ÜÇ-SIRALI İSTEMCİ-SUNUCU MİMARİSİNDE WEB SERVİSLERİ KULLANIMI VE UYGULAMASI

Yıldıray Anagün

Yüksek Lisans Tezi

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Şubat - 2007

ÜÇ-SIRALI İSTEMCİ-SUNUCU MİMARİSİNDE WEB SERVİSLERİ KULLANIMI VE UYGULAMASI

Yıldıray Anagün

Dumlupınar Üniversitesi
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Elektrik-Elektronik Mühendisliği Anabilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır.

Danışman : Yrd. Doç. Dr. Ahmet ÖZMEN

Şubat - 2007

KABUL ve ONAY SAYFASI

Yıldıray ANAGÜN'ün YÜKSEK LİSANS tezi olarak hazırladığı "Üç-Sıralı İstemci-Sunucu Mimarisinde Web Servisleri Kullanımı ve Uygulaması" başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

08 / 02 /2007
(Sınav tarihi)

Üye : Yrd.Doç.Dr. Ahmet ÖZMEN

Üye : Doç.Dr. Kaan ERARSLAN

Üye : Yrd.Doç.Dr. Alpaslan DUYSAK

Fen Bilimleri Enstitüsün Yönetim Kurulu'nun ...../...../..... gün ve ..... sayılı kararıyla onaylanmıştır.

.....
Fen Bilimleri Enstitüsü Müdürü

ÜÇ-SIRALI İSTEMCİ-SUNUCU MİMARİSİNDE WEB SERVİSLERİ KULLANIMI VE UYGULAMASI

Yıldıray ANAGÜN
Elektrik-Elektronik Mühendisliği, Yüksek Lisans Tezi, 2007
Tez Danışmanı: Yrd.Doç.Dr. Ahmet ÖZMEN

ÖZET

Bu çalışmada veritabanı sistemleri incelenmiş ve diğerlerine göre daha üstün özelliklere sahip olan üç-sıralı istemci-sunucu yazılım modeli kullanılarak, otobüs seyahat bileti otomasyon uygulaması gerçeklenmiştir. Şu an ülkemizde kullanılan seyahat e-ticaret sistemleri birbirleri ile etkileşim halinde bulunmamaktadır. Bu e-ticaret sistemleri ortak işlem yapma özelliğine sahip değildir ve bu da seyahat edecek müşterilerin en uygun otobüs seyahat biletini bulmada kolaylık sunmamaktadır. Bu tezde e-ticaret sistemini kullanan firmalar arasında veritabanlarının paylaşıma açılmasıyla, bilgi paylaşımına izin veren bir model geliştirilmiş, daha sonra "Locus Travel" adındaki otomasyon sistemi ASP. Net XML web servisleriyle gerçeklenmiştir. Sistem bilet alıcıları için geliştirilmiş web ara yüzüne ve firma çalışanları için geliştirilmiş form tabanlı ara yüze sahiptir. Sistem seçilen kıstaslara göre firmalar arasında kıyaslama ile arama yaptırma ve arama sonuçlarına göre de bilet satın alınması ve rezervasyon sürecini gerçekleştirmektedir.

Anahtar Kelimeler: Üç-sıralı istemci-sunucu yazılım modeli, XML web servisleri, e-ticaret.

APPLICATION AND USING OF WEB SERVICES IN THE THREE-TIER SOFTWARE ARCHITECTURE

Yıldıray ANAGÜN
Electric&Electronic Engineering, M.S.Thesis, 2007
Thesis Supervisor: Asst.Prof.Dr. Ahmet ÖZMEN

SUMMARY

In this thesis, database systems are studied and a bus travel e-commerce application is implemented with using three-tier architecture approach, which has much better in respect of the others. Currently, individual bus travel e-commerce systems are owned and operated by the travel companies in Turkey. These e-commerce systems are non-inter operable which create problems and difficulties for the travellers who use these systems to purchase travel tickets. This thesis presents a system, called "Locus Travel", implemented in ASP.Net platform and used XML web services to allow information sharing between companies already implemented e-commerce systems. The system provides web based user interfaces for customers and form based interfaces for travel agencies. Using the system one can easily make comprehensive search based on the provided criteria, and then buy a ticket from the search results.

Keywords: There-tier architecture, XML web services, e-commerce.

TEŞEKKÜR

Çalışmalarım sırasında yardımlarını hiçbir zaman esirgemeyen danışman hocam Yrd.Doç.Dr. Ahmet ÖZMEN'e, çalışmalarımda beni sabırla destekleyen aileme, her zaman fikirlerine danıştığım bölümümüzün değerli hocalarına, Arş.Gör. Soydan SERTTAŞ, Arş.Gör. Bahadır HİÇDURMAZ, Arş.Gör. Serdar ÖZYÖN, Arş. Gör. Abdullah Sait ŞEKERCİOĞLU Tekniker İsmail BAŞ'a ve isimlerini yazamadığım diğer tüm mesai arkadaşlarıma sonsuz teşekkür ederim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

SİMGELER ve KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
WTO	Dünya Ticaret Örgütü
EFT	Elektronik Fon Transferi
EDI	Elektronik Veri Değişimi
GSM	Mobil İletişim için Küresel Sistem
OECD	Ekonomik İşbirliği ve Gelişim Örgütü
HTTP	Hiper Metin Transfer Protokolü
HTML	Hiper Metin İşaretleme Dili
DDP	Dağıtılmış Veri İşleme
PC	Kişisel Bilgisayar
GUI	Grafiksel Kullanıcı Ara Yüzü
RAD	Hızlı Uygulama Geliştirme
SQL	Yapısal Sorgulama Dili
DBMS	Veritabanı Yönetim Sistemi
RPC	Uzak İşlev Çağırma
LAN	Yerel Alan Ağı
XML	Genişletilebilir Etiketleme Dili
SOAP	Basit Nesne Erişim Protokolü
WSDL	Web Servisi Tanımlama Dili
UDDI	Evrensel Açıklama, Keşif ve Entegrasyon
JSP	Java Server Sayfaları
ASP	Aktif Server Sayfaları
CGI	Ortak Geçit Ara Yüzü
ISAPI	Internet Sunucusu Uygulama Programı Ara Yüzü
B2B	Firmadan FİrmayaXSL Genişletilebilir Stil Dili
IIOP	Internet Inter-ORB Protokolü
RMI	Uzak Metot Çalıştırma
IDL	Ara Yüz Tanımlama Dili
URL	Tekdüzen Kaynak Bulucu
URI	Tekdüzen Kaynak Tanımlayıcı
PKI	Açık Anahtar Altyapısı

SİMGELER ve KISALTMALAR DİZİNİ (devam)

Kısaltmalar	Açıklama
SSL	Güvenli Soket Katmanı
IIS	Internet Bilgi Servisleri
IPSec	Internet Protokol Güvenliği

1. GİRİŞ

Bu çalışmada öncelikle e-ticaretin gelişim süreci, üç-katmanlı istemci-sunucu modeli ve web servisleri mimarisinin çalışma mekanizması detayları ile incelenmiştir. Mimariyi oluşturan yapılar, protokoller ve standartların, gerçeklediğimiz sisteme nasıl uygulandığı açıklanmıştır. Burada, web servis mimarisi geliştirilirken ihtiyaç duyulacak teknoloji ve kavramlara değinilmektedir.

İkinci bölümde, üç-katmanlı model temelinde ASP.Net web servisleri kullanılarak gerçeklediğimiz sistem ele alınmaktadır. Uygulamanın veritabanı paylaşımına imkan vermesinden dolayı sağladığı avantajlar açıklanmıştır. Bu kısımda sistem performansı da deneylerle test edilmiştir. Tezde web servis mimarisi açıklanırken ve uygulama geliştirilirken C# (C-Sharp) merkezli bir yaklaşım seçilmiştir.

Son olarak, çalışmanın genel bir değerlendirilmesi yapılarak web'in geleceğinden, uygulamanın kullanılabileceği alanlardan ve geliştirilebilirliliğinden bahsedilmektedir.

1.1. E-ticaret ve Kapsamı

E-ticaret için verilebilecek tek bir tanım olmamasına rağmen, e-ticaret Dünya Ticaret Örgütünce (WTO: World Trade Organization) mal ve hizmetlerin üretim, reklam, satış ve dağıtımlarının telekomünikasyon ağları üzerinden yapılması olarak tanımlanmaktadır [1]. Diğer bir deyişle e-ticaret her türlü malın, servis ve hizmetin bilgisayar teknolojisi, elektronik iletişim kanalları ve ilgili teknolojiler (Akıllı Kart – Smart Card, EFT: Elektronik Fon Transferi, Pos Terminalleri, Faks vb.) kullanılarak satılması ve satın alınmasını kapsamaktadır.

1.2. E-ticaretin Gelişim Süreci ve Geleceği

E-ticaret teknolojinin gelişimine paralel olarak ortaya çıkmıştır. E-ticaret sadece internet kullanılarak değil, çeşitli e-ticaret araçlarıyla da yapılabilir. Bunları telefon, faks, televizyon, bilgisayar, elektronik veri değişimi (EDI: Electronic Data Interchange), sayısal televizyon, internet, telekomünikasyon, Mobil İletişim için Küresel Sistem (GSM: Global System for Mobile Communications vb.) olarak sıralayabiliriz. Elbette e-ticaretin büyük bir kısmı internet aracılığı ile gerçekleştirilmektedir. Teknolojide meydana gelen önemli gelişmeler (Hızlı işlemciler, uydular, optik kablolar vb.) e-ticareti önemli noktalara getirmiştir.

Ekonomik İşbirliği ve Gelişim Örgütü (OECD: Organization for Economic Co-operation and Development), Avrupa Birliği, ABD gibi ekonomiler internet üzerinden yapılan

e-ticaretin dünya geneline yayılması, güvenli bir şekilde gerçekleşmesi ve gelişmesi konusunda geçmişten günümüze ortak eylem planları geliştirmektedir.

1.3. Web Ortamındaki Gelişmeler

E-ticaretin gelişmesinin ve web'in başarılı olmasının nedeni basitlik ve yaygınlığıdır. İstediğimiz zaman istediğimiz yerden e-ticareti gerçekleştirebiliriz. Internetin ilk kullanıldığı yıllarda firmalar sadece reklam amaçlı, statik sitelerle yani Hiper Metin Transfer Protokolü (HTTP: Hyper Text Transfer Protocol) ile Hiper Metin İşaretleme Dilinde (HTML: Hyper Text Markup Language) biçimlendirilmiş belgelerle müşterilere ulaşmaktaydılar. Daha sonra işletmelerin müşterilerine web üzerinden bazı iş süreçlerini yaptırma gereksinimi sonucunda uygulama web'i (application web) ortaya çıkmıştır. Bu yapıda sunucu tarafında çalışan programlar (server-side programs) vasıtasıyla hazırlanan dinamik HTML belgeleri ile kullanıcı ve iş uygulaması arasında etkileşim sağlanmıştır.

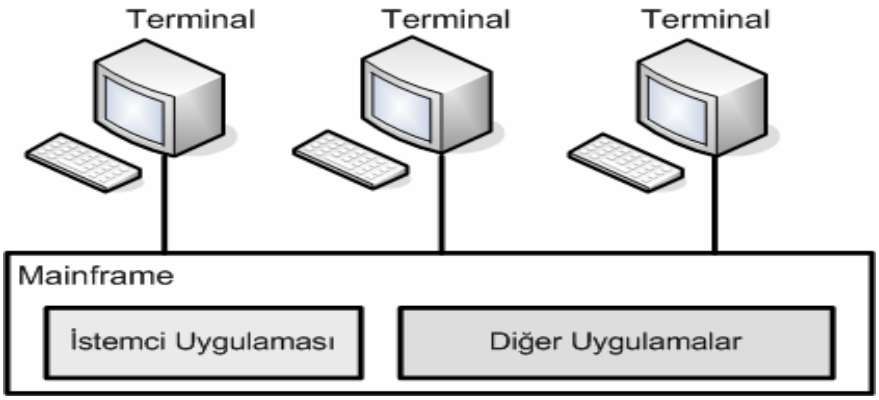
İşletmelerin diğer işletmelerle ortak işlemler yapmak amacıyla web servisleri ortaya çıkmıştır. Günümüzde halen gelişme aşamasında olan bu yapıda temel amaç, işletme bilgi sistemlerindeki program modüllerinin etkileşimini sağlamaktır. Web servislerini web üzerinden yayınlayıp istemcilerin erişmesini ve uygulama fonksiyonlarını çalıştırmalarını sağlayabiliriz. Bu fonksiyonlar değişik iş süreçlerini gerçekleştirmektedir.

2. ÜÇ-SIRALI (KATMANLI) İSTEMCİ-SUNUCU MİMARİSİ

2.1. Bir-Sıralı (One-Tiered) İstemci-Sunucu Mimarisi

Bu tip mimari mainframe mimarisidir. Mainframe'lerin geçmişinde, dağıtılmış veri işlemeden (DDP: Distributed Data Processing) daha ziyade, merkezileştirme söz konusudur. Günümüzde ise bundan uzaklaşılmış ve bir bilgisayar ağında bulunan kullanıcılara ve küçük sunuculara servis yapılmaya başlanmıştır. Bu tür bir sistem birden fazla amaca hizmet etse de yine de kullanışlı ve fiyatça uygun bir sistem değildir [3].

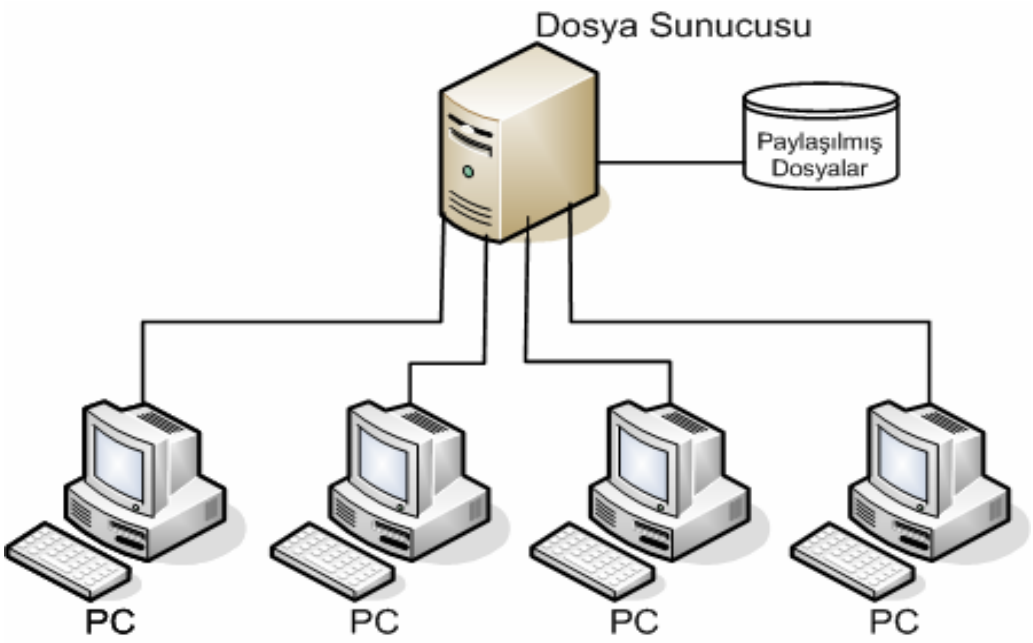
Mainframe yazılım mimarisinde uygulamayı oluşturan tüm zeki kısımlar merkezi bir ana bilgisayardadır. Mainframe mimarisi donanıma bağlı bir yazılım mimarisi değildir. Kullanıcı etkileşimi bir kişisel bilgisayar (PC: Personal Computer) ya da UNIX iş istasyonu ile yapılabilir. Şekil 2.1. bir mainframe mimarisinin yapısını göstermektedir. Tüm işlemler mainframe üzerinde yapılmaktadır. Sistem kullanıcı sayısı ve her kullanıcının sistem üzerindeki artan yükünden dolayı daha çok kaynağa (bellek, işlemci, disk vb.) ihtiyaç duyar ve yeni bir bilgisayarın sisteme dahil olmasını gerektirir. Bu da oldukça maliyetlidir ve sistemin yayılmasını limitli kılar [4]. Ayrıca, kullanıcıların sistem üzerindeki yükünün artması sisteme ulaşılabilirliliği sorunlu kılar. Terminallerin kendi üzerinde bir işlem gücü yoktur. Bu yapının en büyük eksikliklerinden birisi Grafik Ara Yüzü (GUI: Graphical User Interface) destekleyememesi ve coğrafik olarak farklı mekanlarda bulunan veritabanlarına kolay erişememesidir. Çünkü, sadece merkezi yapıyı destekler. Mimarinin güvenlik sorunlarından dolayı da çoğu firma ek güvenlik önlemleri almak zorunda kalmıştır.



Şekil 2.1. Bir-sıralı (one-tiered) istemci-sunucu mimarisi

2.2. Dosya Paylaşımı Mimarisi (File Sharing Architecture)

Bu mimaride temel olarak dosyalar sunucu üzerindeki paylaşılmış bir alanda bulunmaktadır. Masaüstü kullanıcıları ihtiyaç duydukları zaman bu dosyaları kendi makinalarına ağ üzerinden indirir. Şekil 2.2. dosya sunucusu yapısını göstermektedir.



Şekil 2.2. Dosya paylaşımı mimarisi (File sharing architecture)

Bu mimaride tüm iş kuralları ve uygulama masaüstü bilgisayarlarda çalışır. Sunucu sadece dosyaları barındırıp istenildiğinde kullanıcıya bunları iletir. Uygulama ve veri işleme tamamen kullanıcı bilgisayarında çalışır. Kullanıcı bilgisayarında işlemler tamamlandıktan sonra yapılan değişiklikler sunucuya transfer edilmektedir [5].

