Servisler iş akışı içerisinde gevşek bağlarla birbirine bağlanırlar. Bunun sebebi ise gerektiğinde servislerin birbirlerinden ayrılıp başka yapılarla birleştirilmesinin kolay hale getirilmesidir.

2.1.2. Servis odaklı mimari yaşam döngüsü

Servis odaklı mimari yaşam döngüsü kavramı, bu mimariyi temel alan uygulamalar için tanımlı dört ana parçadan oluşmaktadır. Bu parçalar; Şekil 2.2'de görüldüğü gibi; modelleme, birleştirme, devreye alma ve yönetmedir.



Şekil 2.2 Servis Odaklı Mimari Yaşam Döngüsü

Modelleme:

Bu evrede gereksinimler belirlenerek, hedefteki iş süreci ve süreci oluşturacak parçalar modellenir. Oluşturulan modelin hedef iş sürecine uygunluğu benzetim (simüle) yapılarak ölçülür. Bu evrede analistler etkin rol oynamaktadır. Analistler gerekli teknik bilgiye sahip, işin aşamalarını çok iyi bilen kişiler olmalıdır.

Birleştirme:

Bu evrede üretilen modeller ortak bir format kullanılarak yazılım grubuna aktarılır. Yazılım grubu, bu modelleri hayata geçirecek kodları yazarak servisleri oluştururlar. Yazılımcıların etkin rol aldıkları evredir.

Devreye Alma:

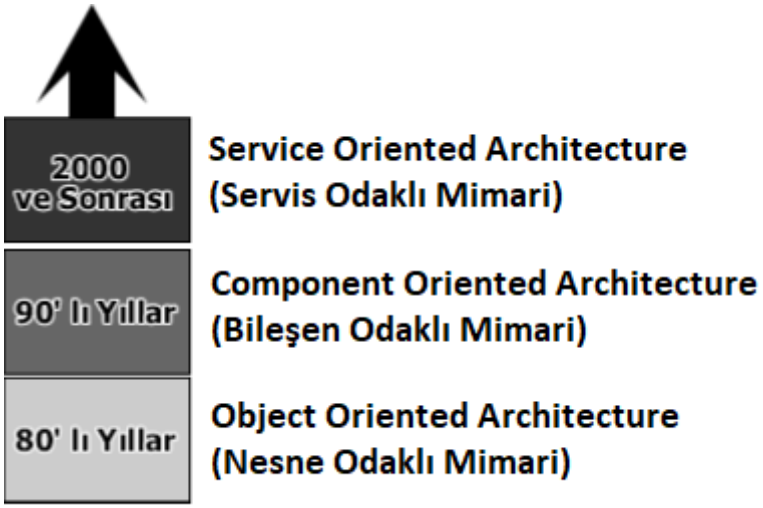
Birleştirme aşaması bitiminde elde edilen aktif servisler ve süreçler çalıştırılır ve kullanıcı deneyimine sunulur.

Yönetim:

Devreye alınan uygulama çeşitli parametreler ele alınarak izlenip, değerlendirilmektedir. Eğer olumsuz bir durum söz konusuysa toplanan sonuç verileri bir sonraki çevrim için dikkate alınacak maddeleri oluşturur. Bu maddeler ışığında modellemede değişiklik yapılabilir veya yeni bir çevrim başlatılabilir. Bu aşamada analistler ve yazılımcılar birlikte çalışmaktadırlar.

2.1.3 Tarihçe

Şekil 2.3'te görüldüğü gibi, 70'li yıllarda programlama sektörü, fonksiyonel programlama ile yürümekteydi. 80'li yıllara gelindiğinde ise, fonksiyonel programlamanın esneklik ve yeniden kullanılabilirlik problemleri programlama sektörünü nesne odaklı programlamaya yönlendirmiştir. Nesne odaklı programlama, tek uygulama içerisinde yeniden kullanılabilirlik sorunlarını çözse de, aynı kodun farklı uygulamalar tarafından kullanılabilirliği sorununa beklendiği şekilde çözüm getirememiştir (Wang et al., 2004). Bu sorun ise, bileşen odaklı mimari ile aşılmıştır. Bunun sonucu olarak da dağıtık bileşen odaklı mimariler doğmuştur. Aynı zamanda kod yine bir kez yazılmış ancak birden fazla kopya oluşması sorunu da aşılmıştır. Bu sayede belirli bir yerde duran kod parçacığı farklı uygulamalar tarafından kullanılabilir hale gelmiştir. Bu durum, uygulama sunucularının doğmasına sebep olmuştur. Uygulama sunucusu üzerindeki kod parçaları, platform aynı olduğu sürece sorunsuz bir şekilde çalışmaktadır. Ancak, farklı platformlarda çalışan uygulamalar söz konusu olduğunda yine bir sorunla karşı karşıya kalınmıştır. Bu noktada servis odaklı mimarinin gelişmesi ile platform farklılıklarından kaynaklanan sorunlar giderilmiştir.



Şekil 2.3 Mimari Gelişim Tarihi

2.1.4. Avantajları

Servis odaklı mimarinin uygulanması ile birlikte aşağıda maddeler halinde verilen avantajlar elde edilmiştir:

- Servis odaklı mimari, yapı itibariyle diğer sistemlerle kıyaslandığında, servis sağlayıcı için en güvenli yapıdır.

- Yapı itibariyle dağıtık olduğundan ölçeklenebilir yazılımlar elde edilebilir.

- Sürekliliği ve mantık katmanını büyük ölçüde servis sağlayıcıya bırakmaktadır. Böylelikle geliştirici için uygulama tek katmana indirgenmiş olur.

- Servis ve uygulama katmanı genellikle farklı sunucularda yer alır. Böylece bağımlılık azaltılmış olur.

- Servis odaklı mimariye sahip uygulamalar aynı katmanları aynı servis üzerinden kullanabilmektedirler. Serviste bir değişiklik ihtiyacı doğduğunda istemci uygulamalarda hiç değişiklik yapmaksızın tek bir servis üzerinde gerekli değişiklikler yapılarak güncelleme tamamlanmış olur. Böylece bakım ve onarım maliyetleri düşer.

- Bir masaüstü uygulaması geliştirileceğinde, istemci tarafında sadece arayüze ihtiyaç duyulacağından, geliştirme işlemi soyutlaşmaktadır.

- Farklı platformlarda çalışan uygulamalarda bile aynı altyapı kullanılabilir.

- Uygulamanın farklı katmanları farklı sunucular üzerinde yer alabilir. Böylece uygulamanın performansı artarken, yönetim maliyeti de azalmış olur.

- Hazırlanan servis veya uygulama için API (Application Programming Interface – Uygulama Programlama Arayüzü) oluşturularak, başka uygulamalar tarafından kullanılmasını sağlanabilir. Böylece bütünleştirme kolaylaştırılmış olmaktadır.

- Bulut teknolojisini kullanan uygulamalar gerçekleştirilebilir.

- Uygulama için oluşturulan servisler başka uygulamalarda da çok kolay şekilde kullanılabildiğinden yeniden kullanılabilirlik arttırılmış olur.

- Servis odaklı mimarinin esnek yapı özelliği sayesinde değişen iş ihtiyaçlarına çok daha hızlı şekilde yanıt verilebilir.

- Artırımlı geliştirmeyi benimsediğinden bir sorun halinde geri dönüş hızlı olduğundan risk azaltılmış olur.

- İş süreçlerini iyileştirerek iş birliğini arttırır.

2.1.5. Servis türleri

Hizmet Servisleri: Birçok iş akışı tarafından kullanılabilecek genel geçer araçları ve özellikleri sunan servislere hizmet servisleri denilmektedir. Hız birimi çevirici, döviz kuru dönüştürücü vb. servisler bu türdendir.

İş Servisleri: Kurumun iş akışı içerisinde tek bir işi/fonksiyonu gerçekleştiren servisler bu kategoride yer almaktadır.

Bir üniversitedeki öğrenci bilgi sisteminde öğrencinin ilk kaydını yapma işini gerçekleştiren servis bu tip servisler için bir örnektir.

Koordinasyon Servisleri: Özellikle karmaşık iş süreçlerinde, iş akışı birden fazla koldan devam edebilmekte, iş akışında birden fazla servis görev alabilmekte ve akışın durumuna bağlı olarak bu servislerin hepsi değil belirli bir kısmı çalışabilmekte ve aynı zamanda bunu farklı sıralarda yapabilmektedirler.

Koordinasyon servisleri bu akışın süreç içerisinde ve doğru sırayla devam etmesi için iş akışını koordine ederler.

2.1.6 Servis geliştirim adımları

Servis Gereksinimlerini Belirlemek: Bu aşamada fonksiyonel ve fonksiyonel olmayan gereksinimler belirlenmektedir. Servisin tam olarak ne iş yapacağı, hangi girdileri alıp, hangi çıktıları üretecekleri netleştirilir.

Bir servisin gereksinimlerine karar verilirken aşağıdaki sorulara cevap aranmalıdır.

- Servis sadece tek bir iş akışında mı kullanılacak, yoksa birden fazla iş akışında mı kullanılacak? (Bu bilgi aynı zamanda, servis üzerinde değişiklik yapılacağında veya servis kaldırılacağında bağımlılıkları takip etmek için gerekli olan kritik bir bilgidir.)

- Servis iş akışı bağımsız mıdır? (Servisin bağımsız olup olmadığına karar vermek, servisin uygulanması sırasını, hızını ve servisle ilgili gereksinimleri etkileyecektir.)

- Servis durum (state) tutmak zorunda mı? («Para birimi çevirme hizmeti», verilen girdilerle bir çıktı üretir. Her istek birbirinden bağımsızdır. (stateless). "Alışveriş sepeti tutma servisi"nde, sepette hangi ürünlerin bulunduğu bilgisinin, sepetteki ürünler satın alınana kadar, istekler arası belirli bir süre tutulması gerekir.)

- Bir veri saklama mekanizması gerekiyor mu? (Ontoloji, dosya, ilişkisel veritabanı, nosql, memcached benzeri bellek mekanizmaları kullanılabilir. Bu bilgi, hem iş akışında hem de hizmetin tasarım ve uygulanmasında kullanılacak önemli bir bilgidir.)

- Servis şirket dışındaki üçüncü partiler tarafından da kullanılabilir olacak mı? (Kimliklendirme ve yetkilendirmenin doğru yapılması için önemlidir.)

- Servisi sadece bazı kurumlar mı kullanacak, yoksa herkes mi?

- Hangi protokoller ile ulaşılabilecek?

- Bu kurumlar için ayrı ayrı hizmet sözleşmeleri mi olacak?

- Servis kullanımı nasıl ücretlendirilecek?

- Başka kurumlar için erişim yetkisi kim tarafından hangi süreç ile verilecek?

Servis Tasarımı: Servisler kendi içerisinde bağımsız çalışırlar. Dışarıya "arayüzler" (interface) ile açılırlar. Servisin iç işleyişi ve dışarıya sunulacak arayüz bağımsız şekilde incelenebilir.

- Servisin iç tasarımı: Servisin iç tasarımı, klasik yazılım mühendisliği paradigmalarından uygun bir tanesinden yararlanılarak gerçekleştirilebilir. Bir servis kendi içinde kara kutu olduğu için, burada uygulanan yöntem ve uygulamanın nasıl olduğu, gereksinimler sağlandığı sürece kimsenin ilgi alanında olmayacaktır.

- Servis arayüzü tasarımı: Servis tarafından alınacak ve gönderilecek mesajların şeklini belirler. Servisin oluşabilecek hata/beklenmeyen durumlarda ne gibi hata mesajları döndüreceği belirlenir. Servislerin dış uygulamalarla sorunsuz bir şekilde çalışmasını garantilemek için servis arayüzünün net bir şekilde belirlenmesi önemlidir. Bu aşamada uluslararası standartların göz önüne alınması ile birlikte çalışabilirliği (interoperability) arttıracaktır.

- Servis performans tasarımı: Tasarlanan arayüzlerde gönderilip alınan mesaj sayısının mümkün olduğunca az olacak şekilde tasarlanması performans açısından önemlidir. Örneğin, bir kur hesaplama servisi için, paranın değerinin, biriminin ve dönüştürüleceği birimin tek bir defada alınması, art arda yapılacak üç ayrı mesaj iletimine göre daha hızlı olacaktır. Hizmetlerin durum bilgisine (state, session) ihtiyaç duyması durumunda bu bilginin nasıl aktarılacağı da arayüzde belirlenmiş olmalıdır.

Servis Uygulaması ve Yayına Alma: Yayına almak, servisin ilgili herkes tarafından ulaşılabileceği bir yerde çalışır hale getirilmesi işlemidir.

Tasarımı yapılmış servislerin sağladığı özelliklerin bir kısmı, kurum bünyesinde servis odaklı olarak geliştirilmemiş yazılımlarda hali hazırda bulunabilir.

Tekrar kodlamak yerine, zaten hali hazırda bulunan bir özelliğe servis odaklı erişmek için birer “adaptör” yazılabilir. Eski programlar klasik yöntemlerle ulaşırken, yeni programlar servis odaklı ulaşabilirler.

Servis Testi: Bu aşamada yazılımdaki hataların belirlenmesi sağlanmaktadır. Yazılımın fonksiyonel ve fonksiyonel olmayan gereksinimleri sağlayıp sağlamadığı kontrol edilir. Servis odaklı mimaride servis testi zor adımlardan biridir. Dışarıdan temin edilmiş kaynak kodu görülemeyen (kara kutu) hizmetler olabilir. Bu sebeple her zaman kaynak koduna dayalı test yöntemleri kullanılamayabilmektedir.

2.1.7 Servis odaklı yaklaşıma özgü problemler

- Dış kaynaklı servislerin işleyişi ve yapısı değiştirilebilir. Bu durum, daha önceden başarıyla tamamlanmış bir testin geçerliliğini yitirmesine sebep olabilir.

- Test ekibinin, bu servisin değiştirildiğini bilmemesi söz konusu olabilir. Yapılan eski testlerin de başarılı olmasından dolayı, testin tekrardan yapılması söz konusu olmaz ve son ürün sağlıklı çalışmayabilir.

- Kullanılacak bazı servislere, programın tasarım/kodlama aşmalarında değil, programın çalışması esnasında karar verilebilir.

- Program çalışma esnasında bir döviz dönüştürme servisini UDDI (Universal Description Discovery and Integration) ile arayıp bulup, WSDL/SOAP (Simple Object Access Protocol) aracılığıyla kullanabilir. Böyle bir durumda, bir döviz servisinin hep erişilebilir olması ve düzgün sonuçlar üretip üretmeyeceği test zamanında kesin olarak bilinemez.

- Servislerin yanıt süreleri üzerlerindeki yüke göre değişebilmektedir.

- Yazılımın performans değerleri bilinip, servis kaynaklı gecikmeler bilinemeyebilir.

- Olası problemler servislerin tipine ve özelliklerine göre değişebilmektedir.

- Risk analizi yapılmalıdır.
- Test senaryoları hazırlanmalıdır.

2.1.8. Konuyla ilgili akademik faaliyetler

S-Cube: Avrupa'da yazılım servisi gelişimini ve bu konuyla ilişkili yardımlaşmayı sağlamak için kurulmuş çok disiplinli, bütünleşik bir araştırma topluluğudur. Amaçları arasında, Avrupa'yı servis odaklı geliştirim lideri yapma ve yeni teknolojileri takip ederek uygulama ve endüstri ile güvenli bir ilişki kurmayı sağlamak yer almaktadır.

2008 yılından beri faaliyet göstermektedir. Servis odaklı yazılım geliştirimi ile ilgili pek çok konferans, çalışma toplantıları ve toplantı gerçekleştirmiştir. Aktif olarak çalışmaya devam etmektedir. Şekil 2.4'te S-Cube topluluğunun araştırma alanları gösterilmektedir (S-Cube Topluluğu, 2016).

Şekil 2.4 S-Cube Araştırma Alanları

S-Cube- PESOS (Principles of Engineering Service-Oriented Systems): S-Cube'ün en önemli çalışmalarından biridir. İlk olarak 2009 çalışma toplantısında çalışmalarına başlamıştır. Akademik çevreden ve iş dünyasından yazılım mühendisliği ile ilgili araştırma yapan kişileri, servis odaklı yazılım geliştirme konusunda deneyimli çalışanları bir araya getirmeyi amaçlamıştır.

Servis odaklı sistemlerle ilgili son gelişmeleri, yeni uygulama senaryolarını, yöntemleri, teknikleri, araçları ve deneyimleri paylaşmak için düzenlenmiştir. Şekil 2.5'te 2012'de düzenlenen bir etkinliğin afişi görülmektedir.

Şekil 2.5 S-Cube Pesos

SOSEG (Service Oriented Software Engineering Group): Notre Dame Üniversitesi'nde bir çalışma grubudur. **S**ervislerin çok hızlı bir şekilde kullanılmaya başlanması ve servis odaklı mimarinin avantajlarının farkedilmesiyle çalışmalarına başlamışlardır.

2007 ve 2008 yıllarında 2 çalışma toplantısı ve 2004 – 2010 yılları arasında servis odaklı yazılım mühendisliği ile ilgili konferanslar düzenlemişlerdir.

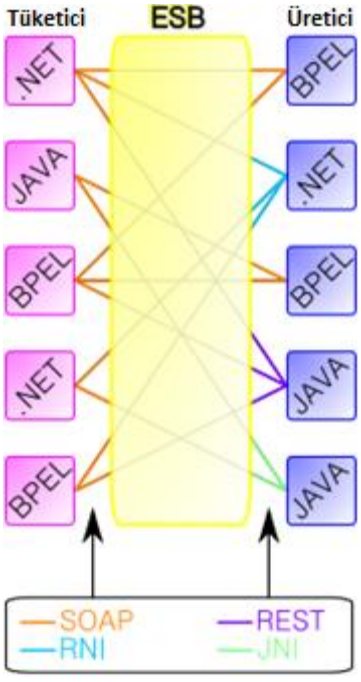
SeCSE(Service Centric Systems Engineering): Lancaster Üniversitesi'nde kurulmuş olan bir çalışma topluluğudur. Bileşen ve servis odaklı mühendislik üzerine çalışmaktadırlar.

Amaçları, sistem bütünleştiricileri ve servis sağlayıcıları için yöntemler, araçlar ve teknikler yaratmak, servis ve servis odaklı uygulamaları kullanmak ve bunları ucuza mal edecek şekilde geliştirmektir. 2003 yılından itibaren pek çok yayın yapmışlardır.

2.2 Kurumsal Servis Kanalı (Enterprise Service Bus - ESB)

Farklı uygulamaların birbirleri ile iletişim kurabilmesi için kullanılan bir uygulama mimarisidir. Yazılı olarak ilk kez, David Chappel'in Enterprise Service Bus isimli kitabında bahsedilmiştir (Chappel, 2004). Uygulamaların birbirleriyle iletişim kurmaları için birçok yöntem mevcuttur. Genellikle birebir uygulama iletişimi kullanılsa da, uygulama sayısı arttıkça birebir iletişimin maliyeti de artacağından, ESB bu noktada çok yararlı bir mimari olarak karşımıza çıkmaktadır. ESB, uygulamalar arasında bulunan ve veri alışverişini sağlayan bir hat olarak düşünülebilir. Bu yapı sayesinde uygulamalar birebir n adet uygulamaya bağlanmaktansa sadece ESB'ye bağlanarak diğer tüm uygulamalarla dolaylı olarak iletişim kurabilirler. ESB altyapısında servis odaklı yazılım mimarisi kullanılmaktadır. Servisler sadece ESB'ye bağlı olup, birbirlerinden ESB sayesinde soyutlanmış, dolayısıyla birbirlerinden bağımsız hale gelmiştir. İhtiyaç dahilinde servisler ESB üzerinden iletişim kurduğundan uç uygulamalar birbirlerinin nerede ve nasıl kurulu olduğundan habersizdirler.