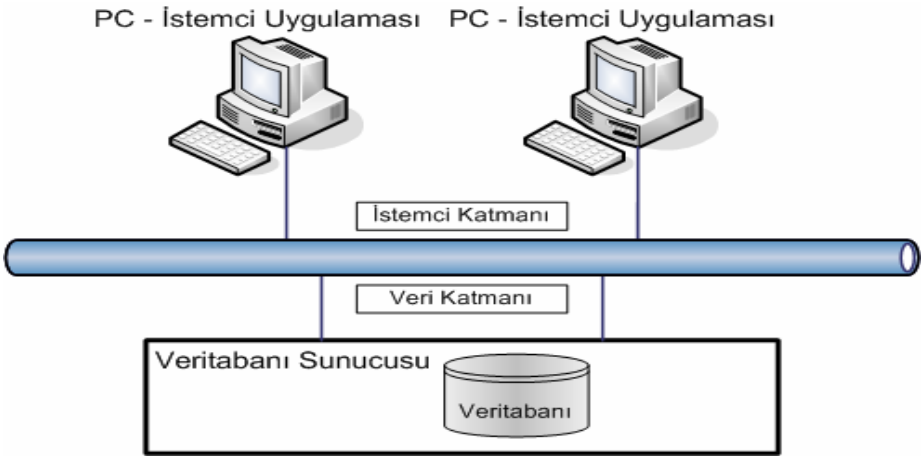
Dosya paylaşımı yönetimi verilerin veya dosyaların küçük olduğu, kaynakların ve bağlantı sayılarının az olduğu yerlerde rahat bir şekilde uygulanmaktadır. Ancak, aynı anda birden fazla kullanıcının aynı dosya üzerinde çalışması tutarsızlık oluşturur. Bunu kontrol edecek mekanizmalar bu mimaride yoktur. Ağ trafiğinin artması, verilerin büyümesi gibi nedenlerle bu mimari pek sık kullanılmamaktadır.

2.3. İki-Sıralı (Two-Tiered) İstemci-Sunucu Mimarisi

Bu mimari adından da anlaşılacağı gibi bir sunucu ve ona bağlı istemciler olmak üzere iki katmandan oluşur. Kullanıcı, istemci uygulamasıyla bağlı olduğu sunucuyla etkileşime

geçer, sunucu da gerekli verileri veri katmanından bulur. Şekil 2.3. iki-sıralı istemci-sunucu mimarisini göstermektedir. Yayılma ve yönetim bu mimaride daha karmaşıktır. Uygulama her istemciye dağıtılmış, kurulmuş ve yapılandırılmış olmalıdır [4]. Ölçeklenebilme tek-katmanlı mimariye göre önemli ölçüde geliştirilmiştir. Her kullanıcının kendine ait özel bilgisayarı vardır. Ölçeklenebilirlilikteki sınır, aynı anda kaç tane kullanıcının veritabanı sunucusunda tutulduğudur. Mainframe ile karşılaştırıldığında nispeten daha düşük maliyet sağlar. Eğer bir veritabanı sunucusuna ulaşılamazsa diğer bir veritabanı sunucusu sisteme dahil edilebilir. Her kullanıcının kendine ait işlemcisi ve kullanıcı ara yüzü bulunmaktadır. Güvenlik, bu tip mimaride mainframe'e göre daha sağlıklıdır. Kullanıcılar tipik olarak istemci uygulamasına giriş yaparlar. Ek olarak, veri katmanında gerekli kimlik denetimleri yapılır. Bu da iki farklı güvenlik sisteminden girişi gerektirir. Böylece, güvenlik arttırılmıştır. Bu tip mimaride uygulama mantığı istemci uygulamasına bağımlıdır ve istemci-sunucu etkileşiminde arabuluculuk yapmak üzere ağır bir ağ işlemi gerekir. İki-sıralı istemci-sunucu mimarisi, çalışma guruplarının birden fazla kullanıcı sayısından, belirli sayıda kullanıcıya kadar aynı anda ağ üzerinde etkileşmesi olarak tanımlandığında, dağıtık uygulamalar için iyi bir çözümdür. Ancak, bunun da ulaşılabilirlilik açısından bazı sınırlamaları vardır. Örneğin, kullanıcı sayısı arttığında sistemin performansı düşer. Bunun sonucunda da sunucu mesajları her istemciye ulaştırılamaz [5].

İki-katmanlı yazılım mimarisi genellikle yönetim ve işlemlerin çok karmaşık olmadığı durumlarda yoğun olarak kullanılmıştır. Bu mimari genellikle işlem yoğunluğunun fazla olmadığı karar destek sistemlerinde yer almıştır. Nispeten homojen ortamlarda ve iş kurallarının (business rules) çok sık değişmediği, çalışma grubu ve kullanıcı sayısının fazla olmadığı yerlerde, iki-katmanlı mimari başarıyla uygulanabilir.



Şekil 2.3. İki-sıralı istemci-sunucu mimarisi

2.3.1. İki-Sıralı (Two-Tiered) İstemci-Sunucu Mimarisinin Avantajları

İki-sıralı mimarinin avantajları şöyle sıralanabilir:

- İki-sıralı uygulamaların en belirgin avantajı uygulamaların çok hızlı geliştirilmesidir. Pek çok durumda iki-sıralı bir sistemi çok kısa bir zaman diliminde kodlamak mümkündür.
- İki-sıralı yapılar için pek çok uygulama geliştirme aracı ve diğer araçlar ve Hızlı Uygulama Geliştirme (RAD: Rapid Application Development) imkanları sunmaktadırlar. Günümüzde bu araçların yeterli ve güvenli olduğu kullanım alanlarının yaygınlaşmasından anlaşılmaktadır.
- İki-sıralı yazılım mimarisi iş kurallarının çok sık değişmediği homojen ortamlarda düzenli çalışır. Fakat iş kurallarının çok sık değiştiği heterojen ortamlarda çok fazla kullanışlı değildir [6].

2.3.2. İki-Sıralı (Two-Tiered) İstemci-Sunucu Mimarisinin Dezavantajları

İki-sıralı mimarinin dezavantajları şöyle sıralanabilir:

- İki-sıralı yapılarda iş kurallarının büyük bir kısmı istemciler üzerindedir. Bu da bir problem olarak iyi bir versiyon kontrolü gerektirir. En küçük değişiklikte dahi tüm istemci makinalara uygulamanın yeniden dağıtılması gerekir.
- İki-sıralı yapılarda sistem güvenliğini sağlamak oldukça karmaşık hale gelmektedir. Bunun sebebi, erişilecek her Yapısal Sorgulama Dili (SQL: Structured Query Language) sunucusuna erişmek için ayrı bir şifre kullanma gereğidir.
- İki-sıralı sistemlerde kullanılan istemci araçları (uygulama geliştirme araçları, sorgulama araçları vb.) ve SQL katmanı araçları (Veritabanı Yönetim Sistemleri) hala üreticilere özgün ürünlerdir. Uzun vadede bu ürünlerin kullanılabilir olup olmadığı üreticilerin ürünlerine olan desteğinin devam edip etmeyeceği günümüzde kolaylıkla tespit edilememektedir [6].

2.4. Üç-Sıralı (Three-Tiered) İstemci-Sunucu Mimarisi

İki-katmanlı istemci-sunucu sistemlerinin ilk örneklerinin ortaya çıkışından bu yana yaklaşık 10 yıl geçti. Bu geçen süre içinde, ülkemizde bir çok sektör hızla gelişmiş ve büyümüştür. Bu büyümeyle birlikte artan kullanıcı sayısı yazılım ve teknik destek hizmetlerinin yetersiz kalmasına ve sistemlerin performansının düşmesine neden oldu. Yazılım geliştiriciler ve

teknik destek hizmeti verenler, hem sunucu hem de her bir istemciyle tek tek uğraşmak zorundadır. Örneğin, bir ilde şube sayısı on olan bir banka sisteminde binlerce istemci vardır. Her bir istemcinin ek maliyetinin yanı sıra, hizmet vermek ve iş üretmek amacında olan kuruluşlar, sistem problemleri nedeniyle iş yapamayacak hale geleceklerdir. Bu yetersizliklerin ortadan kaldırılmasına yönelik olarak multi-katman, n-katman ya da üç-katmanlı yazılım modeli olarak adlandırılan sistemler geliştirilmiştir. Bu mimari istemci ile sunucu arasına üçüncü bir sunucu eklenmesiyle oluşturulur. Şekil 2.4. üç-sıralı istemci-sunucu mimarisini göstermektedir.

Uygulama kısımları aşağıdaki gibi sıralanabilir:

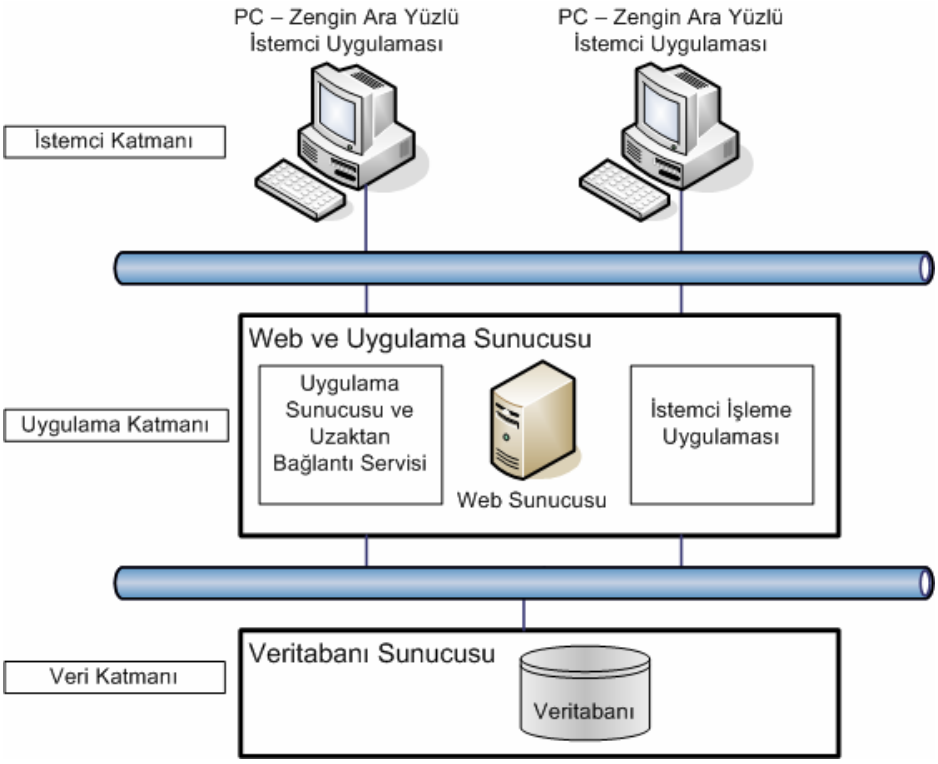
- ***İş istasyonu veya sunum mantığı:*** Kullanıcının uygulama ile nasıl etkileştiğini ele alır ve kullanımı kolay olan bir grafik ara yüzü ile sağlanır. Kısaca kullanıcının veri ile etkileşimini sağlar.

- ***İş mantığı:*** Uygulamanın mekaniklerini veya iş kurallarını ele alır. Bu kurallar, bir kuruluştaki günlük işlerin sistemli yürütülmesi gereksiniminden dolayı meydana gelir. Kurallar gelen bilginin geçerli veri tipi ve formatta olduğundan emin olmak için kullanılır. İş mantığı bir yerel ağ sunucusu veya başka bir paylaşımlı bilgisayarda bulundurulur. İş mantığı iş istasyonlarından gelen istemci istemlerine sunucu olarak karşılık verir. Hangi verilerin gerekli olduğuna karar verir ve üçüncü katman programı ile bu kez istemci olarak çalışır.

- ***Veri erişim mantığı:*** Verinin depolanmasını ve bulunup getirilmesini ele alır çünkü verinin bütünlüğünün korunması önemlidir. Veritabanına okuma ve yazma işlemlerini yöneten programdır. Veri, Veritabanı Yönetim Sistemi (DBMS: Database Management System) tarafından yönetilir.

Bu mimaride amaç, bilgisayar kaynaklarını dilimlere ayırıp, bilgisayar ağına yayarak daha verimli ve performanslı bir sistem sağlamaktır. Üç-katmanlı mimaride orta-katman (middle-tier) sunucularına aynı anda pek çok kullanıcıdan ve farklı uygulamalardan erişilebilir. Bunun için istemciden sunucuya çağrı mekanizması olarak genellikle Uzak İşlev Çağırma (RPC: Remote Procedure Call) tekniği kullanılır. Orta katman, işlem veya süreç yönetimi servislerini sağlamaktadır. Birçok uygulama tarafından paylaşılan uygulama/süreç üretim, gözleme, kaynak izleme vb. bileşenlerden oluşur. Aynı zamanda uygulama sunucusu (application server) olarak da adlandırılan bu orta katman, uygulama mantığını merkezileştirerek performans, esneklik, tekrar kullanılabilirlilik ve ölçeklenebilme gibi özellikleri güçlendirmektedir. Yayılma ve yönetim bu mimaride, iki-katmanlı mimariye göre daha kolaydır. Çünkü, uygulama mantığı merkezi bir

sunucuda bulunmaktadır [4]. Uygulamada bir değişiklik yapılacağı zaman, değişiklik yapılacak fonksiyonun yenilenmesinin ardından ilgili ara katman sunucusuna konması yeterli olur. Ayrıca, ara katman ile işlem (transaction) ve asenkron kuyruk işlemlerinin de güvenilir şekilde sonuçlanması kontrol edilebilir. Dağıtık veritabanı yapısındaki tutarlılık, çift taraflı işlem tamamlanmasıyla (two-phase commit) sağlanır. Mimari, kaynaklara erişimde yer veya adres yerine isim ile erişilmesine olanak sağlayarak ölçeklenebilmeye destek verir [5].



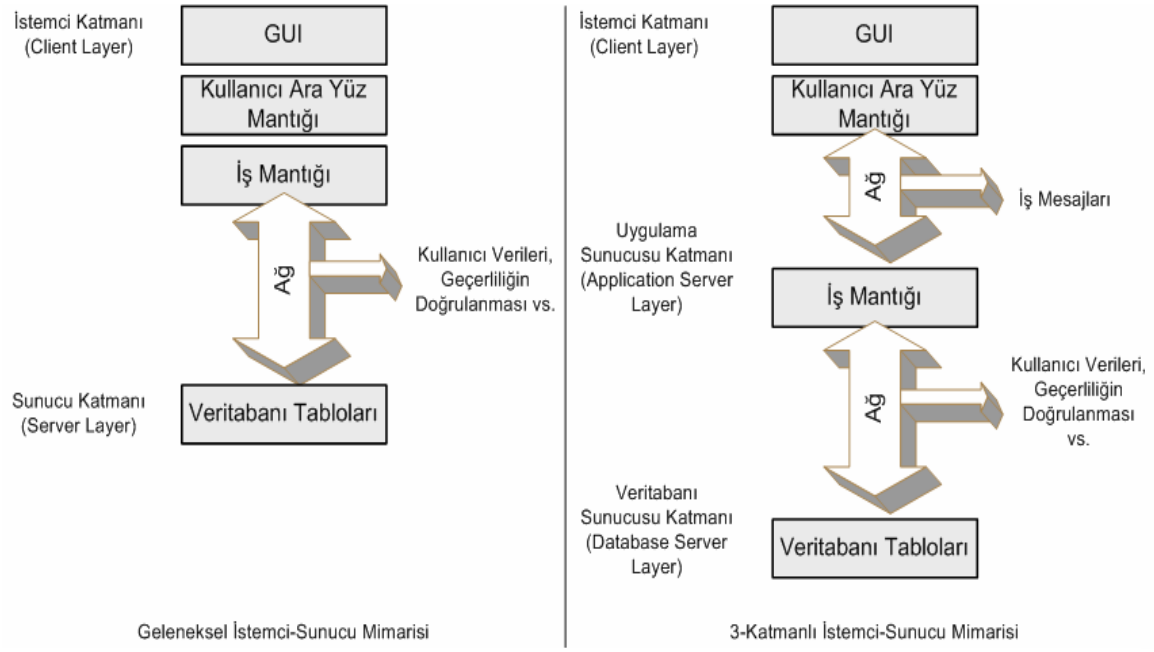
Şekil 2.4. Üç-sıralı istemci-sunucu mimarisi

Üç-sıralı mimaride özellikle en üst sıra en güçlü sistem olan ve birlikte veriyi bulunduran ana bilgisayar tarafından tutulmalıdır. İkinci sıra, çifte görev yapan güçlü bir yerel alan ağı (LAN: Local Area Network) sunucusunda bulunur. Burada yer alan sistemler üst sıradaki ana bilgisayara istekleri gönderirken istemci gibi davranır, alt sıradaki iş istasyonları ve PC'lere ise sunuculuk yapar. Böylesine bir mimari üst sıraya ana bilgisayarlar, ikinci ve üçüncü sıraya LAN'lar ve sunucular eklenerek yatay olarak yayılabilir. Bu bizi daha hızlı ağ iletişimine, daha yüksek güvenilirliğe ve daha yüksek sistem performansına ulaştırır. Uygulama katmanı bir başka

uygulama katmanına istemci olduğunda mimari dikey olarak yayılır, bu şekilde yapı N katmanlı mimari şekline dönüştürülür.

Şekil 2.5. geleneksel ve üç-sıralı istemci-sunucu mimarisinde yapının nasıl değiştiğini göstermektedir. Geleneksel istemci-sunucu uygulamalarındaki temel bazı sorunlar ise şu şekilde sıralanabilir:

a. ***Sunucu tıkanıklığı:*** Bir sunucu devamlılığı sağlamalı ve istemci uygulamaları tarafından yayınlanan geçici SQL komutlarını derleyebilmelidir. Doyum noktasına ulaşan bir makinada bu durum elbette komutların icra süresini arttıracak ve genel anlamda bir performans kaybına neden olacaktır.



Şekil 2.5. Geleneksel ve üç-sıralı istemci-sunucu mimarisinin karşılaştırılması

b. ***Ağ tıkanıklığı:*** İstemci-sunucu arasında gerçekleşen iletişimde istemci uygulamalarından kaynaklanan aşırı isteklerde ağda bir yavaşlama olur .

c. ***Yazılım dağılımı problemleri:*** İstemci yazılımındaki bir hata veri bütünlüğünü tehlikeye atar. Bu hata bulunduğunda yeni istemci yazılımı en kısa sürede devreye sokulmalı ve tamamen tüm kullanıcılara dağıtılmalıdır.

d. ***Çoklu-platform destekleme problemleri:*** Grafiksel istemciyi kullanmayan kullanıcılar için aynı düzeyde karakter modunda uygulamalar desteklenmelidir.

e. ***Güvenlik açıkları:*** Eğer kötü niyetli bir kişi veritabanına ulaşmayı başarırsa bilgilerin bulunduğu tablolara zarar verebilir [7].

Üç-sıralı model, iyi bir bilgisayar ağı yönetimi ve veri bütünlüğü ister. Mimarinin kurulması ve yönetimi karmaşıktır. Buna karşın bilgisayar kapasitesi arttırılmıştır. Bu sayede daha çok kullanıcıya hizmet verilir. Ayrıca, son-kullanıcılar (front-end-users) ve uygulamalar için önemli olan iş istasyonlarının sayısını arttırmak, ek LAN'lar, daha güçlü sunucular hatta ek sıralar eklemeden gerçekleştirilebilir. Üç-sıralı istemci-sunucu mimarisi istemcileri tamamen iş kurallarından soyutlar. Veri depolama ve aynı andaki olaylar kapsüllemenin sonucunda meydana gelir.

2.4.1. Üç-Sıralı (Three-Tier) İstemci-Sunucu Mimarisinin Avantajları

Üç-katmanlı mimari aşağıdaki kritik alanlarda önemli avantajlar sunar. Bunlar:

- Kontrol
- Güvenilirlik
- Ölçülebilirlilik ve performans
- Ölçeklenebilirlilik (Esneklik, büyüme, değişim)
- Standartlar

a. ***Kontrol:*** Orta katmandaki sunucuda çalışan programlar, bilgi sistemlerinin kontrolü altında ve güvenlidir. Bunlar test edilebilir ve denetlenebilir. İş mantığı, sunucu üzerinde olduğundan, yazılım dağıtımı problemleri en aza indirilmiştir. Uygulama sunucusundaki programları güncellemek, binlerce kişisel bilgisayardakini güncellemekten daha kolaydır. Bir hata olması durumunda sunucu ve istemci kısmındaki programların ayrı ayrı hata ayıklaması yapılabilir. Kullanıcı ara yüz kodu değiştiğinde, dağıtılması gerekmez. Uygulama sunucusu dağıtım noktası olarak kullanılabilir. Güvenlik Duvarı (Firewall) etkisi kişisel bilgisayarların dağıtım sorununu azaltır.

b. ***Güvenilirlik:*** Bir uygulama sunucusu donanım olarak, bir masaüstü makinadan daha güvenlidir. Bu sadece makinalardaki işletim sistemlerinin zayıf ve güçlü yanlarından

kaynaklanmaz. Aynı zamanda bir masaüstü makinasının, bir firma yerine, tek bir kişi tarafından kullanılmasından kaynaklanır.

c. ***Ölçülebilirlilik ve performans:*** Sunucu sayısı kontrol edilebildiğinden, aşırı yüklenmelerde sunucu sayısı arttırılarak sunucunun aşırı yüklenmesi engellenebilir. Sunucu üzerindeki yük bilinir ve kullanıcılar kontrolsüz olarak sunucuya bağlandıkça artmaz. Bu sayede sistemin performansında diğer mimarilerde olduğu kadar bir düşüş olmamaktadır.

d. ***Ölçeklenebilirlik (Esneklik, büyüme, değişim):*** Üç-katmanlı yaklaşım modülerliliği zorunlu olarak sağlar. Uygulama bölümlenince, spagetti kod yazmak zorlaşır. Üç-katmanlı mimari esnek bir kaynak tahsisi sağlar. Orta-katmandaki fonksiyonaliteyi gerçekleştiren sunucular organizasyonların ihtiyaçlarına göre kolay bir şekilde değiştirilebilir ve geliştirilebilir.

Bir üç-katmanlı uygulamaya büyük bir alt sistem eklenmesi gerektiğinde, alt sistemi geliştirenler şu adımları izleyerek hareket etmelidirler:

- Yeni uygulama için yeni bir sunucu (donanım) eklenir.
- Tasarım yapılır.
- Yeni iş mantığı (2. katman) hizmetleri (metotlar) geliştirilir.
- Kullanıcı ara yüzleri, müşterinin isteği doğrultusunda değiştirilir.
- Veritabanı genişletilir ve yeni veri hizmetleri (3. katman) oluşturulur.

e. ***Standartlar:*** Üç-katmanlı yaklaşım, eski ve yeni kullanıcı ara yüzlerinin aynı iş mantığını kullanmalarına izin verir. Standartlaşma amacıyla istenen özellikler; özel araçları kullanabilme, yeni yazılım araçları ve teknolojiyle kolayca çalışma ve DBMS'den bağımsızlıktır [6].

2.4.2. Üç-Sıralı (Three-Tier) İstemci-Sunucu Mimarisinin Dezavantajları

Üç-sıralı istemci-sunucu mimarisinin dezavantajları şunlardır:

- Bu mimaride iyi bir ağ trafiği yönetimi, sunucular arası yük dengelemesi yapılması ve sunucular üzerinde meydana gelebilecek problemleri tölore edebilecek yöntemler geliştirilmelidir.
- Mimarinin kurulması oldukça karmaşık bir işlemdir [6].

3. WEB SERVİSLERİ MODELİ

Önümüzdeki yıllarda yazılım uygulamalarının büyük bir kısmının veritabanı paylaşımına dayanacağı Sun, Microsoft, IBM, HP, Oracle gibi sektöre yön veren firmalar tarafından öngörülmektedir. Bu firmalar web servisleri yazılım ve uygulama geliştirme araçlarını geliştiricilere sunmaktadır. Bilgi paylaşımını olanaklı kılan ise hala gelişme aşamasındaki web servisleridir. Son altı yıldır internet ve web teknolojileri sayesinde yeni bir devrim başlamıştır. Genişletilebilir Etiketleme Dili (XML: eXtensible Markup Language) tabanlı web servisleri içerisinde tanımlanan metotlar, istemci uygulamaları tarafından çağrılıp kullanılabilir.

Web servisleri açık internet standartlarına dayanır. Bu standartlar, Nesne Erişim Protokolü (SOAP: Simple Object Access Protocol), Web Servisi Tanımlama Dili (WSDL: Web Services Description Language) ve Evrensel Açıklama, Keşif ve Entegrasyon (UDDI: Universal Description, Discovery and Integration)'dur.

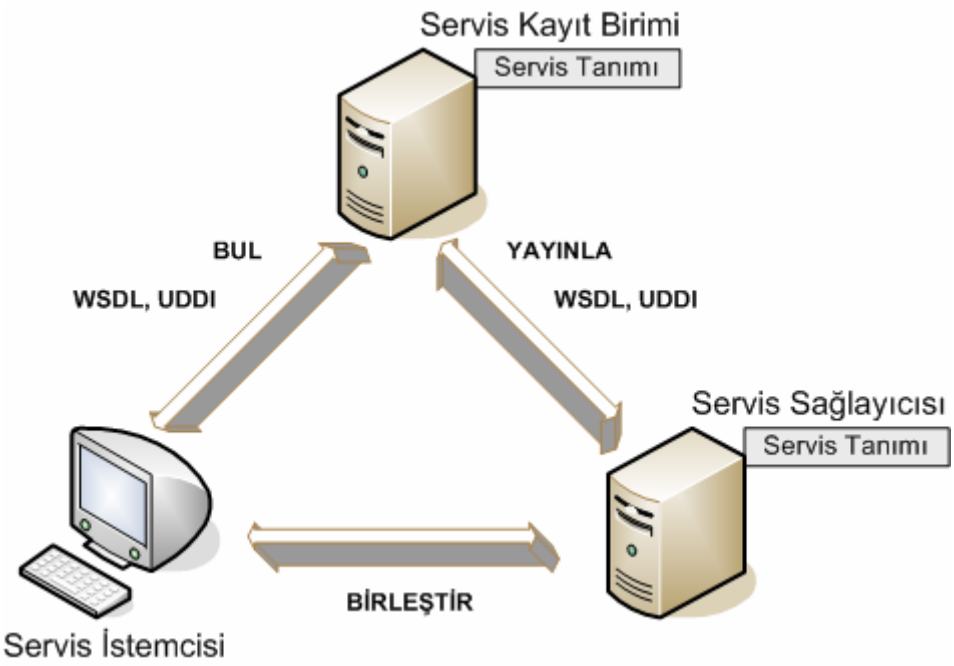
Bir örnek verecek olursak, aynı ya da farklı amaca hizmet eden birden fazla firmanın ortak bir çatı altında toplanarak, müşterilerin firmalara daha kolay ulaşabilmesi, firmaların da müşteri portföyünü genişletmesi amacıyla web servisleri kullanılabilir.

Web servisleri modeli üç temel birimin etkileşimine dayanır. Şekil 3.1.'de gösterilen bu birimler şunlardır:

- ***Servis Sağlayıcısı (Service Provider):*** Servis sağlayıcı istemcilerin sağlayıcıda bulunan servislere erişimini sağlar. Servis sağlayıcı kendi sitesinde bulunan web servisleri tanımını servis kayıt birimine (service registry) kaydederek bu servisinin nasıl çağrılacağını belirtir.

- ***Servis İstemcisi (Service Requester):*** Servis sağlayıcısında bulunan web servislerini çağırarak kullanan istemci uygulamalarıdır. Web servisinin nasıl çağrılacağını belirler ve ilgili parametreleri servis kayıt biriminden arayarak bulunup çağrılmasını sağlar.

- ***Servis Kayıt Birimi (Service Registry):*** Servis sağlayıcılarının yayınladıkları web servisi tanımlarını saklar ve aranıp bulunmasını sağlar. Servis sağlayıcıları servis kayıt birimini tarayarak istediği servisler hakkında bilgi alabilir. Servis kayıt birimi her servisin nasıl çağırılacağı konusunda tanım bilgileri içerir [8].

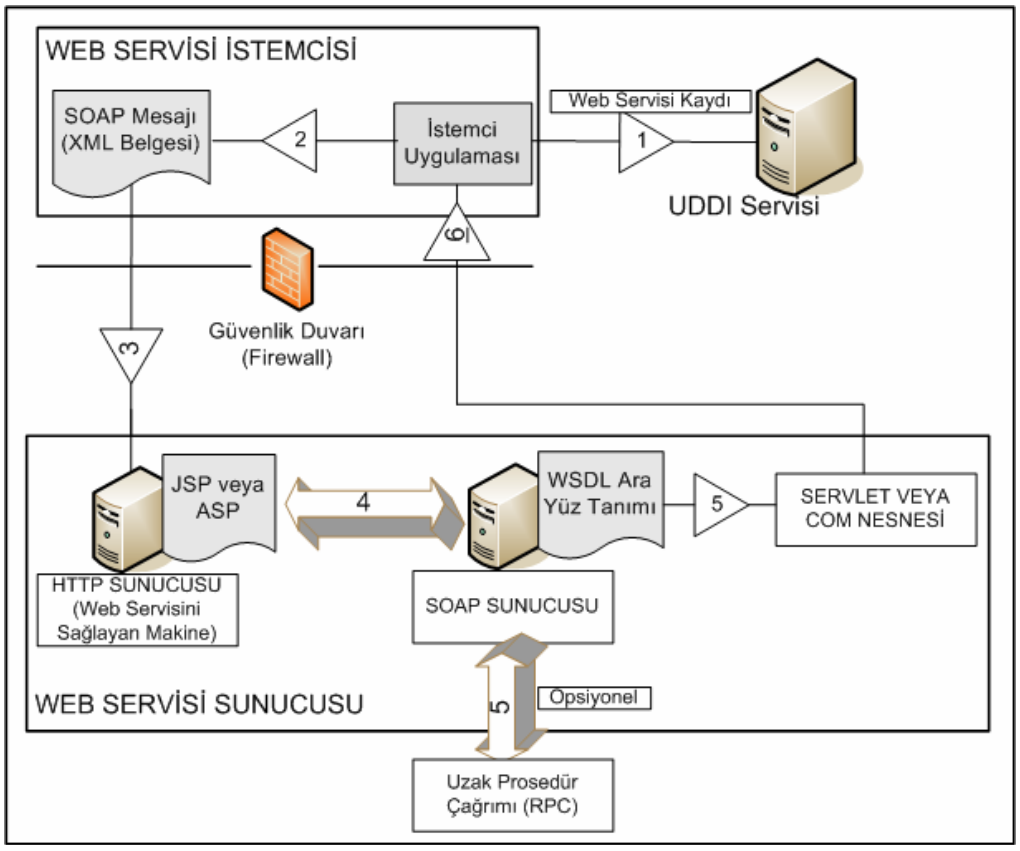


Şekil 3.1. Web servisi modeli

Şekil 3.2.'deki şema bir web servisi istemcisinin bir servis sağlayıcıdan bir servisi çağırma aşamasındaki temel adımları göstermektedir.Bunlar:

1. Web servisi istemcisi (SOAP Client) servis kayıt biriminden (UDDI) web servisini bulur.
2. İstemci bir SOAP mesajı hazırlar. SOAP mesajı bir XML belgesidir.
3. İstemci SOAP mesajını web sunucusu veya uygulama sunucusunda çalışan SOAP istek dinleyicisine gönderir. İstek dinleyici gelen isteklere cevap veren sunucu programlarıdır. Bu programlar bir Java Server Sayfaları (JSP: Java Server Pages), Aktif Server Sayfaları (ASP: Active Server Pages), Ortak Geçit Ara yüzü (CGI: Common Gateway Interface) veya Internet Sunucusu Uygulama Programı Ara yüzü (ISAPI: Internet Server Application Program Interface) programıdır.
4. SOAP sunucusu gelen SOAP mesajını çözümler (parse eder) ve gerekli parametreleri göndererek istenen nesnenin istenen yöntemini çağırır.
5. Çağrılan nesnedeki yöntem çalışır ve sonuçları SOAP sunucusuna gönderir. SOAP sunucusu gelen sonucu SOAP mesajı formatında biçimlendirerek istemciye gönderir.

6. İstemci gelen SOAP mesajının içindeki bilgileri alarak istekte bulunan programa gönderir.



Şekil 3.2. Web servisi istemci ve sağlayıcısı arasındaki temel işlemler

4. WEB SERVİSLERİNİN ÖZELLİKLERİ

Bir web servisi ortak işlem yaparak bilgi paylaşımını sağlar. Amaç, farklı obje modelleri kullanan ve hatta farklı işletim sistemlerinde çalışan bir grup uygulamayı bir araya getirerek kullanımı kolay web uygulamaları oluşturmaktır.

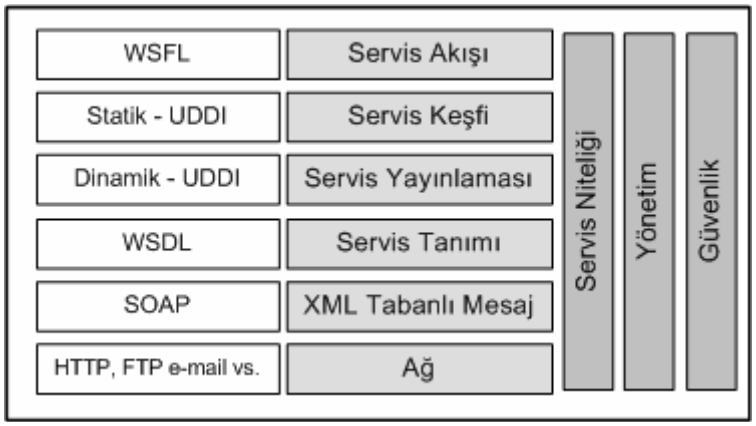
COM objelerinde olduğu gibi web servislerinin de içerisindeki kodun ne olduğu görülmez fakat sağladığı imkanlardan kolayca yararlanılır. Web servisleri, kullanıcılara servisi nasıl kullanacaklarını ve sağladığı servisleri tanımlayan çok iyi bir ara yüz sağlar.

Microsoft firmasının Distrubuted Component Object Model (DCOM), Sun'ın RMI, Object Managament Group'ın, Common Object Request Broker Architecture (CORBA) ve benzeri teknolojilerden farklı olarak web servisleri belirli obje modellerine özgü protokolleri kullanmaz. Web servisleri iletişimde daha önce belirttiğimiz SOAP, HTTP, XML gibi veri formatlarını kullanır. Yani bu standartları destekleyen bir sistem web servislerini rahatlıkla kullanabilir. Bu da XML Web Servislerine farklı bir bilgisayar ve farklı bir platformdan istemci olunabileceğini gösterir. Web servisleri platformdan, dilden ve obje modelinden bağımsızdır. Web servislerinin en büyük avantajlarından biri de budur.

Web Servisi modeli ASP.Net tarafından desteklenir. ASP.Net bütünleşik bir web geliştirme platformudur ve ASP'den geliştirilmiştir. ASP.Net web servisleri modeli durumsuz (stateless) bir mimari sunar. Durumsuz mimariler genelde daha ölçeklenebilirdir. Gelen her servis isteğine karşılık yeni bir nesne yaratılır [9].

5. WEB SERVİSLERİ STANDARTLARI

XML Web Servisleri, adından da anlaşıldığı gibi mesaj alış verişi için XML standardını kullanır. Şekil 5.1. web servisi mimarisindeki temel katmanları göstermektedir. Bu katmanlarda belirtilen güvenlik, iş akışı, servis kalitesi ve yönetim gibi konulardaki web servisi standartları henüz araştırma ve geliştirme aşamasındadır.



Şekil 5.1. Web servisi mimarisi katmanları

5.1. XML Nedir?

XML web üzerinde veri değişimi için hızla standart haline gelen metin temelli bir işaretleme dilidir. HTML'den farklı olarak XML etiketleri verinin nasıl gösterileceğini değil, verinin nasıl tanımlanacağını gösterir.

Internet üzerinde platformdan bağımsız veri alış-verişinde XML kullanılmaktadır. XML'de veriler etiket ve sonlandırcı etiket arasında paketlenerek gönderilir. XML'de veriler iç içe etiketlerden oluştuğu için XML'in hiyerarşik bir yapısı vardır. XML, Internet Komitesi (W3C) tarafından ortaya çıkarılmıştır. XML farklı uygulamalarda, verilerin bütünleştirilmesinde araç olarak kullanılır. Yazılım firmaları XML'e büyük destek vermektedir. XML, Firmadan Firmaya (B2B: Business-to-Business) olan uygulamalarda standart bir dil ve belge türüdür.

5.2. XML Uygulama İçin Neden Önemli?

XML metin halinde olup ikili (binary) formatında olmadığı için standart metin editörlerinden görsel (visual) geliştirme ortamına kadar herhangi bir araçla oluşturulup, geliştirilebilir. Bu da programın hata ayıklaması (debug)'nı ve küçük miktarda verinin

depolanmasını sağlar. Diğer taraftan bir veritabanı büyük miktarda XML verisi depolayabilir. XML, veri tanımlaması sayesinde, ne çeşit verinin (metin, sayı, vb.) olduğunu gösterir. İşaretleme etiketleri bilgiyi belirttiği ve veriyi kısımlara ayırdığı için bir e-posta programı onu işleyebilir, bir arama programı belirli kişilere yollanacak mesajları arayabilir, bir browser ile de XML çözümlenebilir. Kısaca, bilginin farklı bölümleri tanımlandığından bunlar farklı uygulamalarda farklı şekillerde kullanılabilir.