Uygulamalar birer adaptör sayesinde servise bağlanırlar. Farklı protokolleri destekleyerek dönüşümü sağlayabilirler. Teknoloji kısıtı olmaksızın her uygulama bağlandığından, bu bağımlılık da ortadan kaldırılmış olur. Şekil 2.6'da ESB'nin genel yapısı modellenmeye çalışılmıştır.



Şekil 2.6 ESB Genel Yapısı

ESB yapısı veri birleştirme ve/veya veri senkronizasyonu için kullanılabilir. Uygulanacak sistemin ihtiyaç duyduğu yapıya göre, bu başlıklar altında şekillendirilebilir.

Veri Bütünleştirme (Data Integration): Farklı kaynaklardan elde edilen verinin birleştirilerek anlamlı bir bilgi oluşturulmasıdır. Her küçük sistem kendi içerisinde anlamlı veriler barındırır. Ancak bir seviye yukarı çıkıp, küçük sistemlerin tamamından bir anlamlı veri çıkarmak istediğimizde veri bütünleştirme büyük önem kazanır.

Veri Senkronizasyonu (Data Synchronization): Uygulamalarda kavram olarak ortak verilerin paylaşılmasıdır. Örneğin şirketin 'A' sisteminde oluşturulan bir verinin, 'B' sisteminde de bulunması tutarlılık açısından önemlidir. Müşteri verisi gibi her noktada ihtiyaç duyulacak verilerin otomatik olarak diğer sistemlerle senkronize olması kaçınılmaz bir gerekliliktir.

2.2.1 Mimari özellikleri ve faydaları

ESB, altyapısında SOA'yı temel aldığından, SOA'nın çoğu özelliğini barındırmaktadır. Temel özellikleri aşağıdaki gibi özetlenebilir.

- Dağıtık yapıdaki uygulama ve servisleri birlikte çalıştırmak için son derece uygun bir mimaridir.
- Bütün uygulamalar ESB'ye bağlandığından süreçlere belirli bir standart getirmiş olur.
- SOA'nın özelliklerinden de olan gevşek bağlılık ile iş süreçlerinin akışı kolaylıkla değiştirilebilir.
- Uygulamalar arası birebir bağlantıları ortadan kaldırdığından konum bağımsızlığı sağlar.
- Mesaj içeriğine göre mesaj filtreleme ve yönlendirme özelliği vardır.
- Farklı protokolleri desteklediğinden son derece esnek bir yapıya sahiptir. Protokoller arasında dönüşüm yapabilirler.

- İçerilerinde güvenlik servisleri barındırabilirler.

2.2.2 Ticari ürünler

ESB aynı zamanda ürün olarak da karşımıza çıkmaktadır. Bu mimariden yola çıkarak kapsamlı ürünler geliştirilmiştir. Bu ürünler ESB'nin içerisinde güvenlik, veri dönüşümü vs. gibi birçok katmanı da kullanıcılara sunmaktadır. Büyük firmalara ait çok gelişmiş ürünler olduklarından satın alma maliyetleri de oldukça yüksektir. Ürünlerden bazıları aşağıda listelenmektedir.

- IBM WebSphere ESB (IBM Knowledge Centre, 2017)
- Microsoft BizTalk Server (Microsoft, 2017)
- Oracle Enterprise Service Bus (Oracle, 2017)
- Progress Sonic ESB (Progress, 2017)
- Tibco Software (TIBCO Software Inc., 2017)
- Fiorano ESB (Fiarano, 2017)
- IONA (Iona Software Solutions Inc., 2017)
- Mule ESB (MuleSoft, 2017)
- Apache ServiceMix (Apache, 2017)
- JBoss ESB (JBoss, 2017)

3. İLGİLİ ÇALIŞMALAR

ESB, bir akademik çalışma sonucu ortaya çıkmış bir olgudan ziyade gereksinimler sonucu ortaya çıkmış bir yapıdır. David A. Chappell'in de kitabında bahsettiği üzere, temelleri var olan teknolojilerle çözülemeyen karmaşık problemlerin çözümü için gerekli gerçek gereksinimlere dayanır (Chappell, 2004). Filozof Plato'nun da bir sözüne ithafen "Gereksinim, icatların annesidir.". Birçok alanda birçok gereksinim bulunmaktadır. ESB, özelleştirilmiş bir alana ait olmadığı, sabit bir yapıdan ziyade esnek bir mimari olduğundan birçok farklı alanda kullanımına rastlanmaktadır. Bu bölümde ESB'nin farklı kullanım alanları üzerine yapılan çalışmalardan bahsedilmektedir.

(Kurniawan and Ashari, 2015)'te yapılmış olan çalışma, yöneticiler için bilgileri farklı noktalardan toplayarak anlamlı hale getirecek bilgi gösterge sistemi geliştirilmesini hedeflemiştir. Farklı yönetim departmanlarında kullanılan, farklı platformlar ve teknolojiler kullanan veritabanlarından verileri toplayarak tek bir ekranda gösterecek bir sistem amaçlanmıştır. Her bir veritabanına gidilerek ayrı ayrı verileri çekip birleştirmek yerine, ESB yapısı kullanılarak, tek bir istek ile istenen verinin tümü tek seferde çekilebilmektedir. ESB kullanımı sayesinde, yeni veritabanı eklenmesi veya mevcut bir veritabanının değişmesi gibi durumlarda temel uygulama hiç etkilenmeden çalışmaya devam edebilecektir. Bu bağımsız yapısıyla, tez kapsamında yapılacak çalışma için de örnek teşkil etmektedir. Ayrıca burada yapılmaya çalışılan bilgi gösterim sistemi, ileriki çalışmalarda da iş uygulamaları arasındaki ortak raporlama prensibinde kullanılabilir.

ESB'yi içerisinde birçok standart barındıran bir bütünleştirme platformu olarak tanımlamak mümkündür. Web hizmetleri, mesajlaşma, veri dönüşümü, akıllı yönlendirme gibi birçok standardı barındırabilen bir sistemde, kullanıcı sayısı ve trafik gibi konular göz önüne alınarak altyapı tasarlanmalı ve performansın stabil olması için çalışmalar yapılmalıdır. (Liu et al., 2007)' de COTS ESB platformunda oluşturulan bir servisin performansını ölçmek için bir yöntem sunulmuştur. Bu yaklaşım giderleri ölçmek için kıyaslama tekniklerini kullanmaktadır. Servislerin performans özellikleri bir kuyruk ağında modellenir. Bu model, bir kredi brokerliği yapan ESB tabanlı uygulama tarafından doğrulanmıştır.

ESB farklı mimarileri birleştirmek için de kullanılabilir. Marechaux **(**2006), SOA (Service Oriented Architecture) ve EDA (Event Driven Architecture)

konseptlerinin ESB kapsamında nasıl bir araya getirilebildiğini anlatmaktadır. SOA, bağımlılığı az, birebir iletişim sağlayan, tüketici bazlı tetiklemeye ihtiyaç duyan ve senkron çalışan bir mimari örneğidir. EDA ise tamamıyla bağımsız, çoklu iletişimi destekleyen, olay bazlı tetiklenen ve asenkron bir mimaridir. ESB, EDA ve SOA yaklaşımlarını, farklı servislerin bütünleştirmesi için kullanır ve bir köprü görevi görür. ESB üzerinde gerçekleştirilebilecek bir hizmet, bir tüketici veya bir olay tarafından tetiklenebilir. Senkron veya asenkron etkileşimi destekler. Yani özetle, ESB, SOA ve EDA paradigmalarının tüm özelliklerini destekler.

Standart ESB işleyişinde yapısı zaten belli olan mesaj kümeleri alınır ve iletilir. Mesajın nereye yönlendirileceği zaten bellidir ve konfigürasyon dosyaları aracılığıyla adres bulunarak yönlendirme yapılır. (Ziyaeva et al., 2008)'de, yönlendirmenin, mesaj içeriğine göre dinamik şekilde yapılması amaçlanmıştır. Hedeflenen yapı için 3 ana katman öngörülmüştür. Bunlar; iş süreci, ESB ve mesaj içeriğine göre nereye yönlendirileceğine karar verecek olan katmanlardır.

Hartman (2008), Çinli büyük bir üretici firmanın, Batılı bir rakibini aldıktan sonraki yaşadığı sorunları ve çözüm olarak ESB'nin kullanımını anlatmıştır. Satın alımdan önce şirketin satışları sadece yurtiçi satışlara dayanırken, satın alma sonrası ilişki temelli satış ve internet üzerinden doğrudan müşteriye satış modelleri de eklenmiştir. Piyasa koşulları gereği hızlı bir adaptasyon olması gerekmektedir. Farklı ERP sistemlerinin öğrenilmesi gibi konular ortaya çıkmıştır. Bu farklı yapıların birleştirilmesi için ESB kullanılmıştır. Bu yapı, bu tez kapsamında amaçlanan gerçek uygulamaların bütünleştirmesi için örnek teşkil etmektedir.

ESB sistemleri kurumsal mimarilerde uygulamalar arası iletişim ve koordinasyon için kullanılabildiği gibi, konseptlerin ve ilişkilerin tanımlanması, kısıtların ve kuralların belirlenip operasyonların oluşturulması amacıyla bilgi modellerinin (information model) otomatik olarak yaratılmasında da kullanılabilir. (Bushle et al. 2012) çalışmasında; elle yapılan, zaman alıcı ve hataya açık bir işlem olan kurumsal mimari (enterprise architecture) dokümantasyonu sürecini otomatikleştirmek için, moda sektöründe çalışan bir şirketin SAP PI ESB sistemi veri modelleri üzerinde tersine mühendislik yapılarak bilgi modellerinin çıkarımı sağlanmıştır.

Kurumsal servis kanalı sistemlerinin en önemli özelliklerinden biri olan tak-kullan (plug and play), uygulama bütünleştirme için büyük bir avantajdır. Özellikle ölçeklenebilirlik ve koordineli çalışma gerektiren ortamlarda, minimal zaman ve efor ile problemlerin çözülebilmesini sağlar. Örneğin bir çalışmada (Alaa et al. 2010) SOA tabanlı grid hesaplama uygulamaları için ESB temelli bir mimari önerilmekte ve servislerin kolayca ve dinamik bir biçimde gride dahil olabilmesi sağlanmaktadır.

Kurumsal SOA sistemlerinde karmaşıklığın artmasıyla birlikte, çalışma ortamlarındaki değişikliklere kolay adaptasyon sağlayacak servis yönetimi yapacak sistemlere ihtiyaçlar da gün geçtikçe artmaktadır. Adaptive Enterprise Service Bus (Szydło et al. 2012) isimli çalışmada çeşitli kullanım metriklerine göre istatistiksel modeller oluşturan ve bu modellere dayanarak adaptif servis çalıştırması (execution) sağlayan bir ESB uygulama çatısı sunulmaktadır.

ESB sistemlerinin, kurumsal uygulama bütünleştirmesi (EAI - Enterprise Application Integration) alanında, önemi gün geçtikçe artmaktadır. Dağıtık kurumsal uygulamalarda bütünleştirme uygulamaları konulu bir araştırmada (Wu et al. 2014), ESB mimarisinin heterojen ara katman (middleware) ürünleri ile birlikte çalışabilmesi sebebiyle, bir mimariden ziyade yeni bir ara katman teknolojisi olarak görüldüğünden bahsedilmekte ve veri bütünleştirme konusunda ümit verici bir çatı olduğuna değinilmektedir.

Servis tabanlı uygulamalarda yeni eğilimlerden biri de XaaS (her şey bir servistir) yaklaşımıdır. Bu yaklaşımda, depolama alanları ve hesaplama aygıtları da dahil olmak üzere her türlü kaynak bir servis olarak sunulmaktadır. (Baude et al. 2010)'nin çalışmasında bir XaaS ortamı altyapısı için standart bir ESB sistemi baz alınarak, hiyerarşik bir yaklaşım ile ESB'nin mesaj kayıtçıları ve yönlendiricileri üzerine ölçekleme desteği eklenmiştir.

Veri bütünleştirmesinde olduğu gibi ESB sistemlerinden veri dönüşümü operasyonlarında da yararlanılmaktadır. Shi'nin çalışmasında (Shi et al. 2011), bir sayısal kampüs projesi için Oracle'ın ESB sistemi ile standart ETL (Export Transform Load) veri dönüşümü süreci arasında veri boyutuna bağlı olarak performans karşılaştırması yapılmaktadır.

ESB veri dönüşümü üzerinde yapılan bir başka araştırmada (Fan et al. 2012), XML (Extensible Markup Language) tabanlı modelleri benimseyen sistemlerde XSLT 2.0 (Extensible Stylesheet Language Transformations) veri dönüşümü

kaynaklı performans kayıplarından söz edilmekte ve çözüm olarak yeni bir veri bütünleştirme modeli (DIM – Data Integration Model) önerilmektedir. Önerilen model, dönüştürülen veri miktarına bağlı olarak geleneksel XML modeline göre daha hızlı çalışmaktadır.

REST (Representational State Transfer – Gösterimsel Durum Transferi) tabanlı servislerin durum bağımsız (stateless) çalışması, gelen isteklerin (request) basit kaynak adreslerinden (URI - Uniform Resource Identifier) oluşması ve sisteme fazla yük getirmeyen klasik üst metin transfer protokolünü (HTTP - Hyper-Text Transfer Protocol) kullanması sebebiyle, veri bütünleştirmesinde kullanımı yaygınlaşmaktadır. Junye'nin çalışmasında (Junye et al. 2009) REST tabanlı bir kurumsal servis yolu mimarisi önerilmekte ve bir durum çalışması ile desteklenmektedir. İlgili mimari servis yolunu, özelleştirilmiş operasyonları desteklemeksizin sadece veri alışverişi amacıyla kullanmakta olup veri bileşenlerini bir XML tabanlı RMM (Relationship Management Methodology – İlişki Yönetim Metodolojisi) ile ifade etmektedir.

Kurumsal veri yolu mimarileri sağladıkları genel veri bütünleştirme nitelikleri sayesinde, çok farklı uygulama alanlarında kullanılabilmektedir. Öncekilerden farklı olarak (Dai 2011)'nin çalışmasında, elektrik güç sistemleri ile ilgili uygulamaların bütünleştirmesi için, ilgili uygulamalar birer servis şeklinde kapsüllenmiş ve standart bir ESB yapısı üzerinden veri alışverişi yapmaları sağlanmıştır.

Önceki çalışmalarda veri temelli bütünleştirme uygulamaları üzerinde durulmuş ve bunlar üzerinde performans karşılaştırmaları yapılmıştır. Bu tez kapsamında diğer çalışmalardan farklı olarak veri senkronizasyonu yanında süreç senkronizasyonunun da üzerinde durulmuş olup, performans odaklı REST tabanlı bir kurumsal servis yolu tasarımı yapılmıştır. Veri serileştirme için yine performans kısıtları göz önüne alınarak geleneksel XML formatı yerine yüksek performans başarımı sağlayan JSON (Javascript Object Notation) formatı tercih edilmiştir. Bunların yanında üçüncü parti servislerin veri yoluna adaptasyonunu kolaylaştırmak amacıyla, otomatik servis dokümantasyonu araçlarından biri olan Swagger (SmartBear Software Inc., 2017) isimli yazılımdan faydalanılmıştır.

4. PROBLEM TANIMI

Teknolojik gelişmelerin üst düzeyde olduğu günümüzde yazılım sektörü de günden güne büyümektedir. Hayatımızın her alanında kullandığımız uygulamalar sayesinde yazılımla iç içe yaşadığımız gibi, iş dünyasında da yazılımların yeri vazgeçilmez bir noktaya erişmiştir. İş süreçleri yazılımlarla sürdürülmekte, hızlandırılmakta ve geliştirilmektedir.

Şirketlerin kaynaklarının verimi üst seviyeye taşıyacak şekilde planlanması, müşteri bilgilerinin etkin bir şekilde işlenerek memnuniyet sağlanması, çalışanlarla ilgili her türlü bilginin tutulması, mali belgelerin oluşturulması, üretilen ürün veya hizmetin bayilere, son tüketiciye ulaştırılması gibi birçok alanda yazılımların desteği büyüktür. Her biri belli bir alanı benimsemiş farklı yazılımlarla tüm süreçler kontrol altına alınabilir.

İş dünyasında kullanılan yazılımlara yönelik genel geçer hakimiyeti kanıtlanmış bazı alanlar aşağıda açıklanmaya çalışılmıştır.

ERP (Enterprise Resource Planning – Kurumsal Kaynak Planlama): İşletmelerde mal ve hizmet üretimi için gereken işgücü, makine, malzeme gibi kaynakların verimli bir şekilde kullanılmasını sağlayan bütünleşik yönetim sistemleridir. Sistemin amacı, işletmenin tüm kaynaklarının birleştirilerek, en verimli şekilde kullanılmasını sağlamaktır. Bu kavram ilk olarak üretim sektöründe kullanılmaya başlanmış, ancak daha sonra geniş alanlara yayılmıştır. Müşteriler, teklifler, siparişler, stok takibi, çek senet işlemleri ve daha birçok işlem ERP üzerinden yürütülebilir.

B2B (Business-to-Business – Şirketler Arası Pazarlama): İnternet ve e-ticaretin son zamanlardaki popülaritesi, kullanıcıları ve şirketleri internet üzerinden alışverişe yöneltmektedir, Teknoloji sektöründeki gelişmeler birçok sektörde olduğu gibi klasik ticaret anlayışını da etkilemiş ve değiştirmiştir. Elektronik Ticaret Koordinasyon Kurulu (ETKK) Hukuk Çalışma Grubu'nun 8 Mayıs 1998 tarihli raporunda elektronik ticaret; "bireyler ve kurumların, açık ağ ortamında internet ya da sınırlı sayıda kullanıcı tarafından ulaşılabilen kapalı ağ ortamlarında intranet yazı, ses ve görüntü şeklindeki sayısal bilgilerin işlenmesi, iletilmesi ve saklanması temeline dayanan ve bir değer yaratmayı amaçlayan ticari işlemlerin tümünü ifade etmektedir." şeklinde tanımlanmıştır. B2B sistemleri canlı satışın yapıldığı sistemler

olduğundan, ürünlerin güncel bir şekilde sistemle entegre tutulması oldukça önemlidir.

CRM (Customer Relationship Management – Müşteri İlişkileri Yönetimi): Günümüz tüketim sektöründe müşteri birinci plandaki nesnedir. Eskiden olduğu gibi müşteriler firmaların peşinden koşmamakta, rekabet arttığından firmalar müşterilerin peşinden koşmaktadır. Bu koşullarda müşteriyle ilgili bütün bilgi ve kayıtlar çok kıymetli hale gelmiştir. İşletmeler mevcut müşterilerini korumak ve yeni müşteriler edinmek üzerine çalışmaktadırlar. Mevcut müşteriler için sadakat oluşturmak için müşteri beğenilerine uygun hareket etmektedirler. Müşteri portföyünü geliştirmek için kullanılan müşteri odaklı her türlü bilginin saklanıp işlenebildiği sistemler CRM sistemleri olarak adlandırılır. Bu uygulamalar işletmelerin müşteriyle kurulan bütün etkileşimlerini kayıt altına alır, müşterilerle yürütülecek tüm teklif, sipariş süreçlerinin takibini sağlar. CRM yazılımları, mutlu ve sadık müşteriler yaratma, müşteri ihtiyaçlarına göre doğru hedef kitleye doğru pazarlama teknikleri uygulanmasına da yardımcı olur.

HRM (Human Resources Management – İnsan Kaynakları Yönetimi): Bir işletmenin başarısı birçok etmene bağlı olsa da, bunların başında işletmedeki insan kaynağının kalitesi gelir. Bu göz önüne alındığında, çalışan takibinin yapılabilmesi işletme için önemli bir durum haline gelmektedir. HRM yazılımları, bir işletmenin işe alma sürecinden başlayarak, personel ile ilgili olabilecek tüm süreçlerini içerir. İşletme personeliyle ilgili her türlü veri saklanabilir, firma içi ve dışı etkinlik veya eğitimler düzenlenebilir. Çalışanlara ait bordro ve ücret yönetimi de bu sistemler üzerinden yapılabilir. Kariyer ve izin yönetimi de bu sistemlerde gerçekleştirilebilen işlerdendir.

Yukarıda bahsi geçen ve daha örnekleri çoğaltılabilecek birçok alanda iş yazılımları üreten yerli ve yabancı birçok yazılım şirketi bulunmaktadır. Bu tez kapsamında da 30 yılı aşkın geçmişi olan ve iş yazılımları konusunda sektörde önde gelen bir yazılım firmasının gerçek sorunları üzerine çalışılmıştır. Aşağıda verilecek örnekler bu firmanın sunduğu iş yazılımları kullanılarak oluşturulmuştur.

Yukarıda bahsedilen sistemler birçok firma tarafından ihtiyaç duyulmakta ve kullanılmaktadır. İşletmelerin gereksinimlerine göre bir veya birden fazla sistem aynı anda kullanılabilmektedir. Böylelikle iş akışı eksik nokta kalmaksızın sağlanabilir. Farklı sistemlerin birbirileri arasında ortak noktaları bulunmaktadır. Bu ortak noktaların bazılarından bahsedecek olursak; öncelikli olarak bütün

uygulamalarda sistemi kullanacak kullanıcıların varlığı somuttur. Sistemler arasında hangi kullanıcının diğer sistemdeki hangi kullanıcı olduğu bilgisi paylaşılabilen bir bilgidir.

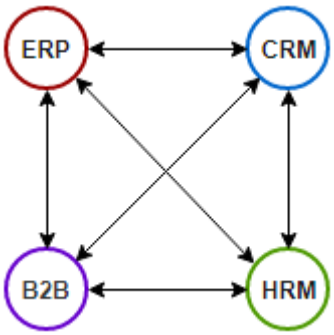
'Müşteri' kavramı ERP, B2B ve CRM alanlarında ortak ve önemli bir kavramdır. Bu sistemler arasında müşteri senkronizasyonu kaçınılmazdır. Bir sistemdeki müşterinin diğer sistemlerde de görünür olması, takip ve yönetim açısından önemlidir.

'Sipariş' kavramı ERP ve B2B alanlarının ortak kavramlarından biridir. İki sistemi de kullanan bir işletme, B2B üzerinden girilmiş olan siparişi ERP üzerinde tekrar girmek istemeyeceğinden, kaçınılmaz bir bütünleştirme gereksinimi doğar.

Yukarıda bahsedilen ve daha da çoğaltılabilecek örneklerle, sistemler arası senkronizasyonun ve iletişimin önemi somut olarak karşımıza çıkmaktadır. Sonuç olarak her sistemin, diğer bütün sistemlerle bağlantı içinde olması gerektiği gerçeği ortaya çıkar. Bu koşullarda matematiksel olarak formüle edersek, n adet uygulamanın birbirleriyle konuşabilmesi için n*(n-1)/2 adet bağlantı gerekmektedir.

n = uygulama sayısı

n*(n-1)/2 = bağlantı sayısı



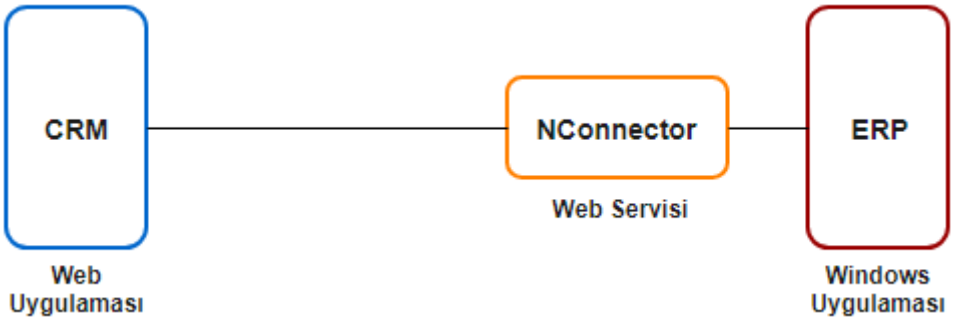
Şekil 4.1 Örnek Uygulamaların Bağlantı Gösterimi

Şekil 4.1'deki yapıda her iki ürün arasında ayrı bir bağ olduğundan, her bir bağlantı için farklı yöntemler kullanılabilir. Sıfırdan yazılan uygulamalar için aynı veritabanı üzerinde uygulama geliştirmek bütünleştirme ihtiyacını en aza indirebilir, ancak mevcut farklı veritabanları olan uygulamalar için oluşturulması imkansızdır. Bu durumda veritabanları aynı konumda tutularak veri bütünleştirmesi veritabanı

düzeyinde yapılabilir, ancak bunun sağlanamayacağı durumda bütünleştirme bir problem haline gelecektir. Veritabanından bağımsız iki uygulama düzeyinde bütünleştirme seçeneklerden biri olabilir. Bu durumda da ilave yazılım maliyeti her ikili ürün grubu için tekrarlanacaktır.

Örnek olarak ERP ile CRM ürünleri arasında kullanılan yapıyı detaylı incelemeye çalışalım.

Çalışma kapsamında incelenen ERP ürünü Windows uygulaması olup, masaüstü uygulaması olarak çalışmaktadır. CRM ürünü ise web uygulaması olarak sunulmaktadır. Her iki uygulama da kendine özel veritabanları kullanmaktadır. ERP ve CRM arasındaki ürün, müşteri, teklif, sipariş vb. gibi kayıtların aktarımı için NConnector adı verilen bir adaptör kullanılır. Bu adaptör ERP uygulaması önünde konumlanarak CRM ve ERP arasında veri alışverişini sağlar. Ancak bu uygulama tek taraflı olup, CRM uygulaması tarafından kullanılmaktadır. Web servisi olarak kurulan bu uygulama sayesinde veri alışverişi sağlanmış olur. İki uygulama arasındaki bağlantı modellemesi Şekil 4.2'de gösterilmektedir.



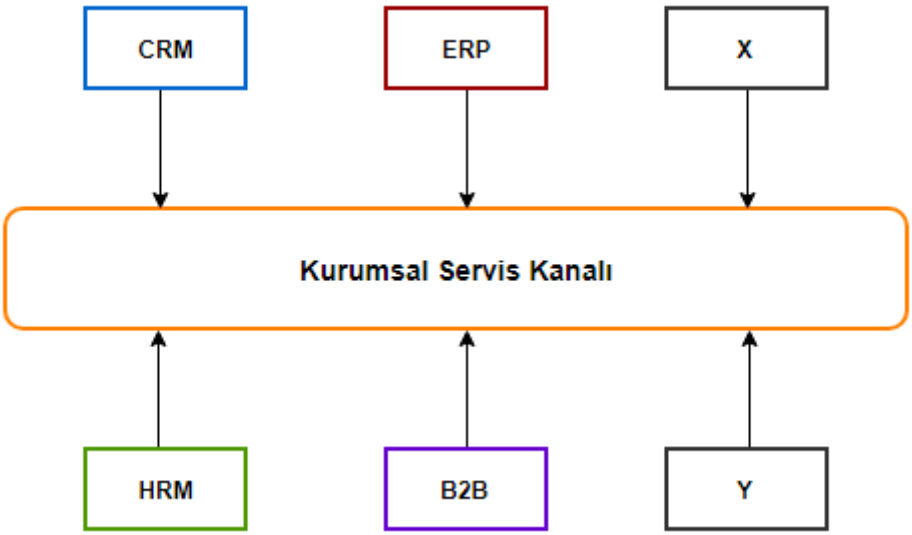
Şekil 4.2 İki Uygulama Arasındaki Bağlantı Modellemesi

Yukarıdaki örnek, sadece iki uygulama arasındaki iletişim için geliştirilmiş olup, başka uygulama türleri arasında kullanılamamaktadır. Başka ikililer için de yine benzer yollarla iletişim kanalı oluşturulması gerekmektedir. Uygulama sayısı az olduğunda, uygulamalar birbirlerinin konumlarını bilerek özelleşmiş yöntemlerle senkronizasyona açık hale gelebilirler. Ancak uygulama sayısı arttıkça, bütün uygulamaların iletişim halinde olması da gittikçe güçleşecektir. Yapıya katılan her yeni uygulama ile birlikte, var olan uygulama sayısı kadar daha özelleşmiş iletişim sistemi geliştirilmesi gerekir. Bu da yönetilebilirliği ve ölçeklenebilirliği düşürecektir. Bu nedenle de, uygulamaların çoğaldığı senaryolar için tüm uygulamaların kolaylıkla adapte olabileceği genel bir çözüme ihtiyaç bulunmaktadır.

5. ÇÖZÜM

Bahsedilen probleme istinaden, yeni bir uygulama kurgulanan bütünleşik sisteme katıldığında, mevcut uygulamaların etkilenmeyeceği bir yapı üzerinde durulmuştur. Kurumsal servis kanalı mimarisi kullanılarak uygulamalar arası veri aktarımının kolayca yapılabilmesi amaçlanmıştır. Kurumsal servis kanalı verinin tek bir noktada toplanması üzerine de kurgulanabilir. Ancak bu tez kapsamında gerçek uygulamaların sorunlarının çözülmesi amaçlandığından ve mevcut uygulamaların yapılarının değiştirilmesi büyük maliyet oluşturacağından verinin servis kanalı üzerinden aktarımı sağlanarak her uygulamanın kendi veritabanını güncellemesi amaçlanmıştır.

Önerilen yapıda, uygulamaların ortasında yer alacak bir servis kanalı oluşturulmuştur. Uygulamalar sadece servis kanalını tanıyacak ve kendilerini servis kanalına tanıtacaklardır. Şekil 5.1'de önerilen mimarinin basit şekilde modellenmiş halini göstermektedir.



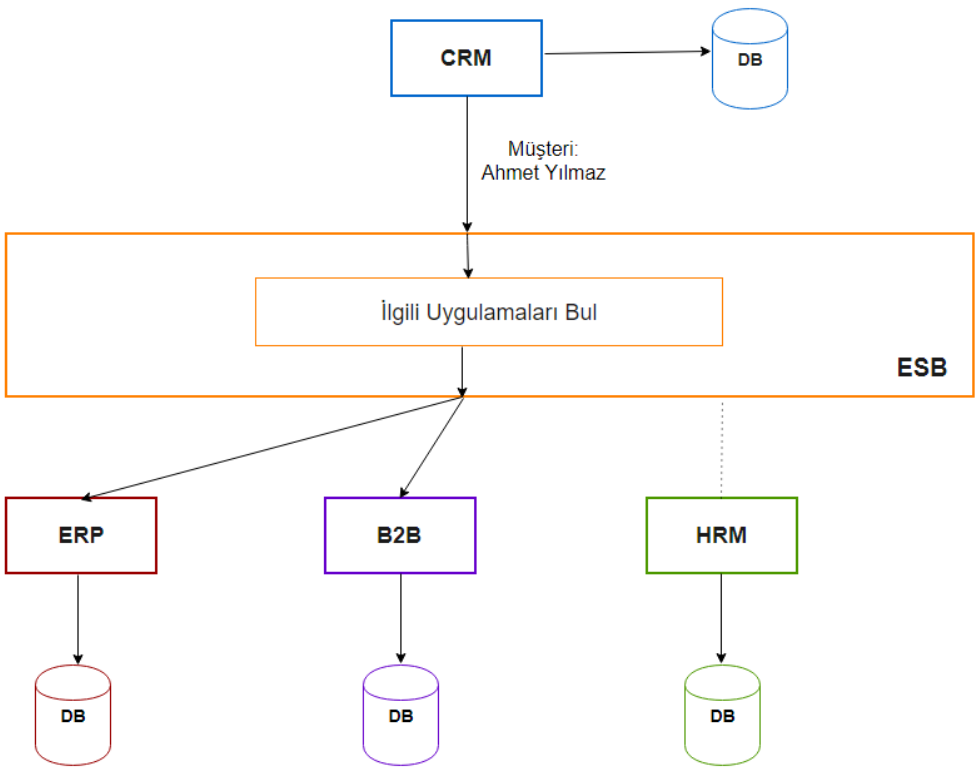
Şekil 5.1 ESB Modellemesi

Her bir uygulama, servis kanalına kendini tanıtarak sisteme katılmış olur. Diğer uygulamalarla birebir iletişim kurmasına gerek kalmaksızın, servis kanalı üzerinden hepsiyle bağlantı sağlamış olur.

Bir verinin servis kanalı üzerinde nasıl hareket edeceğini şu şekilde örneklendirebiliriz. Müşteri bilgisi birçok sistemde kayıt edilen ve kullanılan bir bilgidir. Büyük firmaların süreçlerinde birden fazla yazılım kullandığı düşünülürse,

müşteri bilgisinin bir uygulamadan girildiğinde diğer uygulamalarda da otomatik olarak oluşturulması önemli bir işlevdir. Örneğin; CRM uygulamasından girilmiş olan bir müşteriye, bir satış yapılmak istendiğinde ERP üzerinden kesilecek fatura için aynı müşteri bilgisinin ERP üzerinde de oluşturulması gerekmektedir. Eğer müşteri kaydı otomatik olarak oluşturulmazsa, bu sistemden tekrar kayıt girilecek ve CRM'deki müşteriyle gerçekte aynı kişi olduğu bilgisi anlaşılmayacaktır. Bunun yerine CRM'de müşteri kaydı oluşturulduğunda, aynı müşteri kaydı ERP'de de oluşturulmalı ve faklı veritabanları üzerinde kaydedilmiş olan, gerçekte tek kişiye ait kayıtlara tekil ve benzersiz bir tanımlayıcı (id) atanmalıdır.

Şekil 5.2'de anlatılan senaryo şu şekilde işlemektedir. CRM uygulamasından bir müşteri kaydı yapılır ve kendi veritabanına kaydedilir. Aynı zamanda veri ESB'ye gönderilir. Hem veritabanında oluşan hem de ESB'ye gönderilen veri, içerisinde tekil bir tanımlayıcı barındırır. ESB'ye gelen veri incelenerek, hangi uygulamalara gönderileceği bilgisine karar verilir. Daha sonra ilgili uygulamalara gönderilir ve uygulamalarca veritabanlarına kaydedilir.



Şekil 5.2 ESB - İlgili Uygulamaların Bulunması

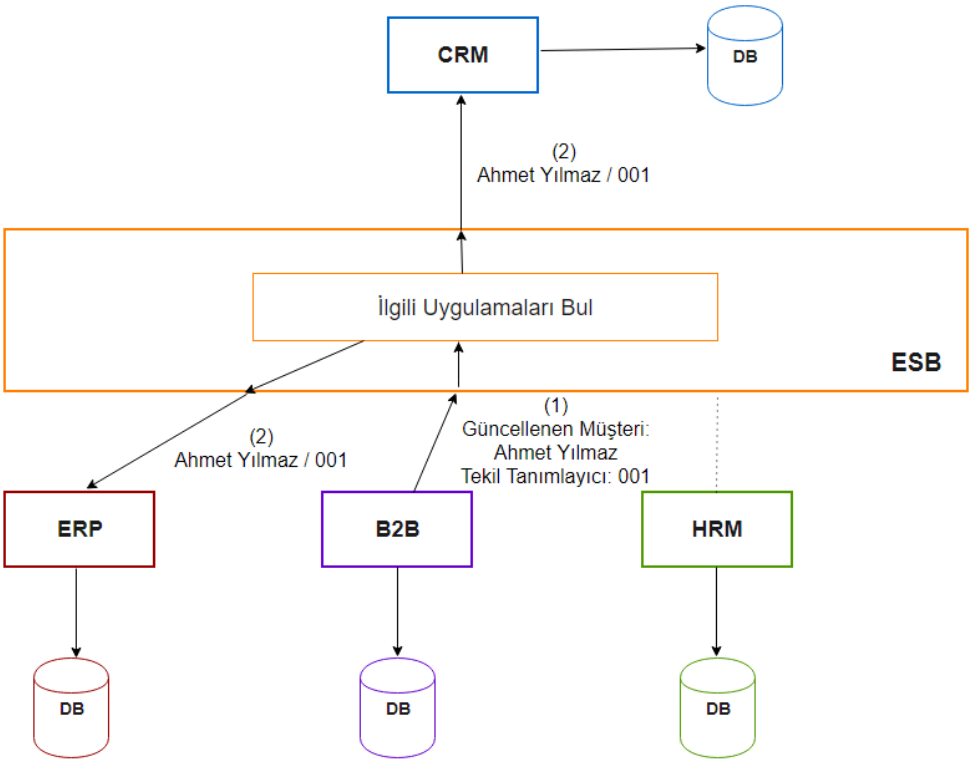
Şekil 5.2'de görüldüğü gibi ERP ve B2B uygulamalarına veri gönderilirken, HRM uygulamasına gönderilmemiştir. Bu sitemlerin veritabanlarına baktığımızda aynı kayıtları aynı tekil tanımlayıcı bilgisiyle bulabiliriz. Veritabanları ve veriyi

tutma şekilleri bu noktada önemsiz hale gelir. Bir kaydın başka bir sistemdeki eşleniğini bulmak için sadece tekil tanımlayıcı bilgisini kullanabiliriz.

Tekil tanımlayıcı bilgisi sorgulama dışında, güncelleme işleminde de bize yardımcı olacak bir alandır. Az önceki senaryoda anlatılan CRM üzerinden girilen müşteri kaydının diğer sistemlere gönderildiği gibi, bu sistemlerden herhangi biri tarafından güncellenen bilginin de diğer sistemlere aktarılması gerekir. Aksi takdirde, bilgiler aynı kişiye ait olsa da bazı bilgileri tutarsızlıklar gösterebilir. Örneğin B2B sisteminde müşteri telefon bilgisi güncellenmişse ve bu CRM sistemine aktarılmazsa, memnuniyet anketi için müşterinin eski numarası arandığında müşteriye ulaşılamayacaktır. Sistemdeki veriler karşılaştırıldığında hangi verinin güvenilir ve doğru olduğu anlaşılamayacaktır.

Telefon vb. gibi birçok bilginin güncellenmesi durumda, diğer sistemlerde de gerekli güncellemelerin otomatik olarak yapılması gerekmektedir. Bunun için de yeni kayıtta olduğu gibi, herhangi bir sistemde müşteri verisi güncellendiğinde güncellenen veri tekil tanımlayıcı ile birlikte servis kanalına gönderilir. Servis kanalı, bu veriyi hangi uygulamalara göndereceğine karar vererek güncelleme bilgisini ilgili yerlere gönderir. Şekil 5.3'te anlatılan yapı gösterilmektedir.