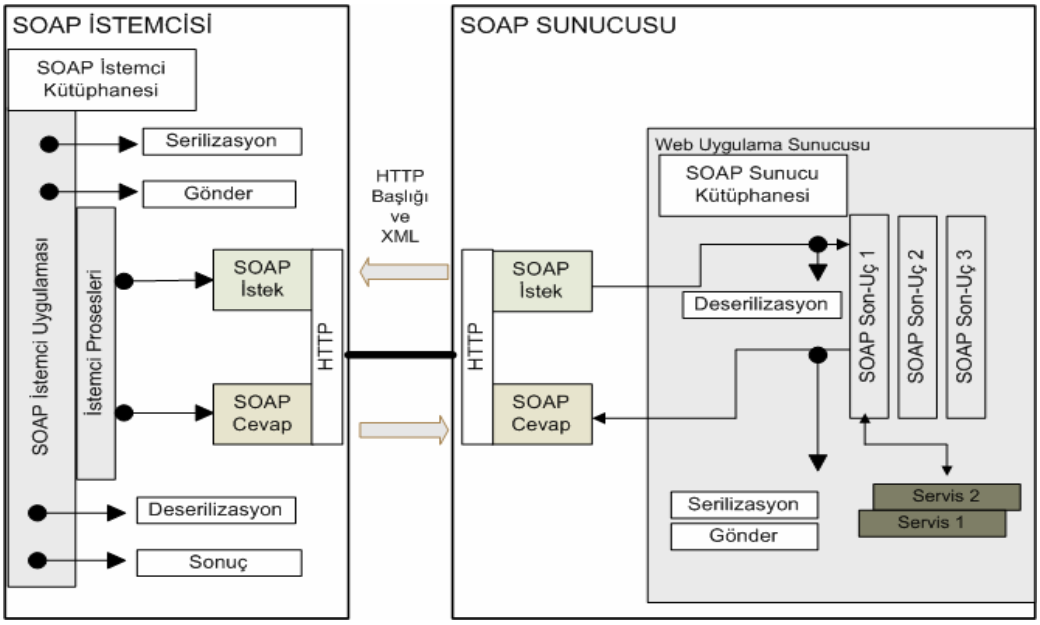
XML uygulama, dil, işletim sistemi gibi kısıtlamalara bağlı değildir. Şu anda hemen her işletim sistemi ve uygulama XML belgelerini okumak ve yazmak için yerleşik özelliklere sahiptir. Bütün XML belgeleri unicode tabanlıdır. İşletim sistemleri farklı kod sayfalarını (code page) kullandıkları için farklı dillerdeki belgeleri görüntüleyemezler. Ancak, unicode kullanımı farklı dillerdeki belgelerin görüntülenmesine olanak tanır [10].

5.3. SOAP

SOAP uygulamaları birleştirmek için geliştirilmiştir. XML ile formatlanmış bilgilerin iletişimini sağlayan bir protokoldür. Uygulama bütünleştirme için ara yazılım (middleware) olarak adlandırılan çözümler sunmuştur. RPC, IIOP (Internet Inter - ORB Protocol) ve Java Uzak Metot Çalıştırma (RMI: Remote Method Invocation) bu çözümlerden bazılarıdır. Bu ara yazılım çözümleri internet ortamında iletişim sağlama konusunda yetersizdir. Bu nedenle XML tabanlı bir protokol olan SOAP giderek yaygınlaşmaktadır.

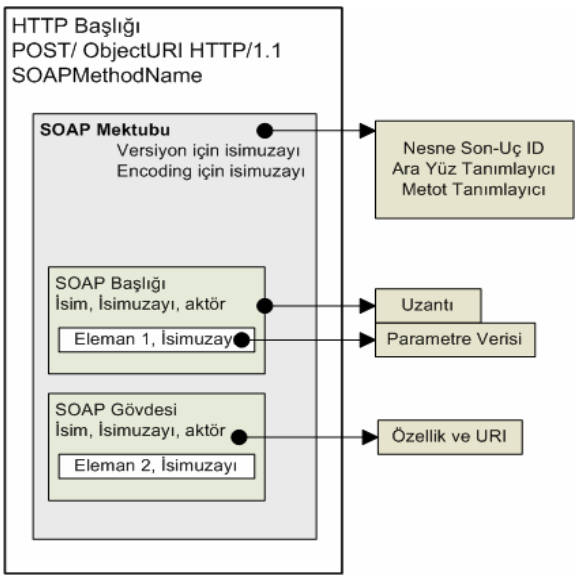
SOAP istemcilerin sunucularda olan nesne yöntemlerini çağırmasını ve sonuçların alınmasını sağlayan basit istek/yanıt (request/response) protokolüdür. SOAP mevcut internet altyapısında olan router, güvenlik duvarı ve proxy sunucularda bir değişiklik yapmadan kolayca çalışmaktadır [11].

Şekil 5.2. bir SOAP istemci ve sunucusu arasındaki iletişimi göstermektedir. Bir SOAP uygulaması geliştirmek için istemci ve sunucuya SOAP kütüphanelerinin yüklenmesi gerekir. Bu kütüphaneler bir XML çözümleyicisi ve SOAP işlemcisi içerir. İstemci SOAP uygulaması bir istek mesajı oluşturarak, bu isteği SOAP sunucusunda tanımlanmış servis uç noktalarına (end-point) gönderir. SOAP sunucusu ilgili servisi çalıştırdıktan sonra SOAP yanıt mesajı hazırlar. Hazırlanan SOAP yanıt mesajı istemciye iletilir.



Şekil 5.2. SOAP istemci ve SOAP sunucusu arasındaki iletişim

Şekil 5.3. HTTP protokolü ile gönderilen bir SOAP mesajını göstermektedir. SOAP mesajı HTTP POST metodu veri paketinin içinde gönderilir. Bir SOAP mesajı bir SOAP zarfından (SOAP envelope) oluşur. SOAP zarfı seçimlik bir SOAP başlığı (SOAP header) ve SOAP gövdesinden (SOAP body) oluşur. SOAP gövdesi çağırılacak metot ve metodun içerdiği parametreleri içerir.



Şekil 5.3. Bir SOAP mesajının yapısı

Şekil 5.4. ve 5.5. gerçekleştiğimiz uygulamadaki bir SOAP istek ve yanıt mesajının içeriğini göstermektedir.

```
POST /ankara_web/CompanyWebService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/GetCustomerReserves"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCustomerReserves xmlns="http://tempuri.org/">
      <customer_id>int</customer_id>
    </GetCustomerReserves>
  </soap:Body>
</soap:Envelope>
```

Şekil 5.4. SOAP istemci istek (request) mesajı

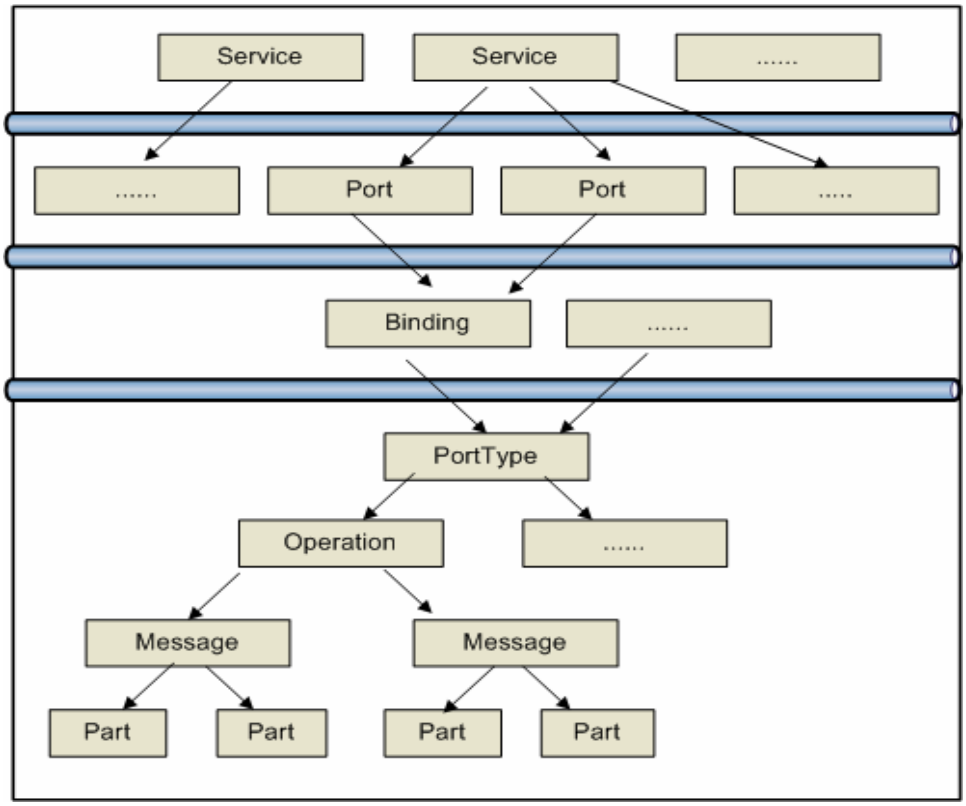
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetCustomerReservesResponse xmlns="http://tempuri.org/">
      <GetCustomerReservesResult>
        <ReserveDetail>
          <travel_id>int</travel_id>
          <reserved_place>int</reserved_place>
        </ReserveDetail>
        <ReserveDetail>
          <travel_id>int</travel_id>
          <reserved_place>int</reserved_place>
        </ReserveDetail>
      </GetCustomerReservesResult>
    </GetCustomerReservesResponse>
  </soap:Body>
</soap:Envelope>
```

Şekil 5.5. SOAP yanıt (response) mesajı

5.4. WSDL

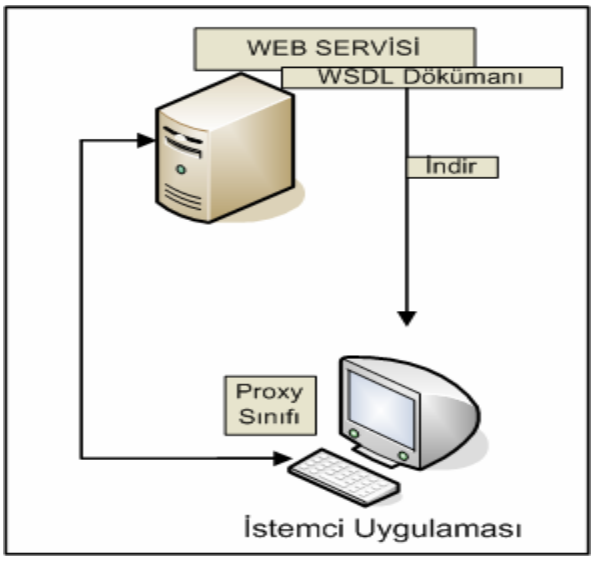
Bir uygulamanın bir web servisini kullanması için web servisinin nasıl çağırılacağı, ara yüzü, protokolleri ve kodlama standartları belirtilmelidir. WSDL web servisini tanımlayan bir XML belgesidir. Bir anlamda dağıtık programlamada kullanılan Ara yüz Tanımlama Dili (IDL: Interface Definition Language)'ne benzer. Web servisi tanımı, işlemler, giren ve çıkan mesaj formatları, ağ ve port adresleri gibi bilgileri tanımlar [12, 13, 14, 15]. Şekil 5.6. bir web servisi wsdl dökümanının yapısını göstermektedir. Bir WSDL belgesi aşağıdaki temel elemanları içerir:

- *Types:* Mesajlarda kullanılacak veri tiplerini belirtir.
- *Message:* İletişimde kullanılacak mesajları tanımlar.
- *PortType:* Web servisinin içerdiği işlemleri (metotları) ve ilgili mesajları tanımlar.
- *Binding:* İşlem ve mesajlarda kullanılacak veri formatlarını tanımlar.
- *Port:* Binding ve web adresinden oluşan servis noktasını tanımlar. Web adresi servisin çalıştırılacağı Tekdüzen Kaynak Bulucu (URL: Uniform Resource Locators)'dur.
- *Service:* Kullanılan portlar kümesidir.



Şekil 5.6. WSDL enformasyon modeli

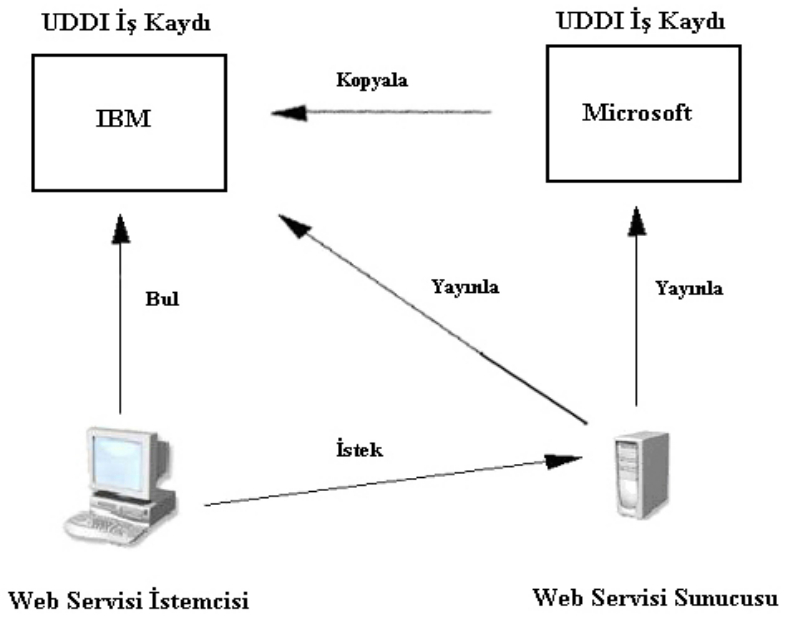
Şekil 5.7. WSDL dökümanının web servislerindeki kullanımını göstermektedir. Burada istemci bu dosyayı kendi bilgisayarına indirdikten sonra, o web servisinin metotlarını kullanabilir.



Şekil 5.7. WSDL dökümanının web servislerindeki kullanımı

5.5. UDDI

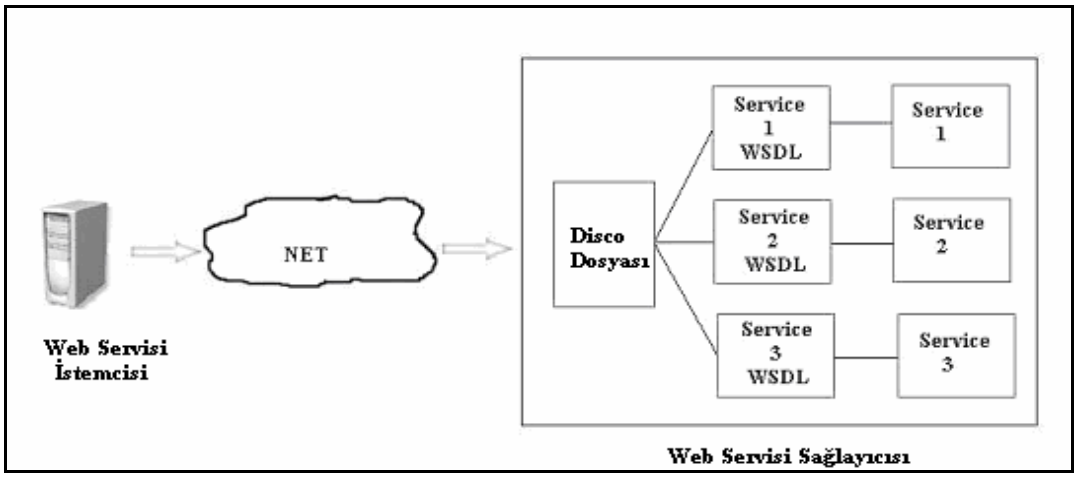
Bir web servisini kullanmak için kullanıcının web servisi sağlayan kurumları ve bu kurumların verdikleri web servislerinin neler olduğunu bilmesi gerekir. UDDI kısaltmasında geçen *evrensel*, *tanım*, *buluş* ve *bütünleştirme* kelimelerinin ifade ettiği gibi UDDI kurumların kendilerini, sağladıkları servisleri yayınlayarak tanımlamalarını, ve bu bilgilerin daha sonra diğer kurumlarca taranıp bulunmasını sağlayan bir standarttır [16, 17, 18]. UDDI Kurum Kayıt Servisi (UDDI Business Registry) kurum ve web servisleri bilgilerini saklayan sunuculardır. Bu sunucular servis sağlayıcılarından gelen bilgileri kendi veritabanlarına kayıt ederek diğer kurumların erişimine açar. Şu anda aktif olarak çalışan kurum kayıt sunucuları *uddi.microsoft.com* ve *uddi.ibm.com*'dur. Şekil 5.8.'de görüldüğü gibi bu sunucular kendilerine kayıt edilen bilgileri diğer sunuculara da kopyalayarak kolay ve hızlı erişilmesini sağlar [2]. UDDI sunucuları kurum ve servis kayıt, güncelleme ve tarama işlemlerini web servisleri (SOAP mesajları) ile gerçekleştirir.



Şekil 5.8. UDDI kurum kayıt sunucuları

6. WEB SERVİSLERİNDE KEŞİF (DISCOVERY)

Web servislerinin sayısı arttığında ve web servisleri web üzerinde iş takibi açısından genel bir dil haline geldiğinde, kullanıcılara web servisinin adresini gönderebilmek sorun olabilmektedir. Kullanıcılara bunu e-posta, telefon vb. ile bildirmek elbette verim ve pratiklik sağlamayacaktır. Ayrıca tüm web servis adreslerini içeren bir HTML dökümanı hazırlamak mantıklı gibi görünse de güncellenmesi gerekecek ve yine insan bağımlı bir hal alacaktır. Bu nedenle web servis adreslerinin HTML dökümanında sunulmasının yerine disco (discovery kelimesinin kısaltılmış hali) standardı kullanılır. Bu dosya web servisinin nerede bulunduğunu belirten ".disco" uzantılı bir dosyadır. Disco dosyası XML Web Servisini tanımlayan kaynaklara bağlantı (link) içerir. Şekil 6.1. bu dosyanın web servislerindeki konumunu göstermektedir.



Şekil 6.1. Disco dosyasının web servis modelindeki konumu

Bir istemci UDDI aracılığı ile bir XML Web Servisine eriştiğinde UDDI servisi istemciye XML Web Servisinin discovery dosyası olan disco uzantılı dosyanın adresini döner.

Bu dosyanın yapısı şöyledir:

```
<?xml version = "1.0" encoding = "utf-8"?>
        <discovery xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.xmlsoap.org/disco/">
<contractRef  ref=http://localhost/ankara_web/CompanyWebService.asmx?wsdl
docRef="http://localhost/ankara_web/CompanyWebService.asmx"
xmlns="http://schemas.xmlsoap.org/disco/scl/" />
```

<**soap address**=http://localhost/ankara_web/CompanyWebService.asmx
xmlns:q1="http://tempuri.org/" binding="q1:CompanyWebServiceSoap"
xmlns="http://schemas.xmlsoap.org/disco/soap/" /></discovery>

Bu dosyada dört tip değişken kullanılır. Bunların her biri referans olarak tanımlanır.

- ***contractRef:*** (ref) özelliği WSDL veya tanımlama dosyasını ya da diğer sözleşmeleri gösterir. Ayrıca, bu özellik insanların daha detaylı dökümanı okuyabilmesini sağlayan (docRef) niteliğini içerir.

- ***discoveryRef:*** (ref) özelliği daha fazla kaynağı işaret eden WSDL dökümanları veya DISCO dökümanlarını içerir. Bu özellik daha çok dinamik keşif ile kök dizininde meydana getirilen DISCO dökümanında görülür.

- ***schemaRef:*** (ref) özelliği XML şema (.xsd) dökümanını işaret eder ve bir seçimli özellik (targetNamespace) ise şemanın isim uzayını gösterir.

  <schema:schemaRef ref="mySchema.xsd"

  targetNamespace= "http://www.technicallead.com/mySchema.xsd"/ >

- ***soap:*** (address) özelliği WSDL dökümanı içerisinde tanımlanan bir web servisinin Tekdüzen Kaynak Tanımlayıcı (URI: Uniform Resource Identifier) konumunu gösterir [12].

İstemci disco dokümanındaki bilgileri kullanarak sunucudan XML Web Servisi tanımlama dosyasına (WSDL) istek yapar. Bu sayede istemci XML Web Servisinin işlevlerini öğrenip etkileşime geçer.

Bir XML Web servisini kullanabilmek için yapılması gereken işlemler şunlardır:

1. Kullanılacak XML Web Servisinin adresi bilinmiyorsa bir UDDI dizini yardımı ile bir XML Web Servisi bulunur.

2. Discovery dosyasına (.disco) yapılan istek ile web servisinin tanımlama dosyasına yönlenilir.

3. Web Servisinin açıklama dosyası olan WSDL dosyasına istek yapılır. Servis biçimi öğrenilir.

4. Artık XML Web Servisinin metotları çalıştırılmaya hazır olarak istemcinin hizmetindedir. XML Web Servisinin metotları çalıştırılabilir.

7. WEB SERVİSLERİNDE GÜVENLİK

Bir istemcinin, bir web sunucusu ile haberleşirken, haberleşmenin doğru web sunucusu ile gerçekleştiğinden emin olmasını sağlayan güvenlik önlemleri alınmalıdır. Gizlilik ve/veya bütünlüğün gerekli olduğu durumlarda web içerikleri internet üzerinden güvenli bir şekilde taşınmalıdır. WS-Güvenlik (WS-Security) dediğimizde aklımıza ilk gelen koruma kalitesi, mesaj güvenirliliği, gizliliği ve tek mesaj doğrulamasıdır. Bu mekanizmalar çeşitlilik gösteren güvenlik modellerini ve şifreleme teknolojilerini kullanır.

Bize güvenli web servisleri oluştururken, güvenirlilik ve gizlilik oluşturmak için kullanılan standart bir SOAP ekleri seti önerilir. Bu eklere "Web Services Security Language" veya "WS-Security" de denmektedir. WS-Güvenlik esnek ve Açık Anahtar Altyapısı (PKI: Public Key Infrastructure), Kerberos ve Güvenli Soket Katmanı (SSL: Secure Sockets Layer) 'nı içeren güvenlik yapısını temel olarak kullanmak için dizayn edilmiştir [19].

HTTP üzerinden doğrulama, mesajı imzalama, ve mesajın içeriğini şifreleme yapabiliriz. Bu tip durumlarda mesaj güvence altındadır. Şöyle ki, mesajı gönderen bilinir, mesajın alıcısı doğruluğu kanıtlar ve mesaj taşınma esnasında değişmez. Diğer kişiler mesajı ele geçirse bile doğrulama yapamaz ve verinin değişik bir durumda olmasından dolayı veri işlerine yaramaz. Bu açıdan baktığımızda SOAP mesajlaşma sistemi, HTTP temelli güvenlik yetersiz olmasına rağmen büyük problemleri çözmektedir. HTTP mekanizmaları sadece point-to-point güvenliğine hitap eder. Daha karmaşık projeler (çözümler) end-to-end güvenliğe ihtiyaç duyar.

WS-Güvenlik, güvenlikle ilgili olan veriyi taşımak için bir SOAP başlığı tanımlar. Eğer XML imzalama kullanıldıysa, bu başlık mesajın nasıl imzalandığını, imza sonuç değeri gibi bilgileri içerebilir. Ayrıca eğer mesajdaki bir element şifrelendiyse, şifreleme bilgisi WS-Güvenlik başlığı içinde yer alabilir. WS-Güvenlik tam olarak imza ve şifreleme formatını belirtmez. Onun yerine bir SOAP mesajına diğer otoriteler tarafından nasıl bir güvenlik bilgisinin gömüldüğünü tanımlar. WS-Güvenlik, her şeyden önce XML merkezli güvenlik metadata kutusu için bir belirteçtir. Mesajı şifreleme ve imzalama için kullanılan Binary Token'ları göndermek için bir BinarySecurityToken tanımlanır. Bu başlıkta, mesajlar çağıran kişi hakkında bilgiyi, mesajın nasıl imzalandığını ve nasıl şifrelendiğine ait bilgiyi tutar. WS-Güvenlik ise tam olarak temel kullanıcı güven belgesini (credential) UsernameToken elementinden geçerek transfer etmek için bir mekanizmayı anlatır [19].

7.1. Platform/Transport Düzeyli Güvenlik (Point-to-Point)

İki uç nokta (Web servisi istemcisi ve web servisi) arasındaki taşıma kanalı point-to-point güvenliğini sağlamak için kullanılır [19].

7.2. Platform Düzeyli Güvenlik

Platform güvenliğini kullandığımızda (Microsoft Windows İşletim Sistemi ailesinden);

- Internet Bilgi Servisleri (IIS: Internet Information Services) bize Basic, Digest, Integrated ve Certificate Authentication olanaklarını sunar.
- ASP.Net Web Servis'i ASP.Net Authentication ve Authorization özelliklerinden faydalanır.
- SSL ve Internet Protokol Güvenliği (IPSec: Internet Protocol Security) mesaj güvenirliliğini ve gizliliğini sağlamak için kullanılır [19].

7.3. Transport Düzeyli Güvenlik

Transport düzeyli güvenlik modeli basittir, kolay anlaşılır ve çoğu senaryoya (Özellikle intranet tabanlı) adapte olabilir. Bu güvenlik kuramında son noktalar sıkı bir şekilde denetim altındadır. Transport düzeyli güvenlikte önemli noktalar şunlardır:

- Güvenlik, üzerinde bulunduğu platforma, nakil (transport) mekanizmalarına ve güvenlik servis sağlayıcılarına (Kerberos vb.) sıkı sıkıya bağlıdır.
- Güvenlik, ara uygulama bağlantıları (Node) yoluyla point-to-point temeli üzerinde uygulanmaktadır [19].

7.4. Uygulama Düzeyli Güvenlik (Application Level Security)

Bu yaklaşımla uygulama, güvenliği devralır ve özel (kişisel) güvenlik modellerini kullanır. Buna bir örnek verirsek, bir uygulama özel SOAP başlığı (kullanıcı adı ve parola) kullanıcı bilgilerini doğrulamak için kullanabilir.

Bu tip bir güvenlik yaklaşımı özellikle bütün veri akımını şifrelemek yerine mesajımızın belirli (istediğimiz) kısımlarını şifrelemede kullanılabilecek bir yöntemdir [19].

7.5. Mesaj Düzeyli Güvenlik (End-to-End)

Gerçekliğini doğrulama (Authentication), SOAP başlığında yer alan *token* tarafından sağlanır. WS-Güvenlik tarafından özel tip *token*'lara ihtiyaç duyulmaz. Güvenlik token'ları Kerberos biletlerini, X.509 sertifikalarını veya özel binary token'larını içerebilir. Mesaj güvenirliliğini sağlama almak üzere dijital imzalar tarafından güvenli iletişim sağlanmakta ve mesaj gizliliği için de XML şifreleme kullanılmaktadır.

Mesaj düzeyli güvenlik, heterojen web servisleri ortamında güvenli mesaj değişimi için gerekli olan yapıyı inşa etmek için kullanılabilir. Bu güvenlik yaklaşımı, direk olarak her iki uç noktanın (end-points) ve ara uygulama bağlantıları kontrolünde olmadığımız senaryolarda ve heterojen ortamlarda kullanımı idealdir [19].

7.6. Authentication

Authentication, genel olarak kullanıcı kimliğini doğrulamadır. Bazı uygulamalarımız güvenlik nedeni ile kullanıcılardan belirli güvenlik katmanlarından geçmelerini isteyebilir. Bunlardan biri de kimlik denetimidir [19].

7.7. Authorization

Authorization, türkçeye çevrilmiş hali ile yetkilendirme anlamına gelmektedir. Sisteme dahil birden fazla kullanıcı olabilir. Ancak, her kullanıcının hakları eşit olmayabilir. Bir kullanıcı, yönetici (Administrator) konumunda iken diğeri normal kullanıcı konumundadır. Bu iki kullanıcıya sunulan haklar farklıdır. Yönetici kimlikli kullanıcı, sisteme direk müdahale edebiliyorken, normal kullanıcının böyle bir yetkisi yoktur [19].

8. LOCUS TRAVEL UYGULAMASI VE SİSTEM PERFORMANSI

Son yıllarda, insanların yolculuk için seyahat biletlerini satın almalarını, kendilerine uygun olanını aramalarını ya da yer ayırtmalarını sağlamak için birçok elverişli e-ticaret platformu ortaya çıkmıştır [20, 21]. Bu sistemler, yolcular ve seyahat firmaları için bir ara sistem olarak elektronik alışverişi gerçekleştirmektedir. Bu e-ticaret sistemlerinin sahipleri seyahat firmalarından turizm acentelerine, hatta uygulama servis sağlayıcılarına kadar değişiklik göstermektedir. Bu e-ticaret sistemleri yaygın olarak Türkiye'de genellikle şirkete özel olmakla beraber diğer sistemlerden yalıtılmış durumdadır. Şirkete özel olan bilgilerin sınırlı olmasından dolayı, yolcular bir seyahatte tüm olası seçenekleri görememektedir. E-ticaret sistemleri arasında bilgi paylaşımı olduğu zaman, yolcular gereken tüm seçenekleri bu sayede göreceklerdir. Diğer yandan, kişiler bu bilgi paylaşımından yararlanırken, şirketler de elde ettikleri bilgilerle kendileri için daha iyi stratejik plan yapabilecektir. Bu nedenle, hem bilet alıcıları hem de satıcılar bilgi paylaşımından yarar sağlamakta ve kişiler daha fazla seçenek bulma imkanına sahip olacaktır. Firmalar da o anki piyasa durumundan, arz ve talep hakkında daha fazla bilgi sahibi olacaktır

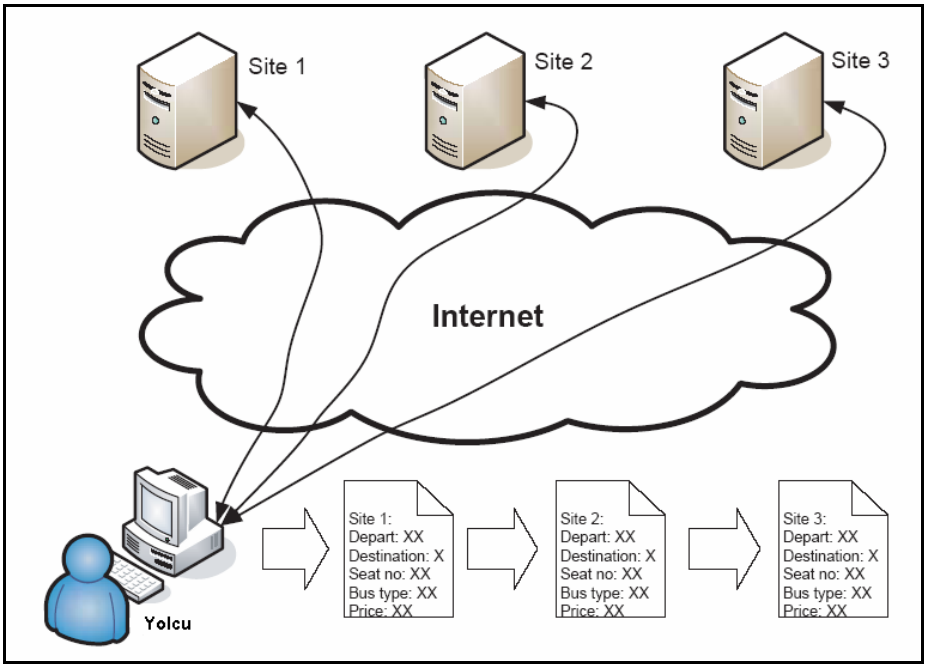
Güvenli bir şekilde bilgi paylaşımını sağlamanın yollarından biri çeşitli katmanlardan meydana gelmiş olan üç-katmanlı istemci-sunucu yazılım mimarisini kullanmaktır. İstemciler servislere erişim için kullanıcı ara yüzünü, uygulama sunucuları ara katmandaki iş mantığı ve fonksiyonelliği sağlamaktadır. Veritabanı sunucuları da arka kısımda veriyi ve işlevsel operasyonları yönetmektedir. Gerçekte, ticari uygulamalar açısından üç-katmanlı mimariye büyük bir eğilim söz konusudur. Sadece bu alanda değil aynı zamanda üç-katmanlı mimari bilimsel uygulamalarda, verilerin simülasyonlarında, uzayda yaşam kontrol ve destek sistemleri gibi geniş bir yelpazede kullanılmaktadır [22, 23, 24, 25].

Bu uygulama *Locus Travel* adındaki sistemi tanıtmaktadır. Locus Travel sistemi "state-of-the-art" bilgi teknolojisini kullanarak firmalar için e-ticaret sistemleri arasındaki bilgi paylaşımını olanaklı kılmaktadır. Burada, bilgi paylaşımı için web servisleri mimarisini kullanan Locus Travel sisteminin genel kavramı, yapısı ve fonksiyonları sunulmaktadır.

8.1. Seyahat Bileti Arama ve Satın Alma (Rezervasyon) İşlemi

Ülkemizde şehirlerarası yolculuklarda en çok otobüs tercih edilmektedir. Piyasada bunu gerçekleştiren birçok firma bulunmaktadır. Firmalardan bazıları, aynı istikamete, aynı gün ve saate sefer koymaktadır. Bu nedenle firmalar arasında oldukça yüksek bir rekabet ortamı meydana gelmiştir. Yolcular ise rekabet ortamında kendileri için en uygun olan firmayı (fiyat,

hizmet vs.) rahatlıkla bulabilmelidir. Genellikle kişiler telefon aracılığı ile arama yapıp rezervasyon ile biletlerini satın almaktadır. Bu rezervasyon sistemi zaman almakta ve tek tek firmaları aramayı gerektirir. Bu yolla sadece rezervasyon yapılır ama bir satın alma işlemi gerçekleştirilemez. Daha sonra ise bazı firmalara özel e-ticaret sistemleri ortaya çıkmıştır. Şekil 8.1. de ise bazı firmalara özel e-ticaret sistemlerinin genel yapısı gösterilmektedir. Ancak, bu geleneksel yöntemde de yer ayırtma ve satın alma işlemi için firma siteleri gezilmekte, fiyat, hizmet gibi faktörler aynı ortamda kıyaslanamadığından dolayı, süreç zaman almaktadır. Bu yöntem ise öncekine göre daha uygun ve hızlıdır.



Şekil 8.1. E-ticaret web sitelerinden geleneksel olarak bilet arama ve satın alınması

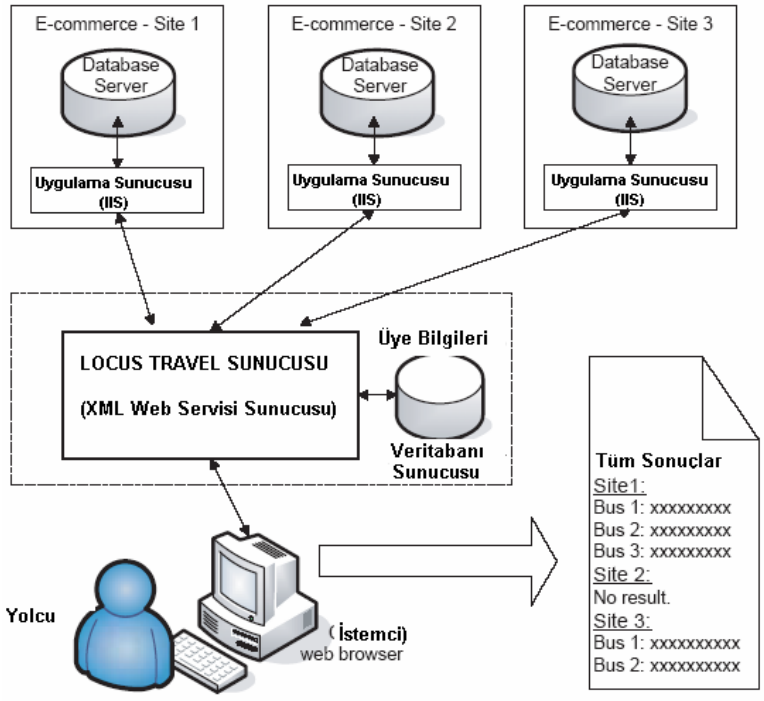
Üç-katmanlı mimari yaklaşımının, firmaların özel e-ticaret veritabanları arasında en iyi bilgi paylaşımı ve performansı sağladığı bu çalışmada incelenmiştir. Üç-katmanlı mimaride kullanıcılar, istemciler (Ör., Browsers) ile etkileşim halindedir. Uygulamanın iş mantığını kapsayan ara katmanda uygulama sunucuları (Ör., Web sunucuları) bulunmakta, arka kısımda firma veritabanları ile işlemleri gerçekleştirmektedir.

Üç-katmanlı mimari ile web servisleri dağıtık sistemlerde kullanılmaktadır. Bu, uygulamaların ağ üzerinde birden fazla bilgisayarda icra edilmesini sağlamaktadır. Web servisi çeşitli metotlardan oluşmaktadır. Bu metotlar dağıtık işlemi sağlamak için diğer makinalardan XML ve HTTP gibi ortak veri formatları ve protokolleri ile çağrılmaktadır. Tablo 8.1. Locus

Travel sistem sunucularındaki web metotlarını ve görevlerini göstermektedir. Web servislerini gerçeklemek için kullanılan platformlardan biri de Microsoft tarafından gerçekleştirilen ASP.Net'dir [26]. ASP.Net'de metot çağrılması SOAP tarafından gerçekleştirilebilinir. HTTP üzerinden RPC gerçekleştirmek için platform bağımsız olan SOAP XML kullanmaktadır. Her istek ve cevap mesajları SOAP paketleri içinde tüm ihtiyaçlara cevap verir nitelikte XML formatında gönderilmektedir. SOAP mesajlarının iletimi için internet üzerinde standart protokol olan HTTP seçilmiştir. HTTP'nin diğer bir faydası da güvenlik duvarı kullanan ağlar içinde de kullanılmasıdır.

Tablo 8.1. Locus Travel sunucuları web metotları ve görevleri

Metot Adı	**Görevi**
Login()	Sisteme güvenli giriş için kullanılır.
AddTravel()	Sisteme yeni bir seyahat ekler.
Order()	Sipariş işlemini gerçekleştirir.
Cancel_Order()	Siparişi iptal eder.
GetFullSeats()	Bir araca ait daha önce satın alınmış koltuk numaralarını gönderir.
GetTravelDetail()	Seyahat numarasına ait detaylı bilgileri (istikamet, tarih, saat, fiyat vb.) gönderir.
GetVehicleDetails()	Firmaya ait tüm araç bilgilerini gönderir.
GetVehicleDetail()	Araç bilgilerini (plaka, model vb.) gönderir.
GetVehicleTravels()	Aracın seyahat bilgilerini gönderir.
GetCustomerReserves()	Müşterinin satın aldığı koltuk numaralarını gönderir.



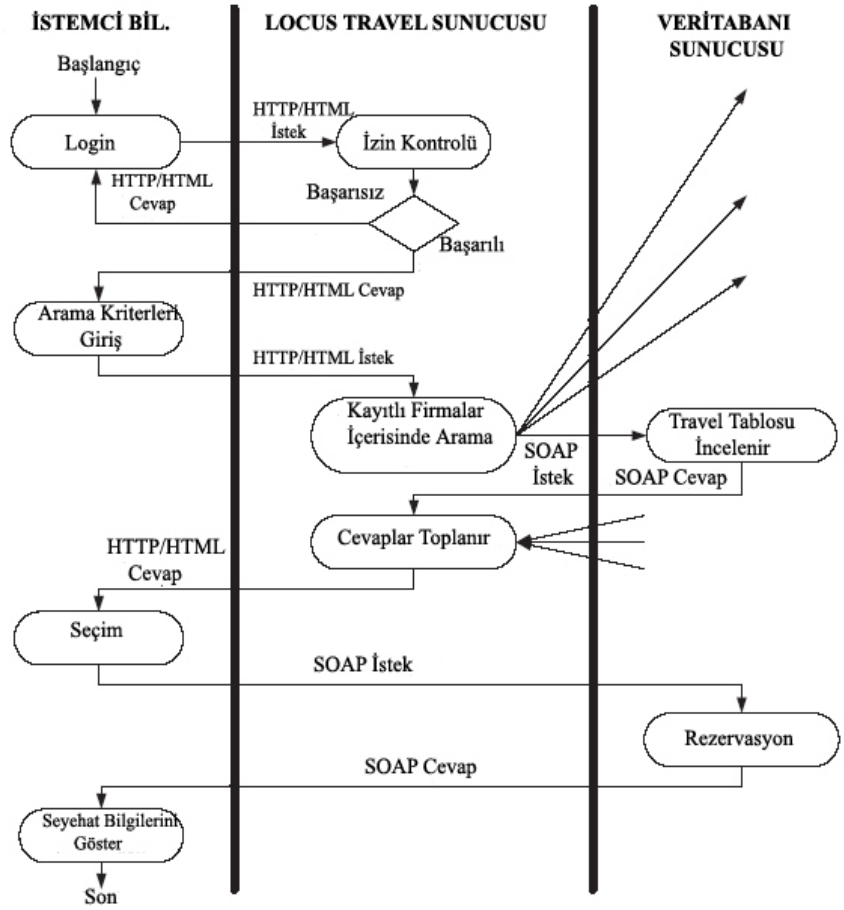
Şekil 8.2. Locus Travel ortamı

Şekil 8.2. Locus Travel platformunun web servisleri kullanılarak gerçeklenmesini göstermektedir. Locus Travel sistemi, istenen üye sitesine uygulama sunucularını yönlendirerek hizmet vermektedir. Üye siteler ise bağlı bulunduğu konumdan yer arama ve bilet alım işlemlerini gerçekleştirir. Arama kriterleri ve kullanıcı bilgileri HTTP kullanarak internet aracılığı ile XML formatında SOAP mesajı halinde paketlenerek Locus Travel üyesine gönderilmektedir. Mesaj güvenilirliliği ve gizliliği kriptoloji ve hashing gibi güncel güvenlik teknolojileri ile sağlanmaktadır.

8.2. Veri Akışı ve Yönetimi

Şekil 8.3. Locus Travel sisteminin veri akış diyagramını göstermektedir. Burada, istemci ya da kayıtlı kullanıcı öncelikle üye bilgilerini HTTP/HTML formatında uygulama sunucusuna gönderir. Eğer kullanıcı bilgileri doğru ise o kullanıcı için yeni bir oturum (session) açılır ve kullanıcı sisteme güvenli giriş yapar. Bir sonraki adımda kullanıcı bir form vasıtası ile arama işlemini gerçekleştirecek bilgileri, HTTP/HTML istek cevabı ile uygulama sunucusuna gönderir. Arama kriterlerini içeren istek mesajı, Locus Travel sunucusunda uygulama sunucusu ile gerçekleştirilir. Daha sonra bir SQL istek mesajı, belirlenen arama kriterlerine uygun seyahati bulmak için kayıtlı olan üye sunucularına gönderilir. SOAP istek mesajını alan üye sunucusu uygulama sunucusu aracılığı ile SQLXML istek mesajını kendi veritabanı sunucusuna

gönderir. Sunucu, veritabanından aldığı SQLXML cevap mesajını SOAP mesajı olarak uygulama sunucusuna gönderir. Üyelerden toplanan tüm yanıtlar, Locus Travel sunucusu ile birleştirilerek HTTP/HTML formatında kullanıcıya geri döndürülür.



Şekil 8.3. Locus Travel veri akış diyagramı

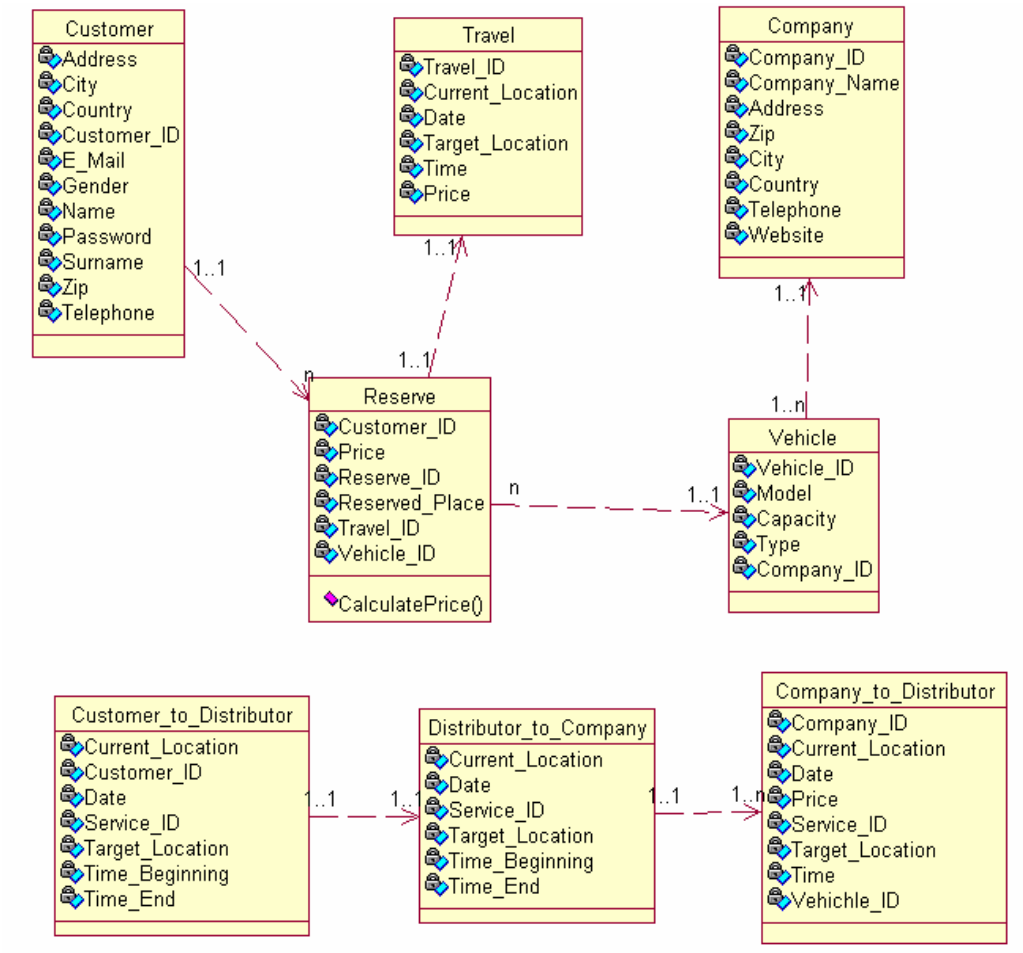
8.3. Sistem Mimarisi

Locus Travel sistemi üç-katmanlı mimari yaklaşımı ile ASP.Net platformunda web servisleri kullanılarak gerçekleştirilmiştir. Bilindiği gibi ASP.Net platformu Visual Basic, Java, C++ ve C# gibi birçok programlama diline destek vermektedir. Bu yüzden web servisleri farklı dillerden meydana gelmiş olabilir. Şu an ASP.Net sadece Windows ortamında çalışmaktadır. Ancak ileride farklı platformlarda da kullanıma imkan verebilir. Locus Travel hem istemci hem sunucu prototipleri tamamen C# dili kullanılarak kodlanmış ve beş sunucudan oluşmaktadır. Bir adet Locus Travel sunucusu ve dört adet kayıtlı firma sunucusundan meydana gelmiştir. İsteğe bağlı olarak bu kayıtlı firma sayısı arttırılabilir. Firmaların tümü, veritabanı sunucusu olarak

Microsoft SQL Server 2000 kullanmaktadır [27]. Tablo 8.2. sistemde kullandığımız birleşenleri, Şekil 8.4. ise sistemin sınıf (class) diyagramını göstermektedir.

Tablo 8.2. Locus Travel sistem ve sunucularının yazılım ortamı

İşletim Sistemi	Microsoft Windows Server 2003
Web Sunucusu	ASP.Net
Veritabanı Sunucusu	Microsoft SQL Server 2000 Personel Edition
SOAP Servisi	Microsoft SOAP 3.0
SOAP Versiyonu	SOAP 1.2
XML Servisi	Microsoft XML 4.0



Şekil 8.4. Locus Travel sınıf (class) diyagramı

Veri transferi internet yoluyla üç adımda XML mesajı halinde gerçekleşir. Örneğin, bir müşteri web servis sağlayıcısına istediği arama kriterlerine uygun bilgileri şu şekilde gönderir:

```
<Customer_to_Provider>
  <Service_ID>165</Service_ID>
  <Customer_ID>15654</Customer_ID>
  <Current_Location>Ankara</Current_Location>
  <Target_Location>Istanbul</Target_Location>
  <Date>01/01/2007</Date>
</Customer_to_Provider>
```

Web servisi ise mesajı firmalara aşağıdaki XML formatında gönderir:

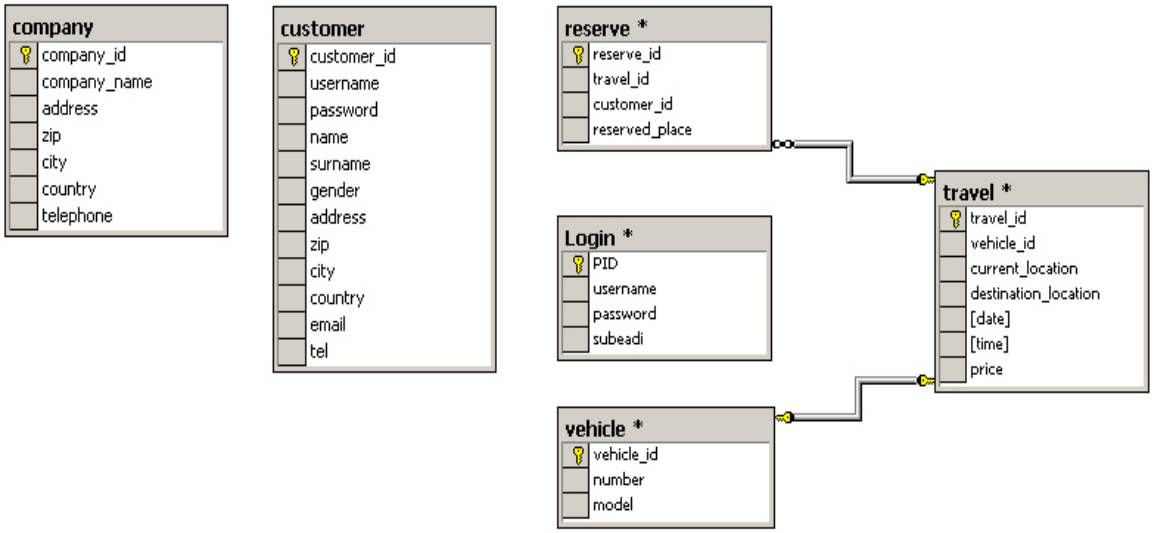
```
<Provider _to_Company>
  <Service_ID>165</Service_ID>
  <Current_Location>Ankara</Current_Location>
  <Target_Location>Istanbul</Target_Location>
  <Date>01/01/2007</Date>
</ Provider _to_Company>
```

En son olarak firmalardan gelen yanıt ise şu şekildedir:

```
<Company_to_ Provider>
  <Service_ID>165</Service_ID>
  <Company_ID>16</Company_ID>
  <Vehicle_ID>174</Vehicle_ID>
  <Current_Location>Ankara</Current_Location>
  <Target_Location>Istanbul</Target_Location>
  <Date>01/01/2007</Date>
  <Time>13:00</Time>
  <Price>20</Price>
</Company_to_ Provider>
```

8.4. Locus Travel Veritabanı Yapısı

Sistemde kullanılmak üzere hem acente için hem de firmalar için veritabanı tabloları oluşturulmuştur. Ancak, sistemin veritabanından bağımsız olması için sistem diğer veritabanı sunucularına (Oracle vb.) da destek vermektedir. Şekil 8.5. veritabanı sisteminin ilişkisel tablolarını göstermektedir.



Şekil 8.5. Locus Travel ilişkisel veritabanı

Aşağıdaki sql cümleciği "distributor" veritabanını oluşturur.

- **Acente veritabanı**

  Veritabanı adı: distributor.

```
CREATE DATABASE distributor
ON
        PRIMARY (NAME=distributorData,
        FILENAME = 'C:\Program Files\Microsoft SQL
        Server\MSSQL\Data\distributor.mdf',
        SIZE =1000MB,
        MAXSIZE = 1500MB,
        FILEGROWTH=%20)
LOGON
        (NAME=distributorLog,
        FILENAME = 'C:\Program Files\Microsoft SQL
        Server\MSSQL\Data\distributor.ldf',
        SIZE =5MB,
        MAXSIZE = 10MB,
        FILEGROWTH=1MB)
COLLATE Turkish_CI_AS
GO
```

Acente (distributor) veritabanı içerisinde bulunan tablolar şunlardır:

- **Customer tablosu:** Bu tablo müşteri bilgilerini saklar. Bu bilgiler üyelik işlemleri için kullanılmaktadır.

Tablo 8.3. Customer tablosu (distributor veritabanı)

Kolon Adı	**Veri Tipi**	**Açıklama**
customer_id	INT	Primary Key
Name	CHAR(30)	
Surname	CHAR(30)	
Gender	BOOLEAN	0 is female, 1 is male.
Username	CHAR(20)	
Password	CHAR(20)	
Address	CHAR(50)	
Zip	CHAR(6)	
City	CHAR(30)	
Country	CHAR(30)	
Email	CHAR(30)	
Tel	CHAR(15)	

- **Company tablosu:** Bu tablo sisteme üye olan firmaların bilgilerini saklar.

Tablo 8.4. Company tablosu (distributor veritabanı)

Kolon Adı	**Veri Tipi**	**Açıklama**
company_id	INT	Primary Key
company_name	CHAR(50)	
Address	CHAR(100)	
Zip	CHAR(6)	
City	CHAR(30)	
Country	CHAR(30)	
Telephone	CHAR(15)	

Aşağıdaki sql cümleciği firma veritabanını oluşturur.

- **Firma veritabanları**

  Veritabanı adı: ankara, izmir, istanbul, antalya.