B2B uygulamasında 001 tekil tanımlayıcılı Ahmet Yılmaz isimli müşteri güncellenmiş ve bu güncellenme bilgisi ESB'ye gönderilmiştir. ESB bu bilgiyi hangi uygulamalara göndereceğine karar verir ve o uygulamalara gönderir. Bu örnekte, güncelleme bilgisi CRM ve ERP uygulamalarına gönderilmiştir. Bu uygulamalara giden istek, 001 tekil tanımlayıcılı müşterinin güncellenmesiyle ilgili isteği içerir. Her bir uygulama için kendi sistemlerinde bu kayıt bulunarak gerekli güncellemeler yapılır. Müşteri kaydı bütün sistemlerde güncellenmiş ve tutarsız kayıt durumu ortadan kaldırılmıştır.



Şekil 5.3 Veri Güncellemesi

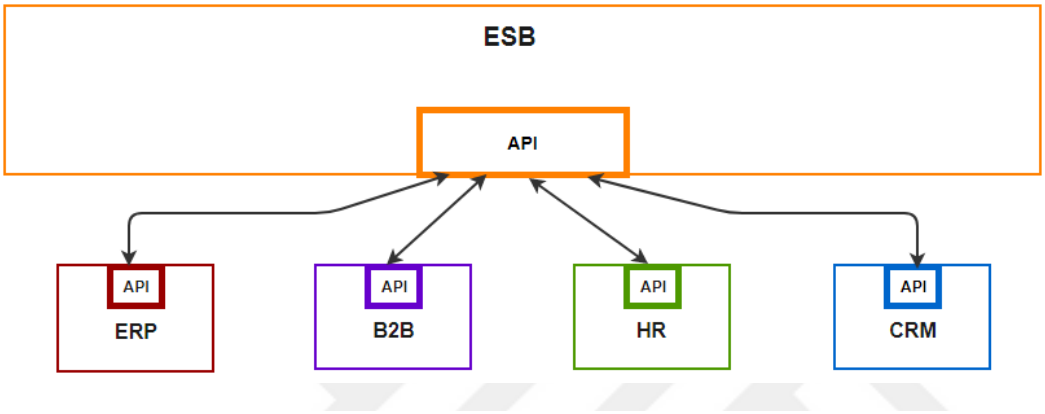
5.1 Uygulamaların ESB Kullanım Süreci

Bir uygulamanın ESB'ye bağlanabilmesi için, öncelikle ESB'nin adresini biliyor olması gerekir. Bu bilginin değişebileceği düşünülmesi gerektiğinden parametrik şekilde tanımlanması doğru olacaktır. ESB'nin uygulamayı tanıyabilmesi için öncelikle uygulamanın kendisini ESB'ye tanıtması gerekmektedir. Hangi uygulama olduğunu, kendine ait olan tekil tanımlayıcı bilgisini, ESB'ye hangi veri objeleri için bağlantı kuracağını ve ESB'nin ulaşması için açtığı API'yi belirtmelidir. Bu bilgiler ESB'ye gönderildikten sonra ESB bu uygulamayı tanır ve nasıl iletişim kuracağı bilgisi netleşir. Bu noktada uygulama bütünleşik sisteme katılmış olur.

Uygulama programlama arayüzü kavramı bu bağlantıdaki en önemli noktadır. API bir uygulamadaki işlevleri, başka bir uygulamanın kullanabilmesi için oluşturulan bir yapıdır. Bu sistemde de ESB'nin uygulama ile iletişime geçebilmesi için bir API'ye ihtiyaç vardır. Uygulamanın API'si içerisinde veri aktarımını sağlayacak fonksiyonların yazılmış olması gerekir. Örneğin; başka bir uygulamadan

girilen müşteri verisinin aktarımının yapılabilmesi için API içerisinin müşteri kaydetme işlevini sağlayacak bir fonksiyon olmalıdır.

Aynı şekilde uygulamaların da ESB'ye bir veri gönderebilmesi için ESB'nin de bir uygulama programlama arayüzü sunması gerekmektedir. Bu API'ler bütünleşik sistemdeki adaptörler olarak düşünülebilir. Çift yönlü veri aktarımını sağlayabilmek için mutlaka uygulama programlama arayüzüne ihtiyaç vardır. Şekil 5.4'te bu yapı modellenmiştir.



Şekil 5.4 ESB API Yapısı

API'ler uygulamaların sağlıklı ve kolay şekilde birbirleriyle iletişime geçebilmelerini sağlar. Bu yüzden de son derece açık, net ve gerekli işlevleri içerecek şekilde tasarlanmalıdırlar. API geliştiriminde dokümantasyon da çok önemlidir. Dışarıya açtığımız her bir fonksiyonun ne iş yaptığı, aldığı parametrelerin ne olduğu net şekilde açıklanmalıdır. Dokümantasyon kalitesi, API'nin kalitesini ve kullanılabilirliğini doğrudan etkiler. Düz yazı şeklinde bir dokümantasyon olabileceği gibi, günümüzde bunun çok ötesinde API dokümantasyon araçları geliştirilmiştir. İnteraktif şekilde API'nin test edilebileceği araçlar çok yaygın olmakla birlikte gereksinim haline gelmiştir.

Yukarıda anlatılan süreç adım adım ifade edilirse, akış aşağıdaki şekilde olacaktır.

1. ESB adresinin ve fonksiyon detaylarının bilinmesi.

2. İş fonksiyonlarının, uygulama arayüzünden bağımsız bir katmanda sunulması.

3. İş fonksiyonlarını kullanan API'nin oluşturulması.

4. Uygulamanın kendisini ESB'ye tanıtması.

5. ESB'ye gönderilecek verilerin ESB'nin kabul edeceği formata çevrilmesi.

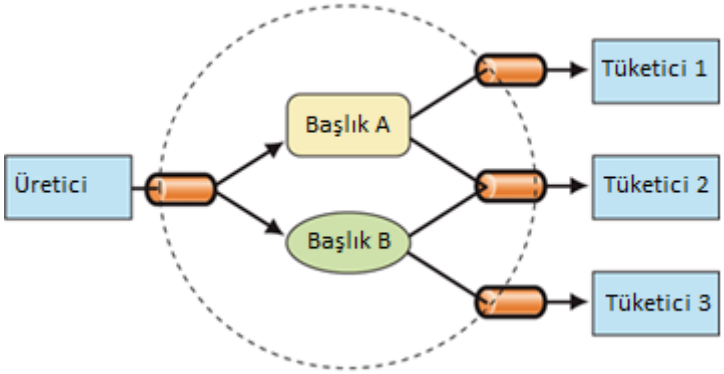
5.2 Üretici/Tüketici Mimari (Publisher/Subscriber Architecture)

Üretici/Tüketici yazılım geliştirme mimarisi, mesajların iletimine dair bir yazılım tasarım desenidir. Son yıllarda oldukça popüler olan bu mimari özellikle büyük ölçekli uygulamalardaki dağıtık ve az çiftlemeli ortama çok iyi uyum sağlamaktadır. Bu tasarım desenini uygulayan sistemlerde, üretici taraf göndereceği veri paketini doğrudan bir başka uygulamaya göndermez ve gönderilmek istenen veri ile ilgilenecek uygulamaları tanımaz. Bunun yerine sadece veriyi gönderir. Veri paketi ile ilgilenen tüketici uygulamalar asenkron olarak veriyi alır ve işler. Paradigmanın birçok versiyonu son zamanlarda önerilmiş olup, her versiyon belirli bir uygulama veya ağ modeli için özel olarak uyarlanmıştır (Eugster et al., 2003).

Yayınlanan verinin filtrelenmesi için iki farklı yol izlenebilir: Başlık bazlı ve içerik bazlı (Baldoni et al., 2003).

Başlık bazlı filtrelemede, yayınlanan mesaj belli bir başlıkta yayımlanır ve bu başlığa üye olmuş bütün tüketicilere iletilir. Bu yapıda mesajın kimlere iletileceğinden üretici sorumludur. Şekil 5.5'te başlık tabanlı filtrelemeye ait bir model gösterilmektedir (Başlık Bazlı Filtreleme, 2017).

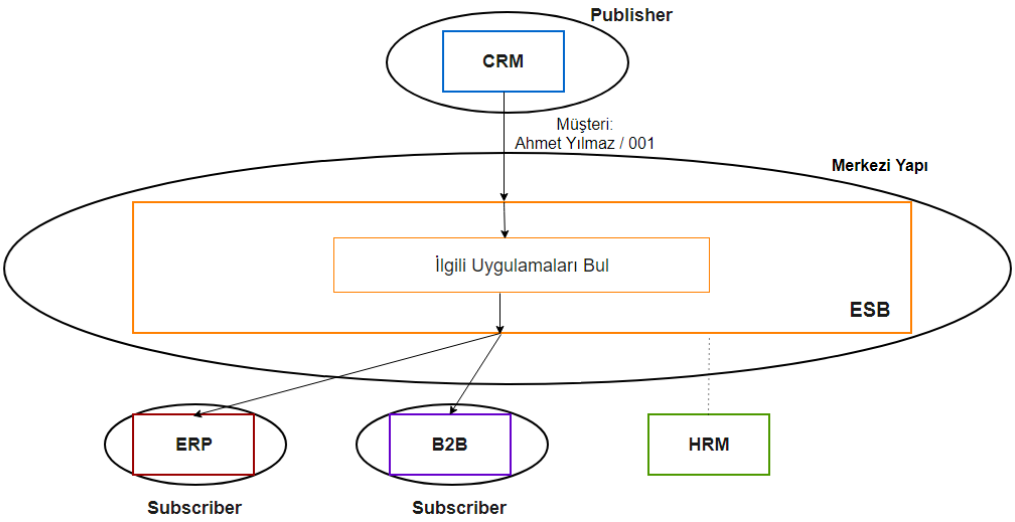
İçerik bazlı filtrelemede ise, üretici herhangi bir işaretleme olmaksızın veriyi iletir. Tüketici verinin içeriğine belli filtreler uygulayarak alıp almayacağına karar verir. Bu yapıda hangi mesajın alınacağı sorumluluğu tüketiciye aittir (Baldoni et al., 2003).



Şekil 5.5 Başlık Bazlı Filtreleme

Bu tez kapsamında önerilen modelde de başlık bazlı üretici/tüketici yapısı kullanılmıştır. Verinin girildiği uygulama sadece veriyi ESB'ye göndermekle sorumludur. Bu verinin hangi sistemlere nasıl iletileceğiyle ilgilenmez. Yeni bir müşteri kaydı girildiğinde, 'Yeni Müşteri' başlığında müşteri verisini ESB'ye gönderir ve bütün işlem o noktada, o uygulama için tamamlanmış olur. Bir uygulama farklı noktalardan başlayan veri aktarımı akışında ya üretici ya da tüketici rolünü üstlenir. Bu yapı uygulamalar arasındaki bağımlılığı düşürerek, ölçülebilirliği arttırır. Birebir iletişimden farklı olarak bu veri akışını sağlayacak merkezi bir yapıya ihtiyaç vardır.

Önerilen modelde CRM'den müşteri girişi yapılan senaryoyu ele alırsak, CRM üretici rolünü, ERP ve B2B ise tüketici rolünü üstlenir. Kurumsal servis kanalı ise bunları dağıtmakla yükümlü merkezi yapıdır. Şekil 5.6'da bu yapının önerilen mimariye nasıl oturduğu gösterilmiştir.



Şekil 5.6 Üretici/Tüketici Mimari

5.3 REST

Yukarıda bahsedilen servis mimarisinin gerçeklenmesinde REST tabanlı web servislerinden yararlanılmıştır. REST, network üzerinde çalışan uygulamaları (Web servislerini) birbirine bağlamak için geliştirilmiş olan mimari bir tasarım stilidir. CORBA (Common Object Request Broker Architecture) ve SOAP'tan farklı şekilde direkt olarak HTTP protokolünü kullanır (Mirgorod, 2013). SOAP tabanlı klasik web servislerinin basit ama güçlü bir alternatifi olarak görülmektedir. CRUD (Create Read Update Delete) operasyonlarının gerçekleştirilmesinde (PUT, GET, POST, DELETE) gibi basit ve anlaşılır HTTP fiillerinden yararlanılır.

JSON ve XML veri formatlarını destekleyen REST servisleri üzerinde istemcilerin ve uçların geliştirilmesi, dokümantasyonun anlaşılması, SOAP tabanlı servislere göre daha kolay olmaktadır. Bununla birlikte ölçekleme sorgu boyutlarının SOAP'a göre çok daha küçük olması ve önbellekleme olanakları sayesinde daha yüksek verimlilik sağlamaktadır. Mumbaikar ve Padiya'nın çalışmasında (Mumbaikar and Padiya, 2013), SOAP ve REST tabanlı servisler üzerinde performans karşılaştırmaları yapılmaktadır. Aşağıda Tablo 5.1'de ilgili çalışma kapsamında incelenen SOAP/REST için mesaj boyutu ve gecikme zamanı karşılaştırması gösterilmektedir.

Tablo 5.1 Mobil hesaplama için SOAP ve REST performans karşılaştırması (Mumbaikar and Padiya, 2013)

Number of array elements	Message Size (byte)				Time (Milliseconds)			
	SOAP/HTTP		*REST (HTTP)*		*SOAP/HTTP*		*REST (HTTP)*	
	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition	String Concatenation	Float Numbers Addition
2	351	357	39	32	781	781	359	359
3	371	383	48	36	828	781	344	407
4	395	409	63	35	828	922	359	375
5	418	435	76	39	969	1016	360	359

REST'in performans getirilerine karşın WS-Security, WS-AtomicTransactions ve WS-ReliableMessaging gibi protokoller üzerinde desteği bulunmamaktadır ve benzer desteklerin sunucu/istemci uygulamalar üzerinde ilave olarak gerçeklenmesi gerekmektedir.

SOAP ve REST servislerin çağrımı arasındaki farkı görmek için örnek bir uygulama üzerinden gidelim (Sketh and Thirunarayan, 2013). Bir telefon rehberi uygulamasından, verilen kullanıcının bilgilerini çekmek istediğimizde oluşturacağımız SOAP isteği Şekil 5.7'deki gibi, REST isteği ise Şekil 5.8'deki gibi olmalıdır.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

Şekil 5.7 SOAP İstek Örneği

```
http://www.acme.com/phonebook/UserDetails/12345
```

Şekil 5.8 REST İstek Örneği

Bu istekler arasında karşılaştırma yaptığımızda REST kullanımının SOAP'a göre çok daha basit olduğunu ve mesaj boyutunun çok daha küçük olduğunu görebiliriz.

Bu tez kapsamında performansla ilgili gereksinimler sebebiyle REST tercih edilmiştir.

6. TEZ KAPSAMINDA GELİŞTİRİLEN UYGULAMA

Tezin uygulama geliştirme kısmında, daha önceki bölümlerde anlatılan modeli gerçekleyen bir prototip üretilmeye çalışılmıştır. Uygulama içerisinde 5 adet proje oluşturulmuştur. Bunlardan biri merkezde konumlandırılacak olan ESB projesidir. İş yazılımlarını örneklemek için ise CRM ve B2B uygulamaları seçilmiştir. Hem CRM hem de B2B uygulamaları için birer adet MVC projesi, birer adet de API projesi oluşturulmuştur.

6.1 Kullanılan Teknolojiler

Bu tez kapsamında uygulama geliştirilirken, geliştirme çerçevesi olarak .Net Framework, programlama dili olarak da C# tercih edilmiştir. Uygulama aracı olarak Visual Studio 2015, veritabanı olarak Microsoft Sql Server kullanılmıştır.

ESB projesi gerçekleştiriminde REST mimari kullanımı uygun bulunmuş ve ve bunun için .Net Framework içerisindeki Web API projesi kullanılmıştır. Aynı şekilde CRM API ve B2B API projeleri de aynı proje tipinde oluşturulmuştur.

CRM ve B2B projelerinin ekranlarının bulunduğu projeler için ise Asp.Net Web Application MVC 5 kullanılmıştır. Model View Controller yapısının sağlamış olduğu katmanlı yapıdan yararlanılarak müşteri girişi ve güncellemesi yapılabilen web sayfaları tasarlanmıştır. Arayüz kısmında Razor View Engine kullanılmıştır. Razor, Asp.Net Mvc 3 teknolojisiyle birlikte çıkan arayüz görüntüleme biçimidir.

REST mimarisi ile hazırlanan projelerin en önemli noktalarından biri de dokümantasyondur. Bu amaçla geliştirilmiş birçok araç mevcuttur. Bu tez kapsamında Swagger aracı kullanılmıştır. Swagger, kod içerisindeki standarda uygun açıklama satırlarını baz alarak, API dokümantasyonu sağlamakta ve bize web sayfası üzerinden iş fonksiyonlarını test edebilme imkanı sunmaktadır.

6.2 ESB Projesi

ESB projesi Web API projesi olarak oluşturulmuştur. Böylece, REST mimariye sahip bir web servis olarak tasarlanmıştır. İçerisinde diğer uygulamaların kendilerini kayıt etme ve kayıt silme işlemlerini, aynı zamanda müşteri kaydetme ve güncelleme işlemlerini yapabilecekleri fonksiyonlar barındırmaktadır.

Bir uygulama ESB'ye kaydolurken; ismini, tekil bir tanımlayıcı bilgisini, dışarı açtığı API bilgisini ve bu API üzerinde ESB'ye ait hangi arayüzleri gerçeklediğini gönderir. Bu bilgiler ESB'ye geldiğinde daha sonra kullanılmak üzere veritabanındaki SUBSCRIBER tablosuna kaydedilir. Kayıtlanma işlemi için yazılmış olan örnek bir kod parçası Şekil 6.1'de gösterilmektedir.

```
1 reference | 0 özel durumlar
internal static void SubscribeToEsb()
{
    var esbAddress = ConfigurationManager.AppSettings["EsbService"];

    var dto = new SubscribeDto();
    dto.AppId = EsbUtils.GetProductId();
    dto.Name = "CRM";
    dto.BaseAddress = "http://localhost:2962/";
    dto.MethodList = new Dictionary<string, string>()
    {
        { "SaveCustomer", "api/Home/SaveCustomer" },
        { "UpdateCustomer", "api/Home/UpdateCustomer" }
    };
    dto.ImplementedInterfaces = new List<string>()
    {
        "IHaveCustomer"
    };

    Exception internalExp;
    Task.Run(async () =>
    {
        try
        {
            using (HttpClient httpClient = new HttpClient())
            {
                httpClient.BaseAddress = new Uri(esbAddress);
                var result = await httpClient.PostAsJsonAsync("/api/home/subscribe", dto);
                string resultContent = await result.Content.ReadAsStringAsync();
                Console.WriteLine(resultContent);
            }
        }
        catch (Exception e)
        {
            internalExp = e;
        }

    });
}
```

Şekil 6.1 Kayıtlanma Kod Parçası

Kayıtlı uygulamalardan biri müşteri kaydetmek için ESB'nin ilgili fonksiyonunu çağırdığında, bu istek ile ilgili detaylar LOG tablosuna kaydedilir. Sonrasında bu isteğin gitmesi gereken uygulamalar bulunarak, uygulamaların paylaştığı API bilgileri üzerinden çağrımlar yapılır. Her bir çağrım işlemi ile ilgili log, TARGETOPERATIONLOG tablosuna kaydedilir. Eğer herhangi bir sebeple bu çağrımlardan bazıları başarısız olursa, ESB belli zaman aralıklarında başarısız çağrımları tekrar göndermeyi dener.

Bir uygulamadan müşteri kaydetme üzerine bir istek geldiğinde ESB üzerinde yapılan işlemler Şekil 6.2'deki örnek kod bloğunda görülmektedir.

```
public bool SaveCustomer(CustomerDto customer)
{
    var clients = Manager.Instance().ClientApiList.Where(m => m.ImplementedInterfaces.Contains("IHaveCustomer"));
    string currentMethod = "SaveCustomer";

    Dictionary<ClientApiDto, Exception> exceptionDict = CallForAllClients(currentMethod, customer, clients);

    if (exceptionDict.Count > 0)
        return false;
    else
        return true;
}
```

Şekil 6.2 Müşteri Kaydetme Kod Parçası

Yukarıdaki görsel detayında görülen ve asıl çağrımları yapan fonksiyon içeriği ise Şekil 6.3'te gösterilmektedir.

```
private static Dictionary<ClientApiDto, Exception> CallForAllClients(string currentMethod, IEsbDTO customer, IEnumerable<ClientApiDto> clients)
{
    clients = clients.Where(m => m.AppId != customer.AppId);
    int logId = Manager.Instance().AddLog(currentMethod, customer, clients);

    Dictionary<ClientApiDto, Exception> exceptionDict = new Dictionary<ClientApiDto, Exception>();
    foreach (var client in clients)
    {
        Exception e;
        Manager.Instance().CallClientApi(currentMethod, customer, client, out e);
        if (e != null)
        {
            Manager.Instance().AddClientOperationLog(logId, currentMethod, customer, client, OPERATIONSTATUS.Error, e.Message);
            exceptionDict.Add(client, e);
        }
        else
        {
            Manager.Instance().AddClientOperationLog(logId, currentMethod, customer, client, OPERATIONSTATUS.Success,string.Empty);
        }
    }

    return exceptionDict;
}
```

Şekil 6.3 Servis İletimi Kod Parçası

Dışarıdan ulaşılabilecek fonksiyonların isimleri aşağıdaki gibidir.

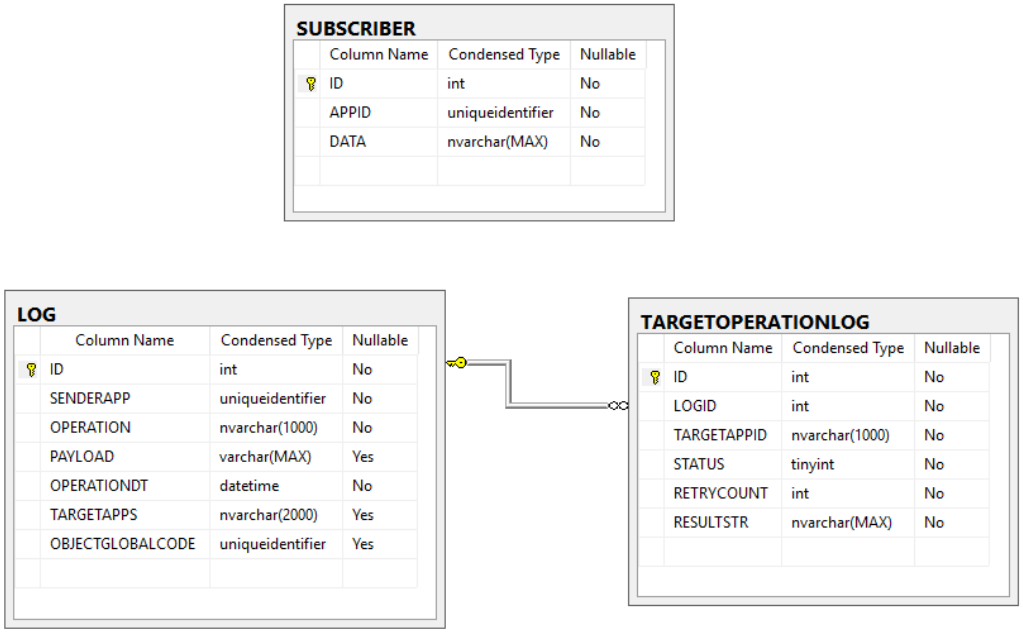
- **Subscribe:**
  Dış uygulamaların ESB'ye kendilerini tanıtmak için kullandığı fonksiyondur. İlgili parametreler ile çağrım yapıldığında uygulama bütünleşik sistemin bir parçası haline gelmektedir.

- **Unsubscribe:**
  Dış uygulamaların ESB'de bulunan kayıtlarını silmeleri için dışarıya açılmış olan fonksiyondur.

- **SaveCustomer**:
  Mimarinin basit modellemesinde müşteri kaydı baz alınacağından, bu işlemi yapacak olan bir fonksiyon eklenmiştir. Gönderilen müşteri kaydını, ilgili diğer uygulamalara gönderir.

- **UpdateCustomer**:
  Zaten girilmiş olan bir müşterinin güncelleme işleminin yapılabilmesi için yazılmış olan fonksiyondur. Kaydetme işleminde olduğu gibi güncelleme kaydını ilgili uygulamalara gönderir.

Şekil 6.4'te veritabanı diyagramı ESB'nin kullandığı tabloları göstermektedir.



Şekil 6.4 ESB Veritabanı Diyagramı

Diyagramda görülen tabloların veri içeren görselleri Şekil 6.5, Şekil 6.6 ve Şekil 6.7'de gösterilmektedir. Kolonların tuttuğu bilgilerin açıklaması şekillerin alt kısmında yer almaktadır.

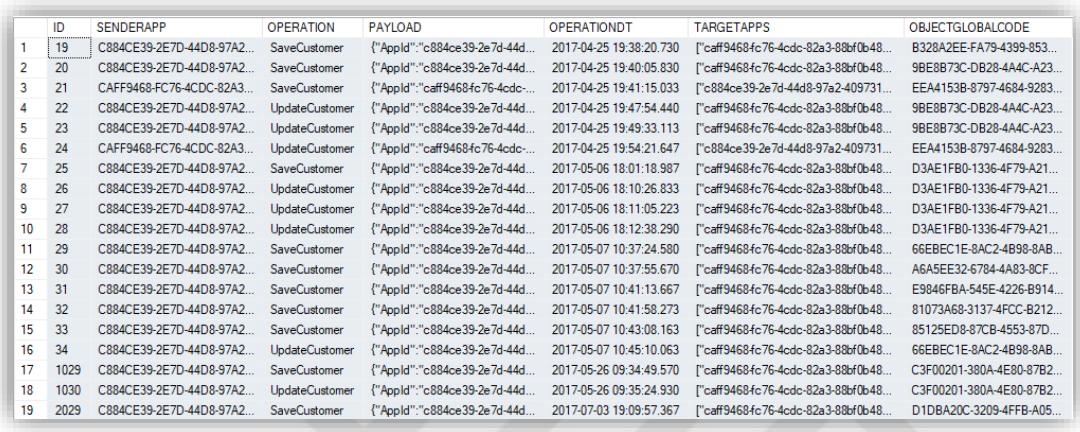
	ID	APPID	DATA
1	2029	C884CE39-2E7D-44D8-97A2-409731161693	{"AppName":"CRM","BaseAddress":"http://localhost:2962/","Implemented...
2	2030	CAFF9468-FC76-4CDC-82A3-88BF0B48A703	{"AppName":"B2B","BaseAddress":"http://localhost:2192/","ImplementedI...

Şekil 6.5 SUBSCRIBER Tablosu Veri Örneği

ID: Tabloya özel tanımlayıcı bilgisi tutulmaktadır.

APPID: ESB'ye kaydolan uygulamaların tekil tanımlayıcı bilgisi tutulmaktadır.

DATA: Uygulamanın ESB'ye kaydolurken ilettiği bilgiler XML formatında tutulmaktadır.



	ID	SENDERAPP	OPERATION	PAYLOAD	OPERATIONDT	TARGETAPPS	OBJECTGLOBALCODE
1	19	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-04-25 19:38:20.730	["caff9468-fc76-4cdc-82a3-88bf0b48...	B328A2EE-FA79-4399-853...
2	20	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-04-25 19:40:05.830	["caff9468-fc76-4cdc-82a3-88bf0b48...	9BE8B73C-DB28-4A4C-A23...
3	21	CAFF9468-FC76-4CDC-82A3...	SaveCustomer	{"AppId":"caff9468-fc76-4cdc-...	2017-04-25 19:41:15.033	["c884ce39-2e7d-44d8-97a2-409731...	EEA4153B-8797-4684-9283...
4	22	C884CE39-2E7D-44D8-97A2...	UpdateCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-04-25 19:47:54.440	["caff9468-fc76-4cdc-82a3-88bf0b48...	9BE8B73C-DB28-4A4C-A23...
5	23	C884CE39-2E7D-44D8-97A2...	UpdateCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-04-25 19:49:33.113	["caff9468-fc76-4cdc-82a3-88bf0b48...	9BE8B73C-DB28-4A4C-A23...
6	24	CAFF9468-FC76-4CDC-82A3...	UpdateCustomer	{"AppId":"caff9468-fc76-4cdc-...	2017-04-25 19:54:21.647	["c884ce39-2e7d-44d8-97a2-409731...	EEA4153B-8797-4684-9283...
7	25	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-06 18:01:18.987	["caff9468-fc76-4cdc-82a3-88bf0b48...	D3AE1FB0-1336-4F79-A21...
8	26	C884CE39-2E7D-44D8-97A2...	UpdateCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-06 18:10:26.833	["caff9468-fc76-4cdc-82a3-88bf0b48...	D3AE1FB0-1336-4F79-A21...
9	27	C884CE39-2E7D-44D8-97A2...	UpdateCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-06 18:11:05.223	["caff9468-fc76-4cdc-82a3-88bf0b48...	D3AE1FB0-1336-4F79-A21...
10	28	C884CE39-2E7D-44D8-97A2...	UpdateCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-06 18:12:38.290	["caff9468-fc76-4cdc-82a3-88bf0b48...	D3AE1FB0-1336-4F79-A21...
11	29	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-07 10:37:24.580	["caff9468-fc76-4cdc-82a3-88bf0b48...	66EBEC1E-8AC2-4B98-8AB...
12	30	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-07 10:37:55.670	["caff9468-fc76-4cdc-82a3-88bf0b48...	A6A5EE32-6784-4A83-8CF...
13	31	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-07 10:41:13.667	["caff9468-fc76-4cdc-82a3-88bf0b48...	E9846FBA-545E-4226-B914...
14	32	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-07 10:41:58.273	["caff9468-fc76-4cdc-82a3-88bf0b48...	81073A68-3137-4FCC-B212...
15	33	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-07 10:43:08.163	["caff9468-fc76-4cdc-82a3-88bf0b48...	85125ED8-87CB-4553-87D...
16	34	C884CE39-2E7D-44D8-97A2...	UpdateCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-07 10:45:10.063	["caff9468-fc76-4cdc-82a3-88bf0b48...	66EBEC1E-8AC2-4B98-8AB...
17	1029	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-26 09:34:49.570	["caff9468-fc76-4cdc-82a3-88bf0b48...	C3F00201-380A-4E80-87B2...
18	1030	C884CE39-2E7D-44D8-97A2...	UpdateCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-05-26 09:35:24.930	["caff9468-fc76-4cdc-82a3-88bf0b48...	C3F00201-380A-4E80-87B2...
19	2029	C884CE39-2E7D-44D8-97A2...	SaveCustomer	{"AppId":"c884ce39-2e7d-44d...	2017-07-03 19:09:57.367	["caff9468-fc76-4cdc-82a3-88bf0b48...	D1DBA20C-3209-4FFB-A05...

Şekil 6.6 LOG Tablosu Veri Örneği

ID: Tabloya özel tanımlayıcı bilgisi tutulmaktadır.

SENDERAPP: ESB'ye gelen isteğin hangi uygulama tarafından gönderildiği bilgisi tutulmaktadır.

OPERATION: Uygulamaya gelen isteğin tipi tutulmaktadır.

PAYLOAD: ESB'ye gelen istek tüm detaylarıyla XML formatında tutulmaktadır.

OPERATIONDT: İsteğin geldiği gün ve saat tutulmaktadır.

TARGETAPPS: İsteğin ESB tarafından hangi uygulamalara iletileceği bilgisi yer almaktadır.

OBJECTGLOBALCODE: Gönderilen nesnenin ESB sisteminde anlamlı olan tekil tanımlayıcı bilgisi tutulmaktadır.

	ID	LOGID	TARGETAPPID	STATUS	RETRYCOUNT	RESULTSTR
1	12	20	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
2	13	21	c884ce39-2e7d-44d8-97a2-409731161693	0	0	
3	14	22	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	13	
4	15	23	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	4	
5	16	24	c884ce39-2e7d-44d8-97a2-409731161693	0	0	
6	17	25	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
7	18	26	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
8	19	28	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
9	20	29	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
10	21	30	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
11	22	31	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
12	23	32	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
13	24	33	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
14	25	34	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
15	1020	1029	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
16	1021	1030	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
17	2020	2029	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	
18	2021	2030	c884ce39-2e7d-44d8-97a2-409731161693	0	0	
19	2022	2031	caff9468-fc76-4cdc-82a3-88bf0b48a703	0	0	

Şekil 6.7 TARGETLOG Tablosu Veri Örneği

ID: Tabloya özel tanımlayıcı bilgisi tutulmaktadır.

LOGID: ESB'ye gelen isteğe ait logun tanımlayıcı bilgisi tutulmaktadır.

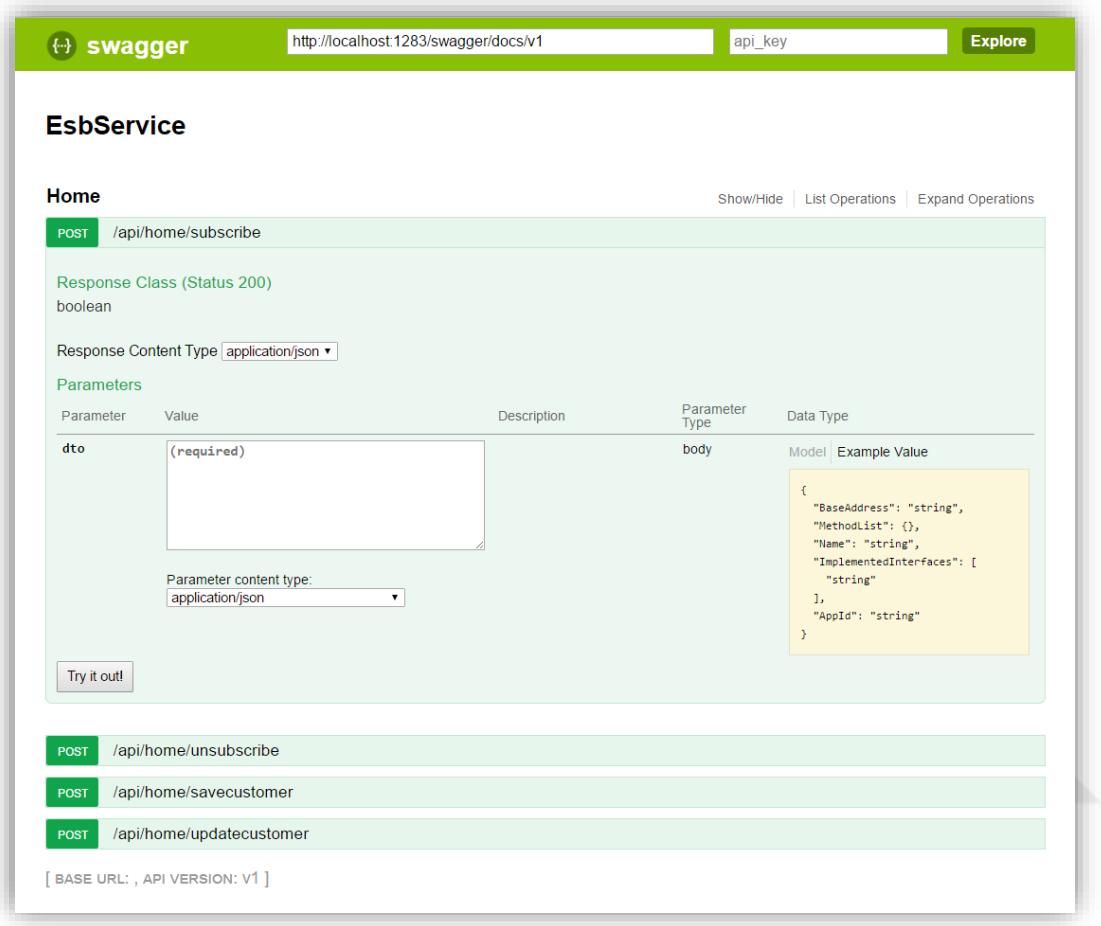
TARGETAPPID: İsteğin iletileceği uygulamaya ait bilgi tutulmaktadır.

STATUS: İletimin başarılı olup olmadığı bilgisi tutulmaktadır. Değer '1' olduğunda iletim başarısız, '0' olduğunda başarılı anlamına gelmektedir.

RETRYCOUNT: İsteğin kaç kez iletilmeye çalışıldığı bilgisi tutulmaktadır.

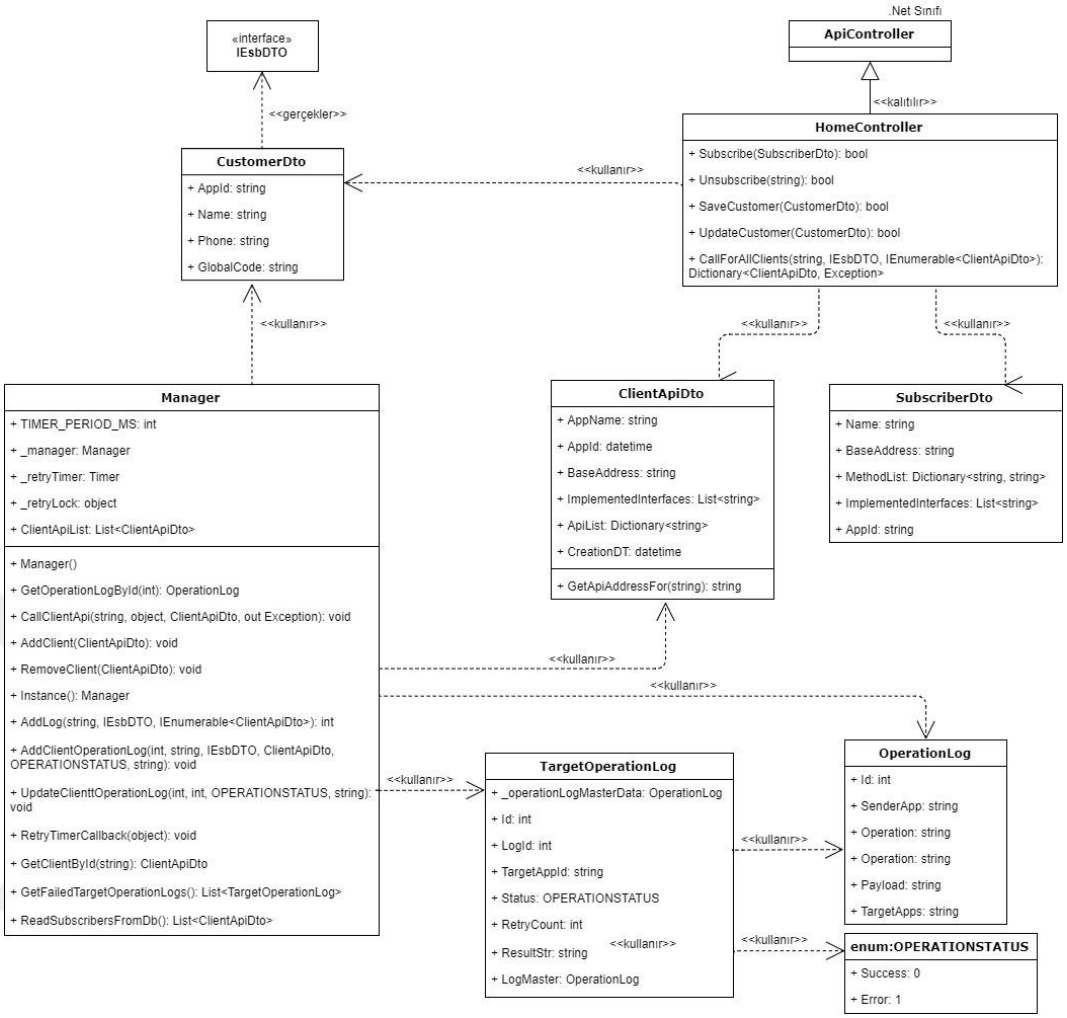
RESULTSTR: Başarısız iletim denemeleri sırasında dönen hata mesajlarını barındırmaktadır.