```
CREATE DATABASE ankara
ON
PRIMARY (NAME=ankaraData,
```

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL\Data\ankara.mdf',
SIZE =1000MB,
MAXSIZE = 1500MB,
FILEGROWTH=%20)
LOGON
(**NAME**=ankaraLog,
FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL\Data\ankara.ldf',
SIZE =5MB,
MAXSIZE = 10MB,
FILEGROWTH=1MB)
COLLATE Turkish_CI_AS
GO

Firmaların veritabanı içerisinde bulunan tablolar şunlardır:

- **Login tablosu:** Bu tablo firma çalışanlarının sisteme giriş bilgilerini tutar. Çalışanlar sisteme kullanıcı adı ve şifrelerini kullanarak güvenli giriş yapar.

Tablo 8.5. Login tablosu (firma veritabanı)

Kolon Adı	**Veri Tipi**	**Açıklama**
PID	INT	Primary Key
Username	CHAR(30)	
Password	CHAR(30)	
Subeadi	CHAR(30)	

- **Vehicle tablosu:** Bu tablo araç bilgilerini (model, plaka vb.) saklar.

Tablo 8.6. Vehicle tablosu (firma veritabanı)

Kolon Adı	**Veri Tipi**	**Açıklama**
vehicle_id	INT	Primary Key
Number	CHAR(15)	
Model	CHAR(20)	

- **Travel tablosu:** Bu tablo hangi aracın, ne zaman, nereden nereye yolculuk edeceği, saat, tarih, fiyat vb. bilgileri saklar.

Tablo 8.7. Travel tablosu (firma veritabanı)

Kolon Adı	**Veri Tipi**	**Açıklama**
travel_id	INT	Primary Key
vehicle_id	INT	Foreign Key
current_location	CHAR(20)	
destination_location	CHAR(20)	
Date	CHAR(10)	
Time	CHAR(5)	
Price	INT	

- **Reserve tablosu:** Bu tablo müşterinin satın aldığı ya da reserve ettiği koltuk numaralarını tutar.

Tablo 8.8. Reserve tablosu (firma veritabanı)

reserve_id	INT	Primary Key
vehicle_id	INT	Foreign Key
travel_id	INT	Foreign Key
customer_id	INT	
Reserved_place	CHAR(5)	

8.5. Veritabanında Kullanılan Saklı Yordamlar ve İşlevleri

SQL Server'da saklı yordamlar bir takım Transact SQL deyimlerinden oluşur. Veritabanı üzerinde işlemler yapar ve gerekirse diğer saklı yordamları çalıştırabilir. Çalışma sonuçlarına göre bir takım çıktılar verebilecekleri gibi ortaya çıkan durumları ve varsa hataları bildirebilir. Saklı yordamlar SQL Server içerisinde kodlanır ve gerektiğinde program içerisinden çağrılıp kullanılır.

Saklı yordamları kullanmanın getirilerini aşağıdaki gibi sıralayabiliriz:

- **Saklı yordamlar modüler programlamaya imkan sağlar**: Bir saklı yordamı oluşturduktan sonra bunu veritabanı üzerinde saklayabiliriz. Bu yordamı gerek olduğunda istediğimiz kadar kullanabiliriz. Saklı yordamlar veritabanı üzerinde uzmanlaşmış kişiler tarafından yazılıp, veritabanı üzerinde işlem yapmak isteyen

kişilerin kullanımına sunulur. Böylece, diğer programlama dilleri ile yazılan kaynak kodlar tarafından kullanımı mümkündür [26].

- **İşlemlerin daha hızlı yapılmasını sağlar:** Eğer bir işlem çok miktarda Transact-SQL koduna gerek duyuyorsa ve bir çok defa artarda kullanılıyorsa, saklı yordamlar daha hızlı çalışabilir. Bunun nedeni, bir saklı yordam ilk çalıştırıldığında SQL Server tarafından gramer bakımından incelenir ve en uygun duruma getirilir. Aynı zamanda bu şekilde derlenen bir kopya daha sonraki kullanımlar için hafızada kalır. Böylece ikinci ve sonraki kullanımlarda, tekrar gramer incelemesi yapmaya ve en uygun duruma getirmeye gerek yoktur [26].
- **Ağ trafiğini azaltır:** Saklı yordamlar çok uzun kodlar içeriyorsa, bu kodlar ağ üzerinden iletilmek yerine veritabanının üzerinde olduklarından ağ trafiğini azaltırlar [26].
- **Güvenlik aracı olarak kullanılabilir:** Kullanıcılara bir saklı yordamı kullanmaya hak verilebilir. Bazı kullanıcıların saklı yordam içinde geçen deyimleri kullanmaya hakları olmadığı halde, saklı yordamı kullanma hakkı verilebilir [26].

Acente (distributor) veritabanı içerisinde oluşturulan saklı yordamlar şunlardır:

"Login" Saklı Yordamı: Müşterinin kullanıcı adı ve şifre kontrolünü yapar. Bilgilerin doğru olması halinde "customer_id" değerini döndürür.

```
CREATE PROCEDURE Login
(
        @username char(20),
        @password char(20),
        @customerID int OUTPUT
)
AS
SELECT @customerID = customer_id FROM customer WHERE username =
        @username AND password = @password
if @@RowCount < 1
SELECT @customerID = 0
GO
```

“AddCustomer” Saklı Yordamı: Müşteri bilgilerini “customer” tablosuna kayıt etmek için kullanılmıştır. İşlemin gerçekleşmesi halinde “customer_id” değerini döndürür.

```
CREATE PROCEDURE AddCustomer
(
        @username char(20),
        @password char(20),
        @name char(30),
        @surname char(30),
        @gender bit,
        @address char(50),
        @zip char(6),
        @city char(30),
        @country char(30),
        @email char(30),
        @tel char(15),
        @customer_id int OUTPUT
)
AS
SELECT customer_id FROM customer WHERE username = @username
if @@RowCount <1
INSERT INTO customer
(
username, password, name, surname, gender, address, zip, city, country, email, tel
)
values
(
@username, @password, @name, @surname, @gender, @address, @zip, @city, @country, @email, @tel
)
SELECT @customer_id = @@Identity
GO
```

“UpdateCustomer” Saklı Yordamı: Müşteri bilgilerinin güncellenmesi için kullanılmıştır. İşlemin gerçekleşmesi halinde “customer_id” değerini döndürür.

```
CREATE PROCEDURE UpdateCustomer
(
        @username char(20),
        @password char(20),
        @name char(30),
        @surname char(30),
        @gender bit,
        @address char(50),
        @zip char(6),
        @city char(30),
        @country char(30),
        @email char(30),
        @tel char(15),
        @customer_id int OUTPUT
)
AS
UPDATE customer SET password = @password WHERE username = @username
UPDATE customer SET name = @name WHERE username = @username
UPDATE customer SET surname = @surname WHERE username = @username
UPDATE customer SET gender = @gender WHERE username = @username
UPDATE customer SET address = @address WHERE username = @username
UPDATE customer SET zip = @zip WHERE username = @username
UPDATE customer SET city = @city WHERE username = @username
UPDATE customer SET country = @country WHERE username = @username
UPDATE customer SET email = @email WHERE username = @username
UPDATE customer SET tel = @tel WHERE username = @username
SELECT @customer_id = customer_id FROM customer WHERE username =
        @username AND password = @password
GO
```

"GetCompanyName" Saklı Yordamı: Firma isminin veritabanından çekilmesi işlemini gerçekleştirir. İşlemin gerçekleşmesi halinde "company_name" değeri döndürülür.

```
CREATE PROCEDURE GetCompanyName
(
        @company_id int,
```

```
        @company_name char(50) OUTPUT
)
AS
SELECT @company_name = company_name FROM company WHERE company_id
        = @company_id
GO
```

"GetCustomerDetails" Saklı Yordamı: Müşteri bilgilerinin hepsini acente veritabanından çeker.

```
CREATE PROCEDURE GetCustomerDetails
(
        @customer_id int,
        @username char(20) OUTPUT,
        @name char(30)  OUTPUT,
        @surname char(30)  OUTPUT,
        @gender bit  OUTPUT,
        @address char(50)  OUTPUT,
        @zip char(6)  OUTPUT,
        @city char(30)  OUTPUT,
        @country char(30) OUTPUT,
        @email char(30) OUTPUT,
        @tel char(15) OUTPUT
)
AS
SELECT @username = username,  @name = name, @surname = surname, @gender =
gender, @address = address, @zip = zip, @city = city, @country = country, @email =
email, @tel = tel FROM customer WHERE customer_id = @customer_id
GO
```

Firmaların (ankara, istanbul, izmir, antalya) veritabanlarında oluşturulan saklı yordamlar şunlardır:

"LoginP" Saklı Yordamı: Firma çalışanlarının sisteme girişinde kullanıcı adı ve şifre kontrolü yapar. Doğru olması halinde "Login" tablosundaki "PID" değerini döndürür.

```
CREATE PROCEDURE LoginP
```

```
(
        @username char(20),
        @password char(20),
        @PID int OUTPUT
)
AS
SELECT @PID = PID FROM Login WHERE username = @username AND
            password = @password
if @@RowCount < 1
SELECT @PID = 0
GO
```

"AddReserve" Saklı Yordamı: Müşteriler tarafından satın alınan yerleri, firmanın veritabanında bulunan "reserve" tablosuna ekler.

```
CREATE PROCEDURE AddReserve
(
        @travel_id int,
        @customer_id int,
        @reserved_place int
)
AS
INSERT INTO reserve
VALUES
(
        @travel_id, @customer_id, @reserved_place
)
GO
```

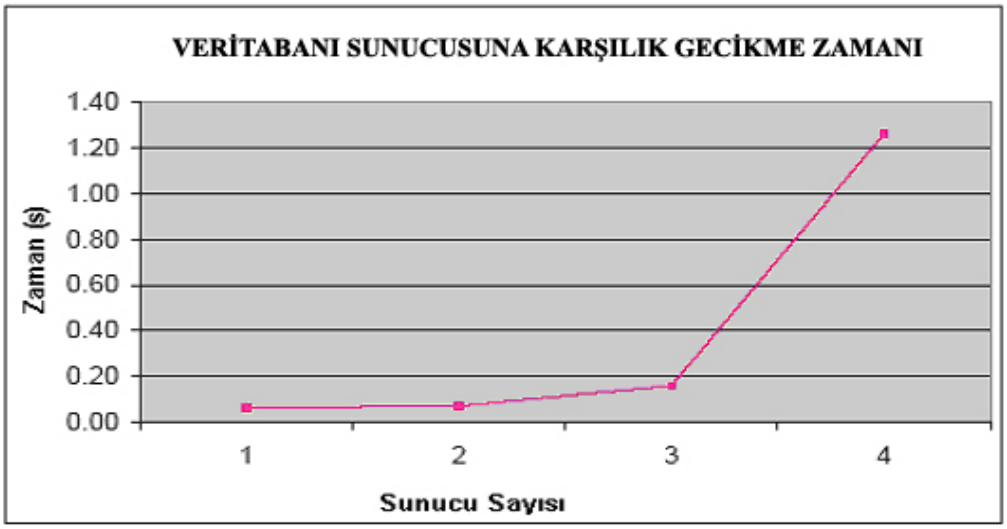
"AddTravel" Saklı Yordamı: Sisteme yeni bir seyahat ekler. İşlemin gerçekleşmesi halinde "travel" tablosundan "travel_id" değerini döndürür.

```
CREATE PROCEDURE AddTravel
(
        @vehicle_id int,
        @current_location char(20),
        @destination_location char(20),
```