Swagger kullanılarak oluşturulan test sayfasının ekran görüntüsü ise Şekil 6.8'de gösterilmektedir. Servisin yayın yaptığı bağlantı adresinin sonuna '/swagger' ifadesi eklenirse bu sayfaya ulaşılacaktır.



Şekil 6.8 ESB Swagger Görseli

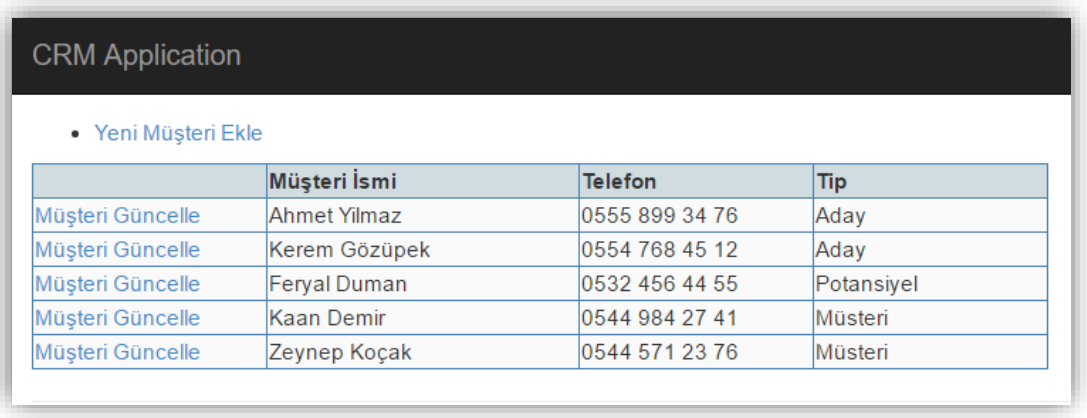
Bu projeye ait sınıf diyagramı Şekil 6.9'da gösterilmektedir.



Şekil 6.9 ESB Projesi Sınıf Diyagramı

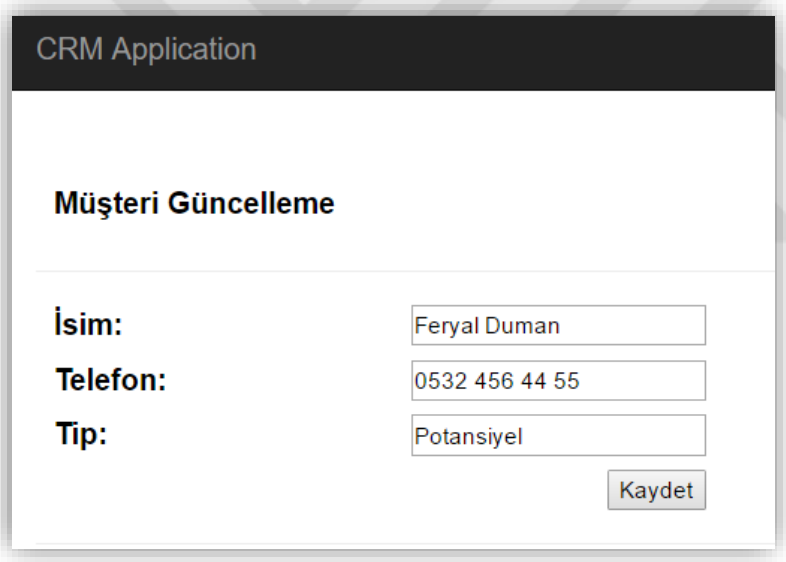
6.3 CRM Projesi (MVC)

Bu web uygulaması CRM uygulamasını temsil etmesi için oluşturulmuştur. Müşteri kayıtlarını listeleyen, yeni müşteri kaydının girilip, var olan müşteri kaydının güncellenebildiği ekranlar içermektedir. Uygulama açıldığında müşterilerin bulunduğu bir listenin ve yeni müşteri eklenebilecek bir bağlantının bulunduğu Şekil 6.10'daki ekran açılır.



Şekil 6.10 CRM Müşteri Listeleme Ekranı

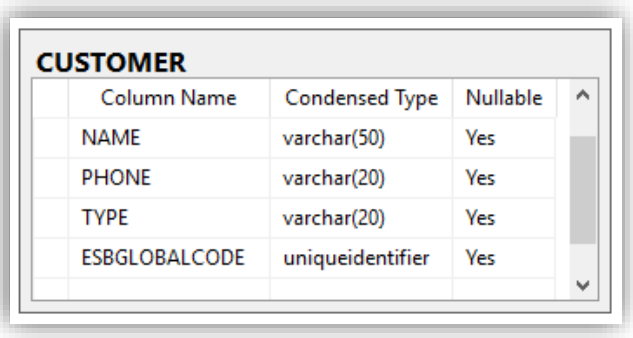
Listede yer alan müşterilerin güncellenebileceği ekranlara özgü bağlantılar da Şekil 6.11'deki ekranda yer almaktadır.



Şekil 6.11 CRM Müşteri Güncelleme Ekranı

Kaydedilen müşteriler CUSTOMER isimli tabloda tutulur. CUSTOMER tablosu uygulamaya ait kolonlar dışında, mevcut kurguda ESB bütünleştirmesi için gerekli olan ESBGLOBALCODE isimli bir kolon barındırmaktadır.

Kullanılan tablonun detayı Şekil 6.12'de gösterilmektedir.



Şekil 6.12 CRM Veritabanı Diyagramı

Tablonun veri içeren görsel Şekil 6.13'te gösterilmektedir. Kolonların tuttuğu bilgilerin açıklaması şeklin alt kısmında yer almaktadır.

	ID	NAME	PHONE	TYPE	ESBGLOBALCODE
1	34	Ahmet Yilmaz	0555 899 34 76	Aday	66EBEC1E-8AC2-4B98-8ABC-E153A7301F53
2	35	Kerem Gözüpek	0554 768 45 12	Aday	A6A5EE32-6784-4A83-8CFA-2BCFB92C7B14
3	36	Feryal Duman	0532 456 44 55	Potansiyel	E9846FBA-545E-4226-B914-11BE76B3B033
4	37	Kaan Demir	0544 984 27 41	Müsteri	81073A68-3137-4FCC-B212-DC59957FAA2F
5	38	Zeynep Koçak	0544 571 23 76	Müsteri	85125ED8-87CB-4553-87D0-7B15BAF14635
6	1034	Onur Ünlü	0532 456 44 57	Potansiyel	C3F00201-380A-4E80-87B2-62749D9FDEBD
7	2034	Hakki Erden	0555 756 3414	Potansiyel	D1DBA20C-3209-4FFB-A05A-544F20144325
8	2035	Arzu Simsek	0555 787 3456	Aday	20A95FBE-7771-4ABC-8A58-3A7E36F5437B
9	2036	Murat Komesli	0555 687 3434	Potansiyel	4DF0F1E4-FCC0-4E68-B308-C3012E3A3962

Şekil 6.13 CUSTOMER Tablosu Veri Örneği

ID: Tabloya özel tanımlayıcı bilgisi tutulmaktadır.

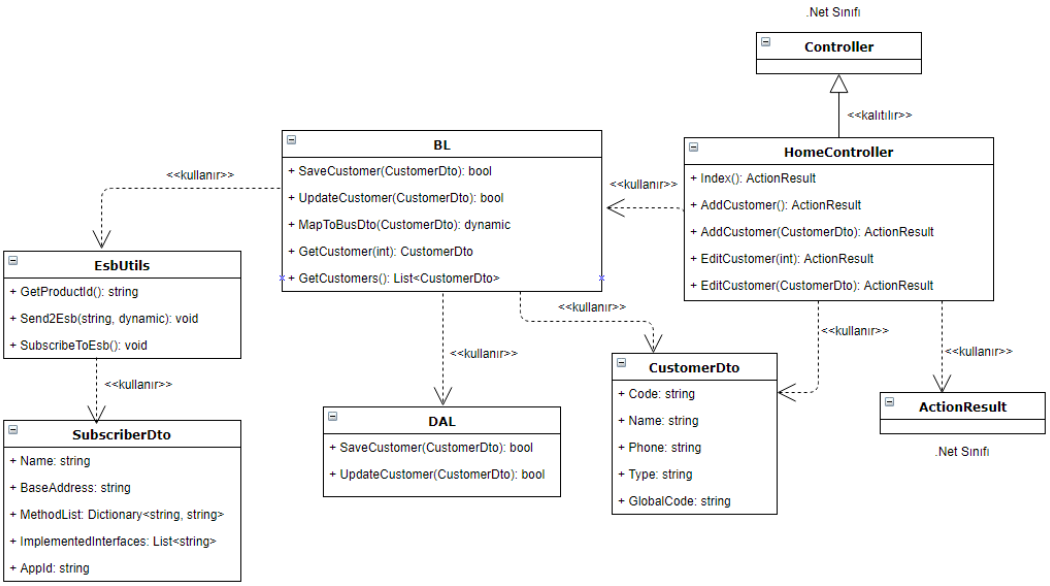
NAME: Müşteri ismi tutulmaktadır.

PHONE: Müşterinin telefon numarası tutulmaktadır.

TYPE: Müşterinin tipi tutulmaktadır.

ESBGLOBALCODE: ESB sistemi içinde kayda tekillik kazandıran bilgi tutulmaktadır.

Bu projeye ait sınıf diyagramı Şekil 6.14'te gösterilmektedir.

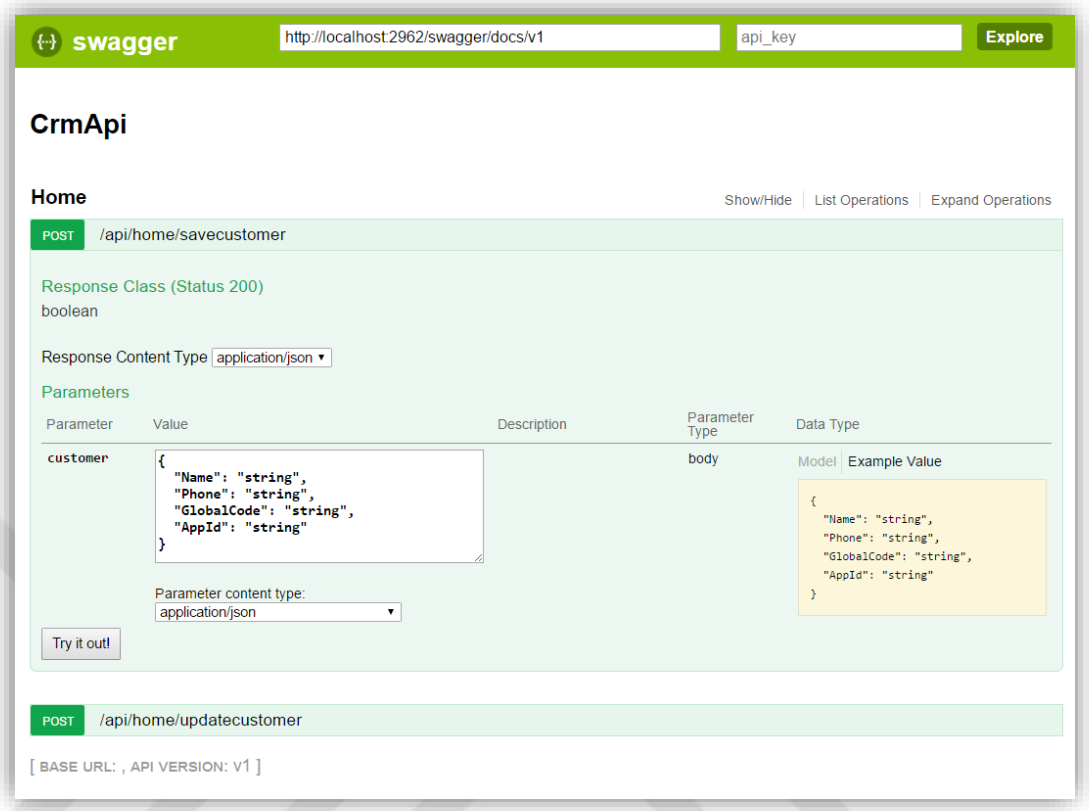


Şekil 6.14 CRM MVC Projesi Sınıf Diyagramı

6.4 CRM Projesi (API)

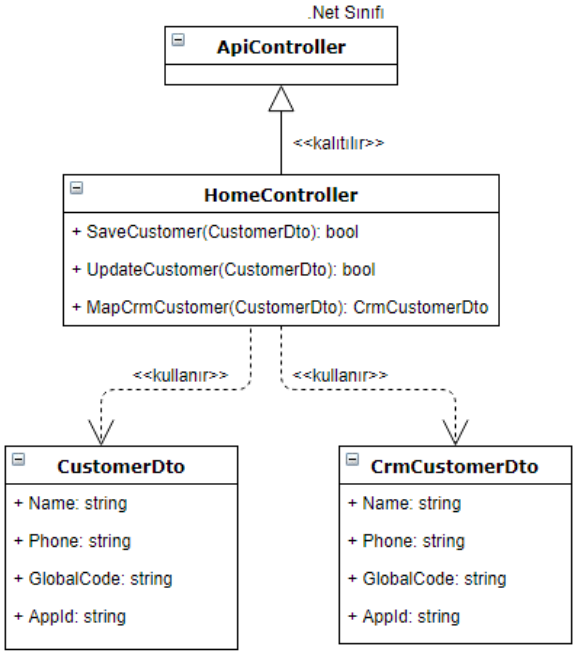
Bu web servisi ESB tarafından gelebilecek olan müşteri kaydetme ve güncelleme işlemleri için fonksiyonlar içerir. Bu fonksiyonlar gelen veriyi kendi yapısına çevirerek veritabanınında gerekli işlemleri gerçekleştirir.

Swagger kullanılarak elde edilen test sayfasının ekran görüntüsü 6.15'teki gibidir.



Şekil 6.15 CRM API Swagger

Bu projeye ait sınıf diyagramı Şekil 6.16'da gösterilmektedir.

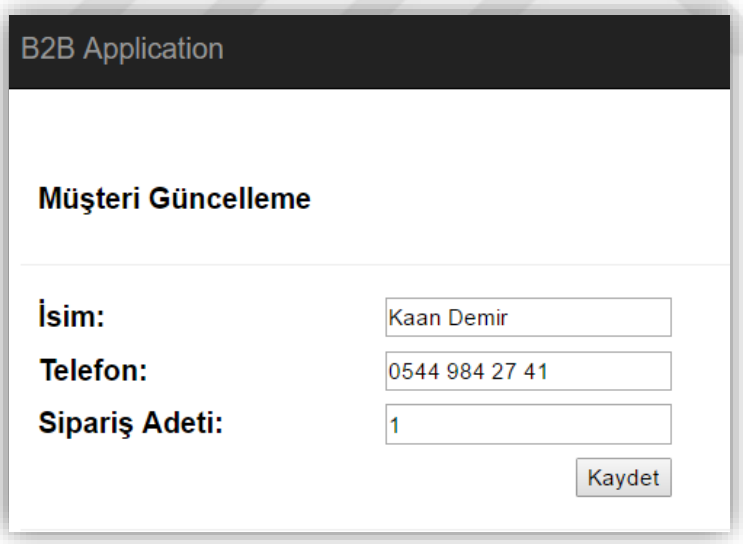


Şekil 6.16 CRM API Projesi Sınıf Diyagramı

6.5 B2B Projesi (MVC)

Bu web uygulaması B2B ugyulamasını temsil etmesi için oluşturulmuştur. CRM uygulamasında olduğu gibi, müşteri kayıtlarını listeleyen, yeni müşteri kaydının girilip, varolan müşteri kaydının güncellenebildiği Şekil 6.17'deki ve Şekil 6.18'deki ekranları içermektedir.

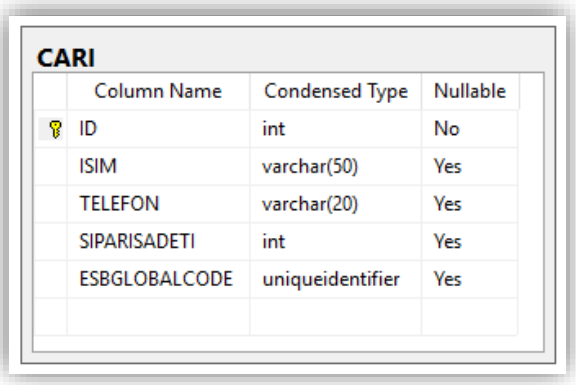
Şekil 6.17 B2B Müşteri Listeleme Ekranı



Şekil 6.18 B2B Müşteri Güncelleme Ekranı

Kaydedilen müşteriler CARI isimli tabloda tutulur. CARI tablosu uygulamaya ait kolonlar dışında, mevcut kurguda ESB bütünleştirmesi için gerekli olan ESBGLOBALCODE isimli bir kolon barındırır.

Kullanılan tablonun detayları Şekil 6.19'da gösterilmektedir.



Şekil 6.19 B2B Veritabanı Diyagramı

Tablonun veri içeren görseli Şekil 6.20'de gösterilmektedir. Kolonların tuttuğu bilgilerin açıklaması şeklin alt kısmında yer almaktadır.

	ID	ISIM	TELEFON	SIPARISADETI	ESBGLOBALCODE
1	6	Ahmet Yilmaz	0555 899 34 76	1	66EBEC1E-8AC2-4B98-8ABC-E153A7301F53
2	7	Kerem Gözüpek	0554 768 45 12	0	A6A5EE32-6784-4A83-8CFA-2BCFB92C7B14
3	8	Feryal Duman	0532 456 44 55	2	E9846FBA-545E-4226-B914-11BE76B3B033
4	9	Kaan Demir	0544 984 27 41	1	81073A68-3137-4FCC-B212-DC59957FAA2F
5	10	Zeynep Koçak	0544 571 23 76	5	85125ED8-87CB-4553-87D0-7B15BAF14635
6	11	Onur Ünlü	0532 456 44 57	0	C3F00201-380A-4E80-87B2-62749D9FDEBD
7	1011	Hakki Erden	0555 756 3414	0	D1DBA20C-3209-4FFB-A05A-544F20144325
8	1012	Arzu Simsek	0555 787 3456	0	20A95FBE-7771-4ABC-8A58-3A7E36F5437B
9	1013	Murat Komesli	0555 687 3434	0	4DF0F1E4-FCC0-4E68-B308-C3012E3A3962

Şekil 6.20 CARI Tablosu Veri Örneği

ID: Tabloya özel tanımlayıcı bilgisi tutulmaktadır.

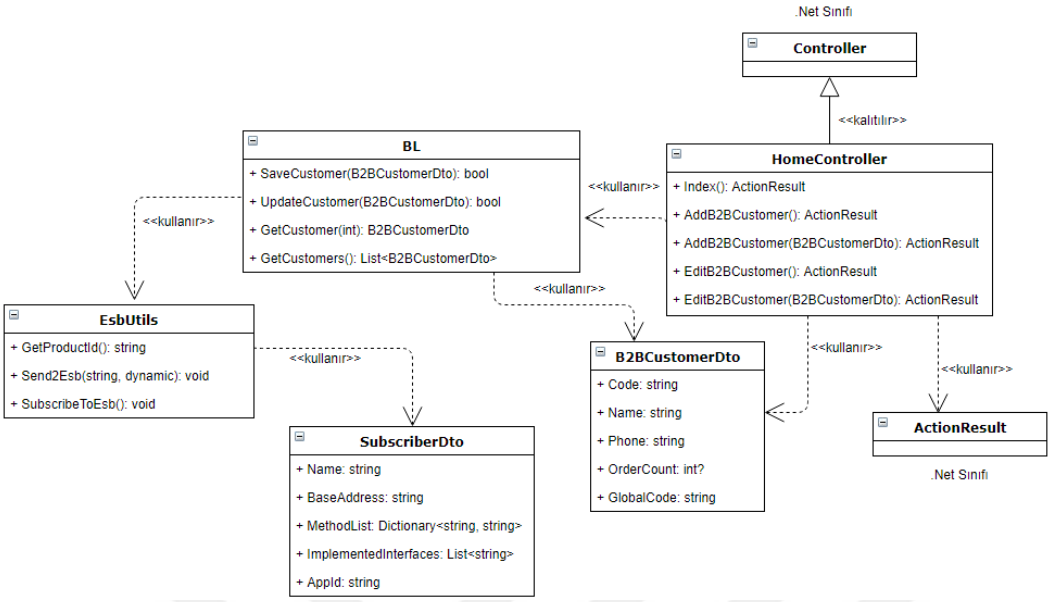
ISIM: Müşteri ismi tutulmaktadır.

TELEFON: Müşterinin telefon numarası tutulmaktadır.

SIPARISADETI: Müşterinin şimdiye kadar vermiş olduğu siparişlerin sayısı tutulmaktadır.

ESBGLOBALCODE: ESB sistemi içinde kayda tekillik kazandıran bilgi tutulmaktadır.

Bu projeye ait sınıf diyagramı Şekil 6.21'de gösterilmektedir.



Şekil 6.21 B2B MVC Projesi Sınıf Diyagramı

6.6 B2B Projesi (API)

Bu web servisi de yine CRM API'de olduğu gibi ESB tarafından gelebilecek olan müşteri kaydetme ve güncelleme işlemleri için fonksiyonlar içerir. Bu fonksiyonlar gelen datayı kendi yapısına çevirerek veritabanınında gerekli işlemleri gerçekleştirir.

ESB tarafından çağrı geldiğinde 'Controller' seviyesindeki Şekil 6.22'deki fonksiyona düşer.

```
public bool SaveCustomer(EsbCustomerDto customer)
{
    bool success = BL.SaveCustomer(customer);
    return success;
}
```

Şekil 6.22 B2B API Müşteri Kaydetme Kod Parçası - Controller

Sonrasında iş katmanına (business layer) geçilir. Öncelikle gelen veri, eşleyici (mapper) ile B2B veri yapısına çevrilir. Daha sonra ise DAL katmanına gidilerek verinin kaydedilmesi gerçekleştirilir. Bu işlemlere ait kod parçaları Şekil 6.23'te ve Şekil 6.24'te gösterilmektedir.

```
internal static bool SaveCustomer(EsbCustomerDto customer)
{
    try
    {
        CustomerDto b2bCustomer = MapB2BCustomer(customer);
        return DAL.SaveCustomer(b2bCustomer);
    }
    catch (Exception)
    {
        return false;
    }
}
```

Şekil 6.23 B2B API Müşteri Kaydetme Kod Parçası – BL

```
private static CustomerDto MapB2BCustomer(EsbCustomerDto customer)
{
    CustomerDto b2bCustomer = new CustomerDto();
    b2bCustomer.Isim = customer.Name;
    b2bCustomer.Telefon = customer.Phone;
    b2bCustomer.GlobalCode = customer.GlobalCode;
    return b2bCustomer;
}
```

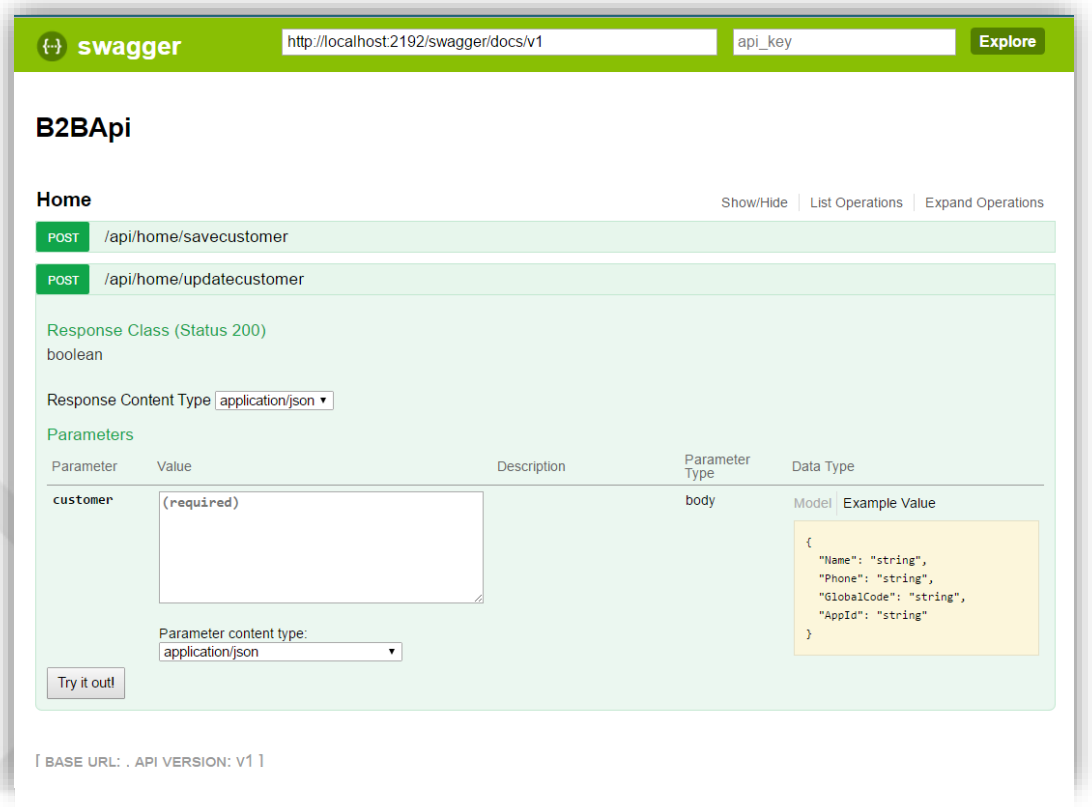
Şekil 6.24 B2B API Eşleyici Kod Parçası

Geliştirme sırasında katmanlı yapıya özen gösterilmiş ve uygulamanın kolay değiştirilebilir olması amaçlanmıştır. DAL (Data Access Layer – Veri Erişim Katmanı) ‘da sql yazılarak veritabanına kayıt işlemi yapılmaktadır. Bu işlemle ilgili kod parçası Şekil 6.25’te gösterilmektedir. Bu kısımda NHibernate, Entity Framework gibi birçok farklı teknoloji kullanılabilir.

```
public static bool SaveCustomer(CustomerDto customer)
{
    try
    {
        string connStr = ConfigurationManager.ConnectionStrings["connection"].ConnectionString;

        SqlConnection conn = new SqlConnection(connStr);
        SqlCommand cmd = new SqlCommand();

        cmd.CommandText = string.Format("INSERT INTO CARI (ISIM, TELEFON, ESBGLOBALCODE) VALUES('{0}', '{1}', '{2}')",
                                        customer.Isım, customer.Telefon, customer.GlobalCode);
        cmd.CommandType = CommandType.Text;
        cmd.Connection = conn;

        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
        return true;
    }
    catch (Exception)
    {
        return false;
    }

}
```

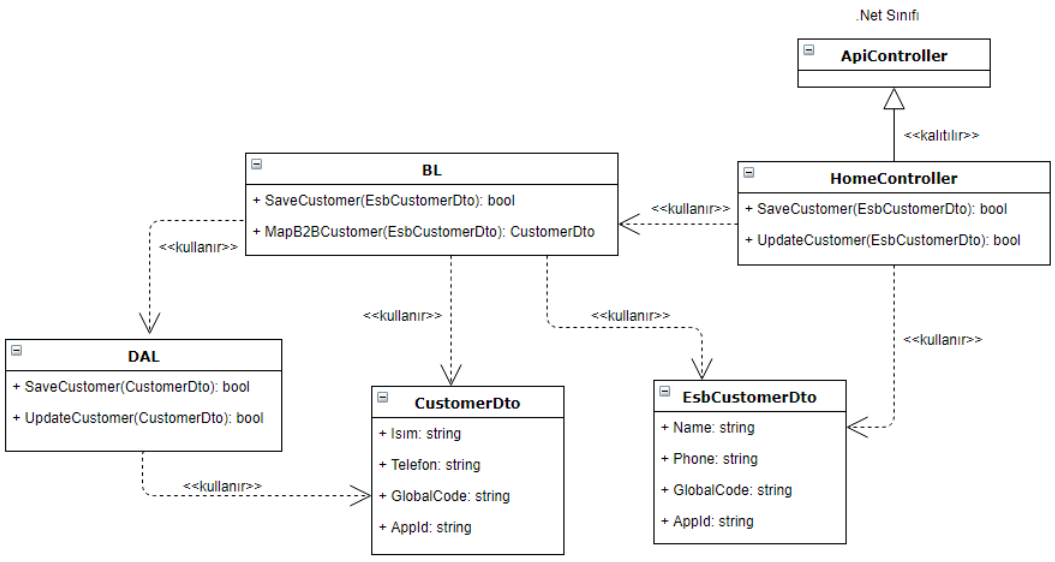
Şekil 6.25 B2B API Müşteri Kaydetme Kod Parçası - DAL

Swagger kullanılarak elde edilen test sayfasının ekran görüntüsü Şekil 6.26'daki gibidir.



Şekil 6.26 B2B API Swagger Görseli

Bu projeye ait sınıf diyagramı Şekil 6.27'de gösterilmektedir.

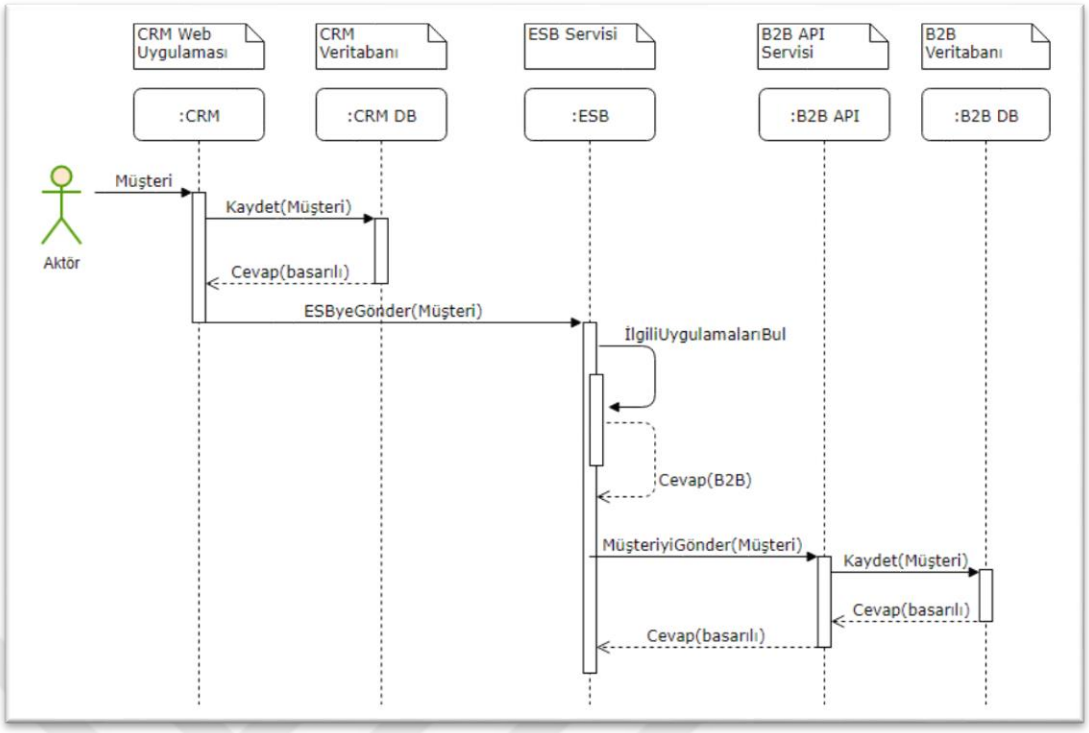


Şekil 6.27 B2B API Projesi Sınıf Diyagramı

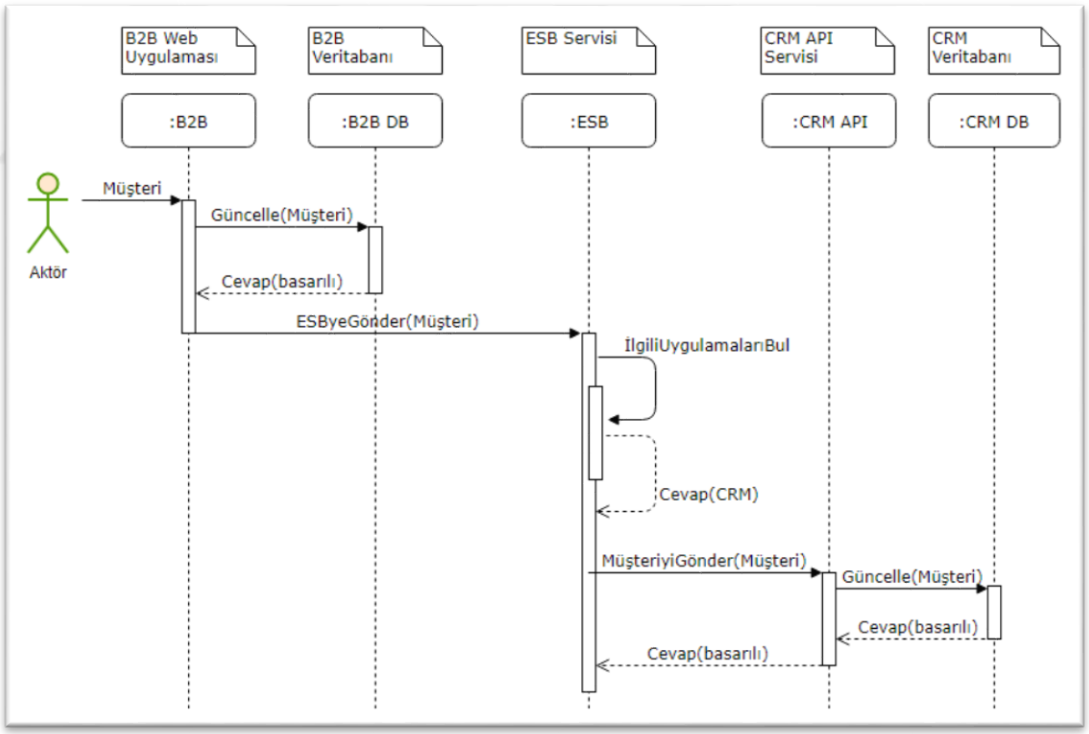
6.7 Akış Diyagramları

Geliştirilen uygulamada gerçekleştirilebilen senaryolardan en önemli ikisi ele alınarak akış diyagramları oluşturulmuştur. Bunlardan ilki CRM uygulaması üzerinden girilen bir müşteri kaydının ESB aracılığıyla B2B veritabanı üzerinde de oluşturulması durumudur. Bu senaryoyla ilgili çizilen akış diyagramı Şekil 6.28'de gösterilmektedir.

Ele alınan ikinci senaryoda ise B2B üzerinden yapılan müşteri güncellemesinin yine ESB aracılığıyla CRM veritabanı üzerinde de güncellenmesi gerçekleştirilmektedir. Bu işleme ait akış diyagramı ise Şekil 6.29'da gösterilmektedir.



Şekil 6.28 Müşteri Kaydı Akış Diyagramı



Şekil 6.29 Müşteri Güncelleme Akış Diyagramı

7. TARTIŞMA VE ÖNERİLER

Bu çalışma kapsamında uygulamalar arası veri aktarımı ele alınmıştır ve ESB mimarisi kullanılmıştır. ESB kullanmadığımız durumda da yine veri aktarımı sağlanabilir. Aşağıda alternatif iki yöntem anlatılmış ve önerilen yöntemle karşılaştırılmıştır.

Veritabanı Düzeyinde Bütünleştirme (Alternatif 1): Konu veri aktarımı olduğunda, bütünleştirmeyi veritabanı düzeyinde yapmak bir seçenek olarak sunulabilir. Örneğin; CRM veritabanına bir müşteri kaydı girildiğinde bir tetikleyici (trigger) yardımıyla B2B veritabanına yazılabilir. Ancak tetikleyicileri iş akışının temelindeki yapılar için kullanmak çeşitli problemlere yol açabileceği gibi performansla ilgili sorunlara da sebep olabilir. Bunun yanı sıra veritabanları farklı sunucularda yer alıp birbirlerine ulaşamıyor olabilirler. Artık şirketlerin güvenlikle ilgili birçok kurala sahip olduğu düşünülürse, sunucunun dışarı açılması engellendiği anda bütün bütünleştirme yapısı çökmüş olacaktır. Bu yüzden uygulamaları bu denli temel katmanlardan direkt bağlamak problem yaracaktır. Eğer birçok uygulamanın bu şekilde bağlandığı düşünülürse, bağlantılarda problem olmasa dahi yönetimi çok zor olacaktır.

Uygulamalar Arası Birebir Bütünleştirme (Alternatif 2): Eğer veritabanı katmanından çıkılıp uygulama katmanında birebir bütünleştirme ele alınırsa, iki uygulama arasında gayet başarılı bir bağlantı kurulabilir. Örneğin; CRM uygulaması üzerinden girilen bir kayıt, B2B uygulamasının API'sini kullanarak veriyi karşı uygulamaya aktarabilir. Bu durumda B2B uygulamasına ait bilgileri, API adresini, beklediği veri yapısını bilmesi gerekmektedir. Bu bilgiler edinildiğinde, bütünleştirme için bu bilgilere dayalı bir kodlama işlemi yapılması gerekir. Kaydedilecek verileri B2B veri yapısına göre düzenlenmesi gerekmektedir. Ancak uygulama sayısının arttığı bir örnek düşünülürse, B2B bütünleştirmesi için yapılan bütün işlemlerin her entegre olunacak uygulama için yapılması gerekmektedir. Beş adet uygulama olduğu varsayılırsa, bu beş uygulamanın servis bilgilerinin bilinmesi, verinin ayrı ayrı uygulamalara uygun veri yapılarına getirilmesi gerekmektedir. Bu da temelde aynı işlemi yapan kod bloklarının defalarca yazılması anlamına gelmektedir. Bu yapıda, eğer yeni bir uygulamayla bütünleştirme söz konusu olursa bu, kaynak kodun değiştirilme zorunluluğunu doğurmaktadır. Mevcut bütünleştirmelerden biri için diğer uygulamanın sağladığı serviste herhangi bir değişiklik olursa ve uygulama bu değişikliğe göre güncellenmezse, mevcutta var

olan bütünleştirme yapısı bir anda çalışamaz hale gelebilir. Bu durumda bütünleştirmenin bozulmasından uygulamayı gönderen taraf sorumlu olacaktır.

Yukarıda iki farklı alternatif senaryo incelendikten sonra, bu çalışma kapsamında sunulan modelin birçok konu için çözüm sağladığı rahatlıkla söylenebilir. Önerilen yapıda uygulamaların birbirlerini değil, sadece ESB'yi tanıma gereksinimi olduğundan uygulama sayısı ne kadar artarsa artsın, yeni bir uygulamanın bütünleştirmesi için diğer uygulamalar bazında bir değişiklik yapmaya gerek kalmayacaktır. Uygulamalar birbirlerinden soyutlanmış olduğundan bir yerde yapılan bir değişikliğin bütünleştirmeyi bozma ihtimali minimuma düşürülmüş olur. Önerilen yapıda birçok konu için sorumluluk ESB'de olacağından, veri paylaşacak bir uygulamanın sorumluluk alanı, veriyi başarılı şekilde ESB'ye ulaştırdığı anda sonlanmaktadır.

Uygulamalar üzerindeki sorumluluğun azalması, yukarıda da bahsedildiği gibi ESB üzerindeki yükü arttıracaktır. Bu durumda ESB geliştirimi sırasında birçok durumun düşünülmesi ve çözümlenmesi gerekmektedir. Uygulamanın sorumluluğu ESB'ye ilettiği anda biterken, veriyi ilgili tüm servislere düzgün şekilde iletmek ESB'nin sorumluluğudur. Birçok farklı sebeple gönderimlerde problem oluşabilir. Bu yüzden ESB gerçekleştirimi sırasında tekrar deneme (retry) yapısının mutlaka olması gerekmektedir.

Bu çalışma kapsamında kullanılan REST mimarisinin birçok yönden pozitif etkisi olmasına rağmen, durum bazlı negatif etkileri de görülebilir. Eğer birbiriyle bağımlı, ardışık (transactional) yapıda çok işlem yapılıyorsa REST mimarisinde bunun üstesinden gelinebilmesi için ekstra geliştirim yapılması gerekecektir.

ESB kullanımı birçok açıdan fayda sağlaması ve kodlama açısından tekrarları ortadan kaldırmasının yanında, bütünleştirme tek bir ortak noktada toplandığından problem yaratabilir. Eğer bir sebeple ESB çalışamaz duruma gelirse bu durumda 'n' adet uygulamanın birbirleriyle olan iletişimi kopacaktır. Bu tarz durumlara örnek olarak, ESB'nin bir ayna görüntüsü oluşturulmalı ve biri çökerse diğeri devreye girerek akışı devam ettirmelidir. Bu da, ayna görüntüsüne sahip iki uygulamanın bakımı ve eşit düzeyde tutulması maliyetini getirecektir.

ESB, bu çalışma kapsamında odak noktası olarak birçok iş uygulaması söz konusu olduğundan uygun görülmüştür. Ancak ESB kullanımı, gereksinime göre avantaj ve dezavantajları düşünülerek seçilmelidir.

Tablo 7.1'de yukarıda anlatılan alternatif yöntemler ve bu tez kapsamında sunulan model belirli özellikler açısından karşılaştırılmaktadır. Karşılaştırma için seçilen özelliklerin kısa açıklamaları aşağıdaki şekildedir:

Veritabanı Bağımlılığı: Entegre olunan uygulamanın veritabanında bir değişiklik olduğunda bütünleştirmenin hangi düzeyde etkileneceğini belirten özelliktir.

Model Bağımlılığı: Entegre olunan uygulamada, uygulama bazında bir model değişikliği olduğunda bütünleştirmenin hangi düzeyde etkileneceğini belirten özelliktir.

Uygulama Sorumluluğu: Uygulamadan gönderilen verinin karşı uygulamaya ulaşıp ulaşmadığının takibini yapma düzeyini belirten özelliktir.

Ağ Konfigürasyon Maliyeti: Uygulamaların birbirlerine bağlanabilmesi için gerekli olan ağ erişilebilirlik düzeyini belirten özelliktir.

Yönetim Zorluğu: Uygulama sayısı arttıkça karşılaşılan yönetim zorluk düzeyini belirten özelliktir.

Bütünleştirme Maliyeti: Uygulama sayısı arttıkça bütün uygulamalarla entegre olmak için harcanacak eforu belirten özelliktir.

Tablo 7.1 Yöntem karşılaştırma tablosu

	Alternatif 1 (Veritabanı Bazında)	**Alternatif 2 (Uygulamalar Arası 1-1)**	**ESB ile Bütünleştirme**
Veritabanı Bağımlılığı	Yüksek	Orta	Düşük
Model Bağımlılığı	Düşük	Yüksek	Düşük
Uygulama Sorumluluğu	Yüksek	Yüksek	Düşük
Ağ Konfigürasyon Maliyeti	Yüksek	Yüksek	Düşük
Yönetim Zorluğu	Yüksek	Yüksek	Orta
Bütünleştirme Maliyeti	Yüksek	Yüksek	Orta

8. GELECEK ÇALIŞMALAR

Bu tez kapsamında var olan uygulamaların minimum düzeyde etkilenmesi için verinin ESB üzerinden aktarılması üzerinde durulmuştur. Ancak ESB ortak bir noktada olduğundan birçok farklı şekilde kullanılabilir.

Yeni oluşturulacak ortak bir kavram için her bir uygulamada verinin ayrı ayrı tutulmasındansa, merkezi bir veritabanında tutularak ESB üzerinden ulaşılıp sorgulanması sağlanabilir. Örneğin; birden fazla uygulamaya yeni modüller eklendiğini ve bu modüllerin kavramsal olarak ‘araç’ kavramını kullandığını düşünelim. Bu yapıda, modüller tasarlanırken ‘araç’ kavramıyla ilgili kısımları ESB üzerinde kurgulanabilir. Her uygulama ortak bir noktada duran veriye ESB üzerinden erişerek kullanabilir.

ESB, bize birçok noktadan veri toplayarak bir özet rapor ekranı hazırlamamıza da yardımcı olabilir. Konumunun merkeziliği gereği birçok uygulama ile bağlı olduğundan ortak bir kavrama farklı açılardan bakarak bu kavramları birleştirebiliriz. Mevcut bir müşteriyi ele alırsak; CRM uygulamasından bu müşteriye yapılan aktivite kayıtlarını, B2B uygulamasından vermiş olduğu siparişleri, ERP uygulamasından da kendisine kesilmiş faturaları ESB aracılığıyla alıp tek bir ekranda gösterilebilir.

Mevcut çalışmada, yapılacak iş temel alınarak, amaca yönelik fonksiyonlar oluşturulmuştur. Eğer işlemler bazında başlıklar dinamik hale getirilirse, çok daha esnek ve dinamik bir yapı kurulabilir. Böylece yeni bir işlem seti eklenmek istendiğinde minimum tanımlamalarla koda müdahale etmeden veri bazlı eklemeler yeterli olabilir.

9. SONUÇLAR

Bu tez çalışmasında iş uygulamaları arasındaki veri aktarımıyla ilgili problemler ele alınmış ve bu problemlerin çözümü için bir mimari model önerilmiştir. Çalışmanın çıkış noktası, sektöründe öncü bir yazılım şirketi bünyesindeki uygulamaların birbirleriyle iletişim kurma ihtiyacı ve bu konuda karşılaşılan zorluklardır. Mevcut sistemde yazılımlar birebir bağlantılarla iletişim kurabilmektedir. Ancak uygulama sayısı arttıkça birebir iletişim kurabilme maliyeti de büyük oranda artmaktadır. Şirket bünyesine yeni bir yazılım eklendiğinde, var olan bütün uygulamaların iletişim kurabilmesi için uygulama sayısı kadar ekstra geliştirmeye ihtiyaç vardır. Bu sorunların çözümü için ESB tabanlı bir mimari sunulmuştur.

ESB, uygulamaların ve servislerin birbirleriyle ilişkilerinin soyutlanarak, iletişimin ortak bir servis kanalı üzerinden yürümesini amaçlayan bir mimaridir. Bu mimarinin temelleri SOA'ya dayandığında SOA'nın da çok iyi şekilde bilinmesi ve anlaşılması gerekmektedir.

Önerilen mimaride iş uygulamalarının ortasında konumlandırılacak bir ESB yer almaktadır. Bütün uygulamalar sadece ESB ile iletişime geçebilmekte ve diğer uygulamaları tanıma zorunluluğu olmamaktadır.

İş uygulamaları kendilerini ESB'ye tanıtır ve bütünleşik sisteme girmiş olurlar. ESB üzerinde veri alışverişi yapabilmeleri için ayrıca bir API oluşturmaları gerekmektedir. Bu API, ESB'nin uygulamalarla iletişim kurmasını sağlamaktadır.

Uygulamalardan ESB'ye herhangi bir kayıt isteği geldiğinde, ESB bu veriyi almak için uygun uygulamaları bularak iletir. Bu iletim API'ler sayesinde gerçekleşmektedir. Eğer bu gönderimlerden başarısız olanlar olursa, kendi log mekanizmasını kullanarak başarılı olana kadar tekrar denemektedir. Bu yapı, iletilecek olan verinin sağlığı için eklenmiş olup, herhangi bir nedenle bir kez hata aldığında sürecin sonuçlanmaması için önemlidir.

API gerçekleştiriminde REST mimari tercih edilmiştir. HTTP protokolü kullandığından, çağrımlar çok daha basit ve kolay olmaktadır. Bunun dışındaki en önemli tercih sebebi ise performans açısından diğer teknolojilerin önüne çıkmasıdır. Uygulama API'lerinin yanı sıra ESB de REST mimaride tasarlanmıştır.

Tez kapsamında önerilen mimariyi gerçekleyen bir uygulama da yazılmıştır. ESB işlevini görecek bir servis, CRM ve B2B iş yazılımlarını simüle eden iki web uygulaması ve bu web uygulamalarının API'leri olmak üzere beş projeden oluşmaktadır. Web uygulamaları öncelikle kendilerini ESB'ye kayıt ettirerek hangi işlemler için sisteme katılmak istediklerini ve kendi API'leriyle ilgili bilgileri gönderirler. Sonrasında yapılmak istenen işlem için ESB'ye istek gönderildiğinde, ESB kendisine kayıt olan diğer uygulamalar arasından uygun olanları bulup gerekli dağıtımı yapar. Geçekleştirilen uygulamada CRM üzerinden kayıt edilen bir müşterinin ESB üzerinden geçerek B2B'ye de kaydedilmesi sağlanabilmektedir. Sisteme ERP'nin dahil edilmesidurumunda, hem CRM hem de B2B için bir adaptöre yazılması yerine, sadece ESB ile iletişim kuracak bir adaptör yazılması yeterli olmaktadır. Böylece bütünleştirme maliyetleri ciddi düzeyde azaltılmış olur.

Uygulama geliştirme kısmında büyük oranda Microsoft teknolojileri ve araçlarından yararlanılmıştır. İş uygulamalarını modelleyen yazılımlar web tabanlı olarak tasarlanmış ve güncel teknolojiler kullanılmaya çalışılmıştır. RESTful API dokümantasyonu için Swagger isimli bir yazılım aracından yararlanılmıştır. Bu sayede API fonksiyonlarının kullanımları ve parametreleri görsel bir web sayfası olarak dokümante edilmiş olup, aynı zamanda test edilebilir hale gelmiştir.

KAYNAKLAR DİZİNİ

Alaa, M.R., Ahmed, E.H., Qusay, H., 2010, Design of SOA-based Grid Computing with Enterprise Service Bus, International Journal on Advances in Information Sciences and Service Sciences, 2(1): 71-82pp.

Apache, "Apache ServiceMix", http://servicemix.apache.org/ (Erişim Tarihi: 15 Mart 2017)

Baldoni, R., Contenti, M. and Virgillito, A., 2003, The Evolution of Publish/Subscribe Communication Systems, Future Directions of Distributed Computing, 2584, Berlin.

Baude, F., Filali, I., Huet, F., Legrand, V., Mathias, E., Merle, P., Ruz, C., Krummenacher, R., Simperl, E., Hammerling, C., Lorre, J., 2010, ESB Federation for Large-Scale SOA, Proceedings of the 2010 ACM Symposium on Applied Computing, 2459-2466pp.

Buschle, M., Ekstedt, M., Grunow, S., Hauder, M., Matthes, F. and Roth, S., 2012, Automating Enterprise Architecture Documentation using an Enterprise Service Bus, AMCIS 2012 Proceedings, 13.

Chappel, D.A., 2004, Therory In Practice Enterprise Service Bus, O'Reilly Media., 276p.

Dai, P., 2011, Design and implementation of ESB based on SOA in power system, Electric Utility Deregulation and Restructuring and Power Technologies (DRPT), 2011 4th International Conference

Eugster, P., Felber, P., Guerraoui, R. and Kermarrec, A., 2003, The many faces of publish/subscribe, *ACM Computing Surveys CSUR*, 35(2):114-131 pp.

Fiorano, "Fiorano ESB", http://www.fiorano.com/products/esb-enterprise-service-bus/ (Erişim Tarihi: 15 Mart 2017)

Hartman, P., 2008, ESB Enablement of an International Corporate Acquisition, an Experience Report, 2008. ICGSE 2008. IEEE International Conference on Global Software Engineering.

He, W., Xu, L.D., 2014, Integration of Distributed Enterprise Applications: A Survey, IEEE Transactions on Industrial Informatics, 10(1).

KAYNAKLAR DİZİNİ (devam)

IBM Knowledge Center, "IBM WebSphere ESB", https://www.ibm.com/support/knowledgecenter/en/SSQH9M_7.0.0/com.ibm.websphere.wesb.doc/doc/welcome_wps_ovw.html (Erişim Tarihi: 14 Mart 2017)

Iona Software Solutions Inc, "IONA", http://www.iona-solutions.com/ (Erişim Tarihi: 15 Mart 2017)

JBoss, "JBoss ESB", http://jbossesb.jboss.org/ (Erişim Tarihi: 15 Mart 2017)

Jing, F., Cong, X., Lirong, X., 2012, Data transformation of ESB based on the data model for integration, 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery.

Junye, W., Lirui, M., Hongming, C., 2009, A REST-Based Approach to Integrate Enterprise Resources, Computer Science-Technology and Applications, 2009. IFCSTA '09. International Forum.

Kurniawan, K. and Ashari, Ahmad., 2015, Service Orchestration using Enterprise Service Bus for Real-time Government Executive Dashboard System, 2015 Internatioanl Conference on Data and Software Engineering.

Liu, Y., Gorton, I. and Zhu, L., 2007, Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus, nComputer Software and Application Conference, 2007. COMPSAC 2007. 31. Annual International.

Marechaux, J., 2006, Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus, https://www.ibm.com/developerworks/library/ws-soa-eda-esb/

Microsoft, "Microsoft BizTalk Server", https://www.microsoft.com/en-us/cloud-platform/biztalk (Erişim Tarihi: 14 Mart 2017)

Mirgorod, V., 2013, Backbone.Js Cookbook, Packt Publishing Ltd., 282p.

MSDN, "Başlık Bazlı Filtreleme", https://msdn.microsoft.com/en-us/library/ff649664.aspx#despublishsubscribe_problem (Erişim Tarihi: 6 Haziran 2017)

KAYNAKLAR DİZİNİ (devam)

MuleSoft, "Mule ESB", https://www.mulesoft.com/resources/esb/what-mule-esb (Erişim Tarihi: 15 Mart 2017)

Mumbaikar S. and Padiya, P., 2013, Web Services Based On SOAP and REST Principles, International Journal of Scientific and Research Publications, 3

Oracle, "Oracle Enterprise Service Bus", http://www.oracle.com/technetwork/middleware/service-bus/overview/index.html (Erişim Tarihi: 14 Mart 2017)

Progress, "Progress Sonic ESB", https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/dvesb/sonic-esb.html, (Erişim Tarihi: 14 Mart 2017).

S-Cube Topluluğu, "S-Cube Araştırma Alanları", http://www.s-cube-network.eu/ (Erişim tarihi: 5 Nisan 2016)

Shi, L, Wang, G., 2011, The Analysis of a Data Transformation Method Based on ESB Internet Technology and Applications (iTAP), 2011 International Conference

Sketh, A. and Thirunarayan, K., 2013, Semantics Empowered Web 3.0: Managing Enterprise, Social, Sensor, and Cloud Based Data and Services for Advanced Applications.

SmartBear Software Inc., "Swagger", https://swagger.io/ (Erişim Tarihi: 3 Ocak 2017)

Szydło, T., Zieliński, K., 2012, Adaptive Enterprise Service Bus, New Generation Computing (2012), 20(2):189-214pp.

TIBCO Software Inc., "Tibco Software", https://www.tibco.com/ (Erişim Tarihi: 15 Mart 2017).

Wang, G., Fung, C., 2004, Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems, 37. Hawaii International Conference on System Sciences.

KAYNAKLAR DİZİNİ (devam)

Ziyaeva, G., Choi, E. and Min, D., 2008, Content-Based Intelligent Routing and Message Processing in Enterprise Service Bus, 2008. ICHIT '08. International Conference on Convergence and Hybrid Information Technology.

ÖZGEÇMİŞ

Genel

Adı Soyadı :	Beril KAVAKLI		
Yazışma Adresi:	Bilgisayar Müh. Bölümü, Ege Üniversitesi, Bornova, İzmir		
Doğum Tarihi ve Yeri:	26.01.1989, Milas		
Tel:	2327659199	**GSM:**	5546672420
E-Posta:	**berilkavakli@gmail.com**	**Faks:**	2327659199

Eğitim

Öğrenim Dönemi	Derece (*)	Üniversite	Öğrenim Alanı
2011 – …	Yüksek Lisans	Ege Üniversitesi	Bilgisayar Mühendisliği
2007 – 2011	Lisans	Ege Üniversitesi	Bilgisayar Mühendisliği

Akademik ve Mesleki Deneyim

Görev Dönemi	Unvan	Kurum	Bölüm
2011 – Devam Ediyor	Kıdemli Yazılım Uzmanı	Logo Yazılım Netsis Yazılım	Yazılım Geliştirme