```
        @date char(10),
        @time char(5),
        @price int,
        @travel_id int OUTPUT
)
AS
INSERT INTO travel
(
        vehicle_id , current_location, destination_location, date, time, price
)
VALUES
(
        @vehicle_id, @current_location, @destination_location, @date, @time,
        @price
)
SELECT @travel_id = @@Identity
GO
```

8.6. Performans Ölçümleri

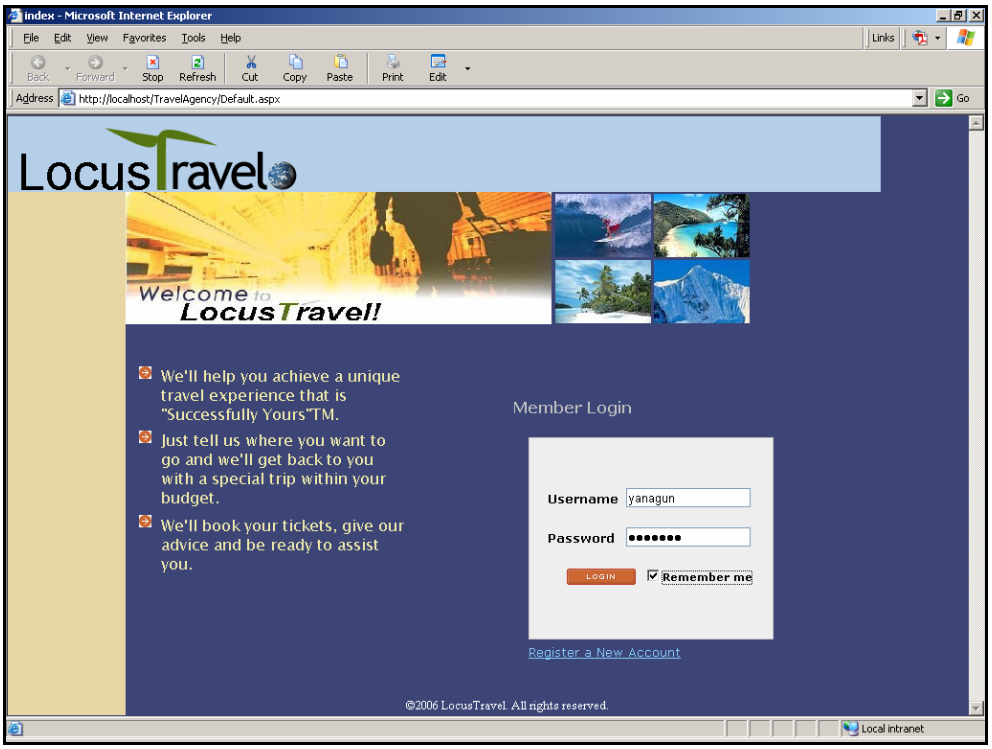
Kendi laboratuarlarımızda Locus Travel sistemini gerçekleyerek çeşitli deneylerle performansını test ettik. Temel olarak aynı zamanda veritabanlarının gecikme zamanlarının ölçümünü yaptık. Buradaki gecikme zamanındaki kasıt bir kullanıcı aynı makinada arama yapmaya başladığı andan itibaren, arama sonuçlarının kullanıcıya ulaşana kadar geçen zamandır. Ölçümler her parametre için beş kere tekrar edilmiş ve hesaplamada bir ortalama gecikme zamanı hesaplanmıştır. Test için kullanılan bilgisayarlar: Pentium IV 2.8 Mhz İşlemci, 256 MByte hafızalıdır. Kullanılan ağ yapısı 100 MBit/s hızındadır. Şekil 8.6. gecikme zamanı ile aynı zamandaki veritabanı sunucusu sayısını göstermektedir.



Şekil 8.6. Locus Travel sistemi sunucu sayısına karşılık gecikme zaman grafiği

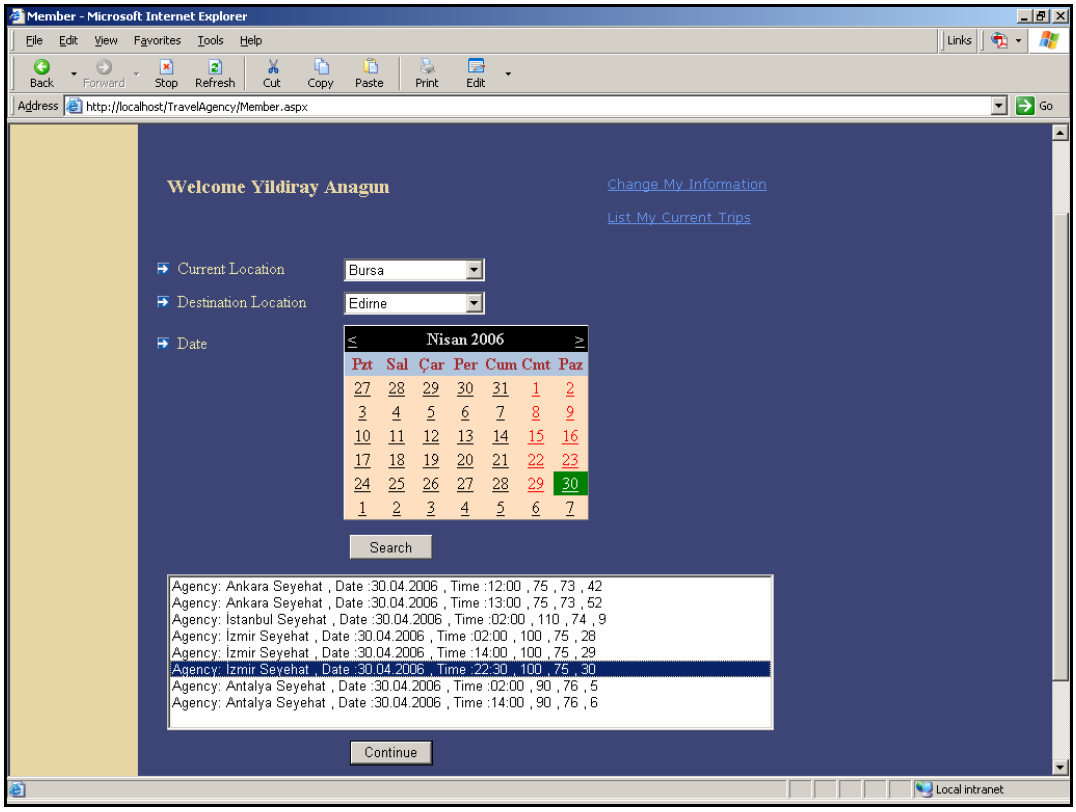
9. LOCUS TRAVEL KULLANICI ARA YÜZLERİ

Bu çalışmada iki tip kullanıcı ara yüzü geliştirilmiştir. Birincisi web kullanıcıları (genellikle müşteriler) için web ara yüzü, diğeri ise seyahat acentesi firmalar için form tabanlı kullanıcı ara yüzüdür. Form tabanlı kullanıcı ara yüzü son kullanıcıların bilgisayarına yüklenmektedir. Bir kullanıcı Locus Travel sisteminden yararlanmak için öncelikle sisteme üye olmalıdır. Şekil 9.1. üye giriş sayfasını göstermektedir.



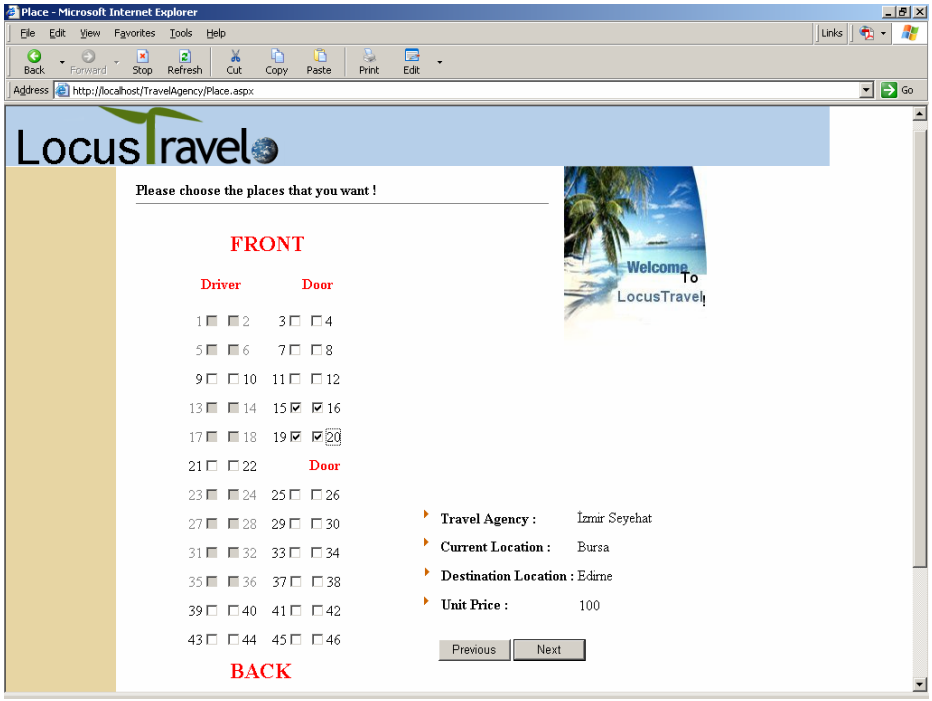
Şekil 9.1. Locus Travel web tabanlı üye giriş ara yüzü

Örneğin, kullanıcı bulunduğu yeri, gitmek istediği yeri ve zaman gibi arama kriterlerini bu ara yüzleri kullanarak girmektedir. Şekil 9.2. bir aramada ekrana gelen sonuçların gösterildiği ara yüzdür. Bu sonuçlarda sisteme kayıtlı firmalardan gelen bilgiler bulunmaktadır. Daha sonra yolcu, bunlardan kendisine uygun olanı seçmekte ve bir sonraki adıma geçmektedir.

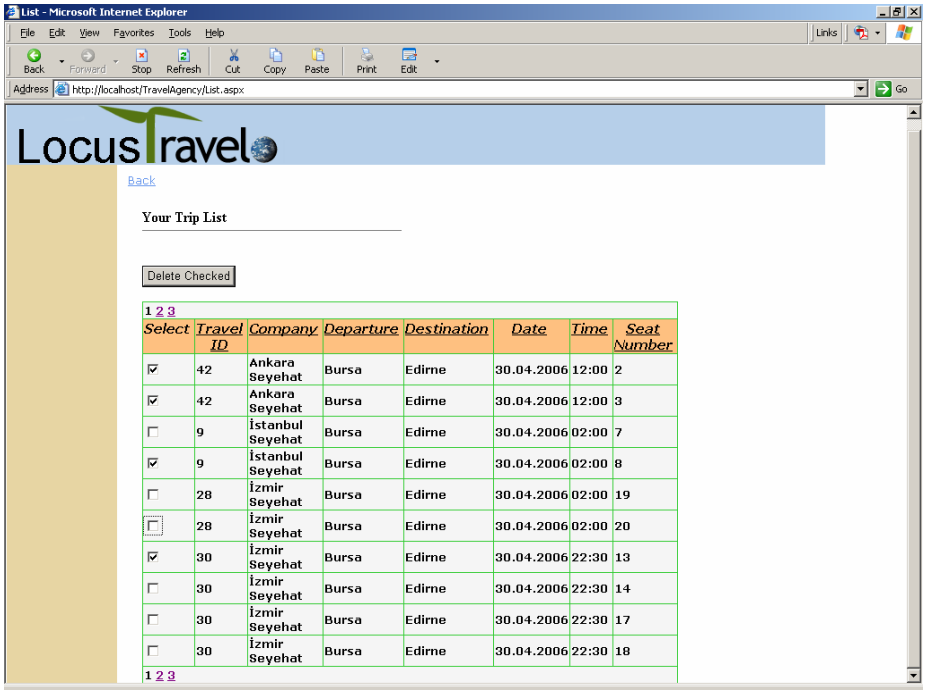


Şekil 9.2. Arama sonuçlarının gösterildiği ara yüz

Kullanıcı kendisine uygun olan firmayı seçtiğinde artık rezervasyon işlemini gerçekleştirebilir ve yerini satın alabilir. Şekil 9.3. yer ayırtma işleminin gerçekleştiği ara yüzdür. Ayrıca kullanıcılar, sisteme üye olan firmalardan rezerve ettikleri yerlerin detaylarını ve numaralarını görebilmekte ve istenmeyen rezervasyonlar Şekil 9.4.'de gösterilen ara yüz kullanılarak gerekli güncelleme ve silme işlemleri gerçekleştirilmektedir.

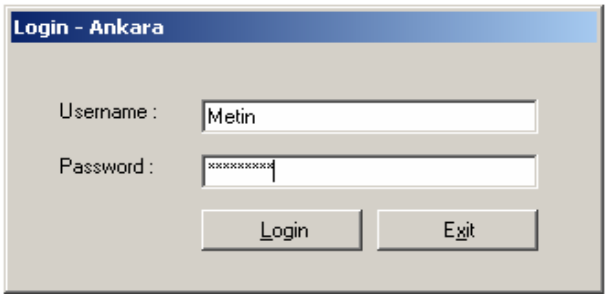


Şekil 9.3. Yer ayırtma işlemi için kullanılan ara yüz

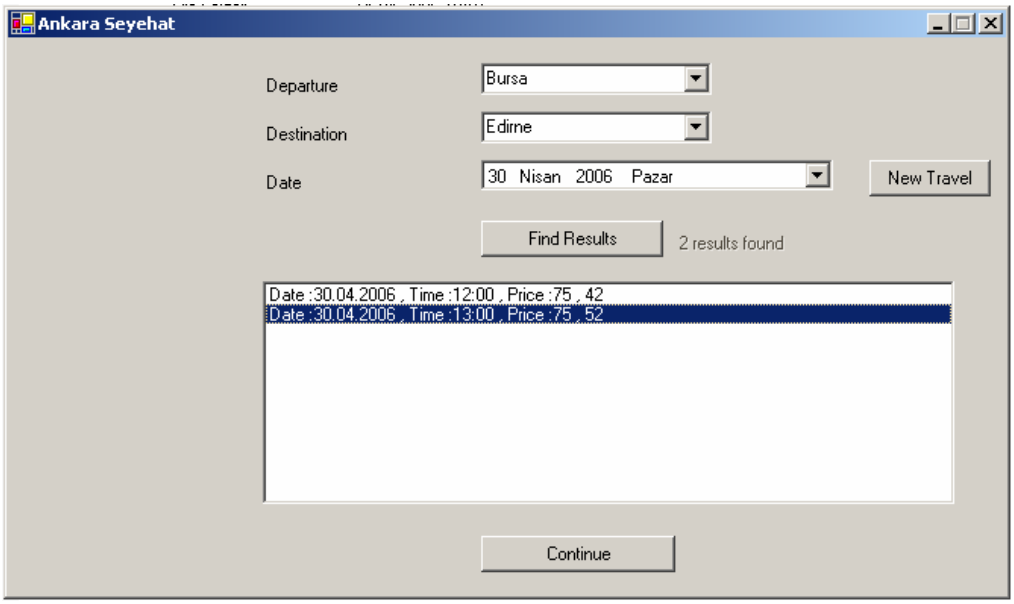


Şekil 9.4. Rezervasyon güncelleme ve silme ara yüzü

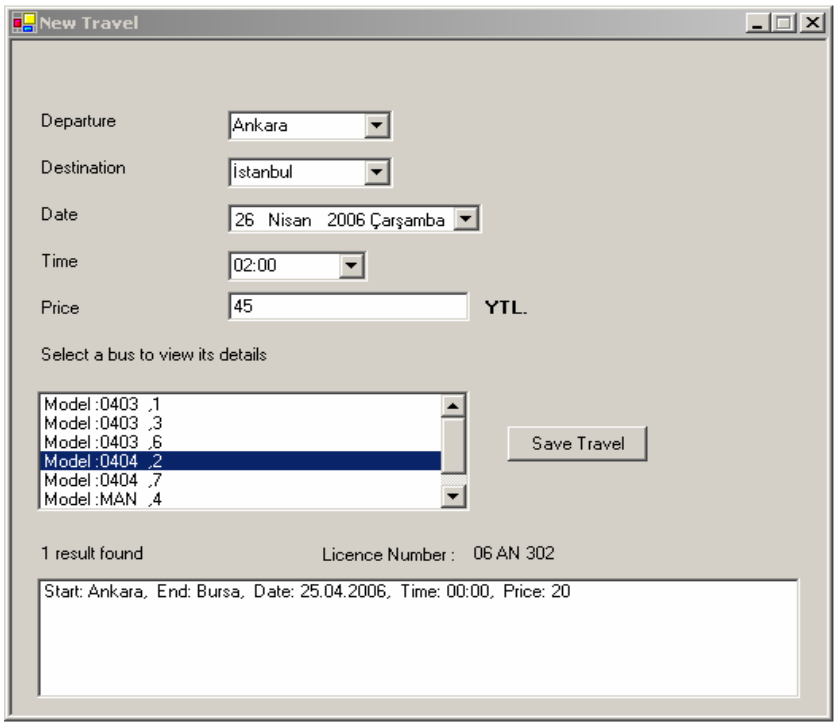
Sistemi kullanmak için gereken diğer ara yüzler ise firma bilgisayarlarına yüklenen form tabanlı bir programdır. Firma çalışanlarının kullanabileceği şekilde tasarlanmıştır. Şekil 9.5. firma çalışanlarının sisteme giriş ekranını göstermektedir. Şekil 9.6. kriterlere uygun arama yapmak, Şekil 9.7. yeni bir seyahat belirlemek, Şekil 9.8. ise yolcu yerlerinin ayırtılmasını sağlamak ve satın alma işlemini sona erdirmek için geliştirilmiş son kullanıcı ara yüzlerini göstermektedir.



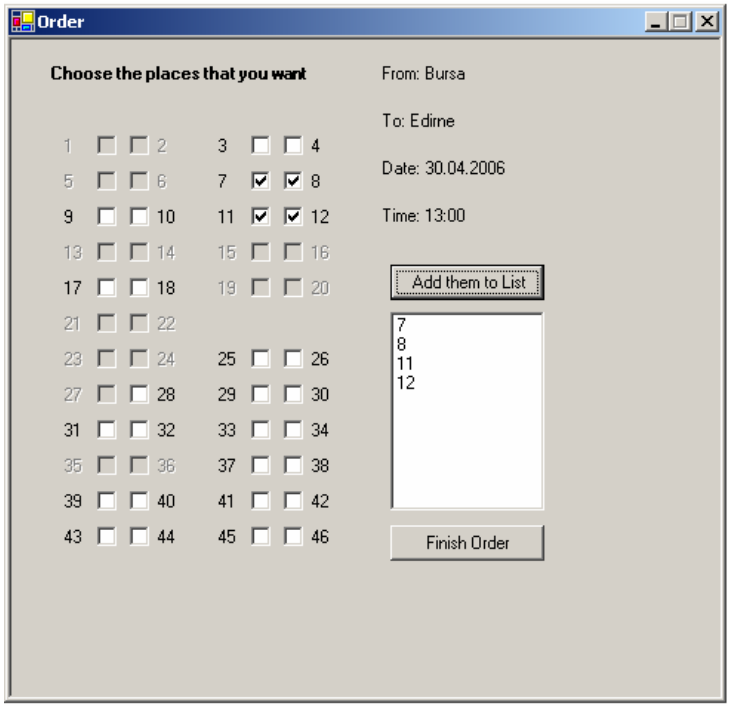
Şekil 9.5. Firma çalışanlarının sisteme giriş için kullandıkları ara yüz



Şekil 9.6. Girilen kriterlere uygun arama yapmak için kullanılan ara yüz



Şekil 9.7. Yeni seyahat ve araç belirlemek için kullanılan ara yüz



Şekil 9.8. İstenen yerlerin satın alınması işlemi için kullanılan ara yüz

10. SONUÇLAR ve ÖNERİLER

Web servisleri modeli Haziran 2000'de ortaya çıkmış ve yoğun bir ilgi ile karşılaşmıştır. Yeni nesil web uygulamaları web servisleri modeline dayanmaktadır. Web servisleri ile kurumlar, işbirliği yaptığı satıcı, müşteri ve banka gibi diğer kurumlardaki iş süreçlerini birleştirerek daha iyi işlevsellik ve daha az maliyet gibi bir çok avantajlar sağlayacaktır. Firmalar, web servisleri modeline büyük destek ve yatırımlar yapmaktadır. Mevcut web servisleri modeli uygulama bütünleştirme konusunda bir takım temel yapıları içermekle birlikte bir takım eksiklikleri de vardır. Bunlar, güvenlik, iş akışı, servis sürekliliği, servis kalitesi ve yönetim gibi konulardır. Web servislerinin başarılı ve yaygın olarak kullanılması için mevcut modelin geliştirilerek, bu konuların çözüme kavuşturulması gerekmektedir.

Bu çalışmada seyahat işlemlerinde kullanılabilecek bir e-ticaret modeli gerçeklenmiştir. Dağıtık veritabanı sistemleri için üç-katmanlı mimariyi ve e-ticaret sistemleri arasındaki etkileşimi sağlamak için web servisleri modeli bu uygulamada kullanılmıştır. Ayrıca, gerçeklenen Locus Travel uygulamasının sistem performansı laboratuar ortamında ölçeklenebilirliliğini görmek için test edilmiştir. Gelecekte ölçeklenebilirliliğini ve kullanımını arttırmak için de uygulama katmanında arama kriterlerine daha fazla mantıksal işlevsellik eklenmesi planlanmaktadır. Üç-sıralı istemci-sunucu mimarisi sayesinde ileride sisteme yeni bir firma kolaylıkla dahil edilebilir ve sistem bu sayede genişletilebilir. Uygulamaya geçilmesi halinde bir şehirden diğer şehire ya da bir ülkeden diğer ülkeye seyahatlerde bilet bulmak ya da seyahat planlaması yapmak oldukça kolaylaşacaktır. Sistem genişletilerek sadece karayolu ulaşımına değil, aynı zamanda denizyolu, demiryolu ve havayolu şirketlerine de hizmet verebilir duruma gelecektir.

KAYNAKLAR DİZİNİ

[1] http://www.wto.org/wto/ecom.

[2] Ryman, A., Understading web services, 2006, http://www7.software.ibm.com/vad.nsf/Data/Document4362?OpenDocument&p=1&BCT=3&Footer=1.

[3] Prasad, N.S., 1989, IBM Mainframes Architecture and Design, Mc Graw-Hill, Inc., 32-55 p.

[4] Microsoft, Three-Tiered Distribution, 2006, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/ArcTieredDistribution.asp?frame=true.

[5] Berson, A.., 1996, Client-Server Architecture, McGraw-Hill, Inc., 29-53 p.

[6] Dickman, A.., 1995, Two-Tier versus Three-Tier Apps. Informationweek, 553 p, 74-80 p.

[7] Benefits of the Three-Tiered Architecture, 2006 http://www.babysentry.com/tech_specs_benefits.htm.

[8] Heather Kreger, Web Services Conceptual Architecture, http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf, 2006.

[9] Microsoft, 2003, ASP.NET, http://www.asp.net.

[10] W3C, 2003, Extensible Markup Language (XML), http://www.w3.org/xml.

[11] Albrecht, C.C., Dean, L.D., Hansen, J.V., 17 May 2005, Marketplace and technology standards for B2B e-commerce: progress, challenges, and the state of the art.

[12] Tabor, R., 2001, “Microsoft .NET XML Web Services”, Sams Press, United States of America, 124-144 p.

[13] Web Services Description Language WSDL 1.1, 2001, http://www.w3.org/TR/wsdl, W3C Note.

[14] Shohoud, Y., Introduction to WSDL, 2006, http://w2ks.dei.isep.ipp.pt/labdotnet/recursos/wsdl.pdf.

[15] Understanding WSDL in a UDDI registry, 2006 http://www-106.ibm.com/developerworks/webservices/library/ws-wsdl/.

[16] UDDI Technical White Paper, 2000 http://www.hpmiddleware.com/downloads/pdf/web_services_technicalwhite.pdf.

[17] UDDI, 2006, http://uddi.microsoft.com

KAYNAKLAR DİZİNİ (devam)

[18] UDDI, 2006, http://uddi.ibm.com

[19] http://www.msakademik.net

[20] http://www.expedia.com

[21] http://www.travel.com

[22] Kong, S., Li, H., Hung, T., Shi, J., Castro-Lacouture, D., Skibniewski, M., 2004, Enabling Information Sharing Between E-commerce systems for construction material procurement. Automotion in Construction, 261-276 p.

[23] Ma, Y., Aimer, E., 2001, Intelligent Agent in Electronic Commerce - XMLFinder. Proceedeng of Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises 273 p.

[24] Holmes, V.P., Miller, D.J., Pavlakos, C., Poore, C.A, Vandewart, R.L., Crowley, C.P., 2002, An architecture and implementation to support large-scale data access in scientific simulation environments. The proceedings of the 35th IEEE Annual Simulation Symposium (SS'02).

[25] Schreckenghost, D., Bonasso, P., Kortenkamp, D. Ryan, D., 1998, Three-tier architecture for controlling space life support system. Proceedings of IEEE SIS'98, Washington.

[26] http://www.microsoft.com/sql.

[27] http://www.microsoft.com/vstudio.

EKLER

Ek. 1. LocusTravel Uygulaması WSDL Belgesi (CompanyWebService.asmx?WSDL)

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://tempuri.org/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://tempuri.org/" xmlns="http://schemas.xmlsoap.org/wsdl/">
- <types>
- <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
- <s:element name="AddTravel">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="vehicle_id" type="s:int" />
  <s:element minOccurs="0" maxOccurs="1" name="current_location" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="destination_location" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="date" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="time" type="s:string" />
  <s:element minOccurs="1" maxOccurs="1" name="price" type="s:int" />
    </s:sequence>
    </s:complexType>
    </s:element>
- <s:element name="AddTravelResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="AddTravelResult" type="s:int" />
    </s:sequence>
    </s:complexType>
    </s:element>
- <s:element name="GetTravel">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="current_location" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="destination_location" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="date" type="s:string" />
    </s:sequence>
    </s:complexType>
    </s:element>
- <s:element name="GetTravelResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="GetTravelResult"
    type="s0:ArrayOfTravelResponse" />
    </s:sequence>
```

```
      </s:complexType>
      </s:element>
- <s:complexType name="ArrayOfTravelResponse">
- <s:sequence>
   <s:element minOccurs="0" maxOccurs="unbounded" name="TravelResponse" nillable="true"
      type="s0:TravelResponse" />
      </s:sequence>
      </s:complexType>
- <s:complexType name="TravelResponse">
- <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="travel_id" type="s:int" />
   <s:element minOccurs="0" maxOccurs="1" name="date" type="s:string" />
   <s:element minOccurs="0" maxOccurs="1" name="time" type="s:string" />
   <s:element minOccurs="1" maxOccurs="1" name="price" type="s:int" />
   <s:element minOccurs="1" maxOccurs="1" name="company_id" type="s:int" />
      </s:sequence>
      </s:complexType>
- <s:element name="Order">
- <s:complexType>
- <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="travel_id" type="s:int" />
   <s:element minOccurs="1" maxOccurs="1" name="customer_id" type="s:int" />
   <s:element minOccurs="1" maxOccurs="1" name="reserved_place" type="s:int" />
      </s:sequence>
      </s:complexType>
      </s:element>
- <s:element name="OrderResponse">
- <s:complexType>
- <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="OrderResult" type="s:int" />
      </s:sequence>
      </s:complexType>
      </s:element>
- <s:element name="Cancel_Order">
- <s:complexType>
- <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="travel_id" type="s:int" />
   <s:element minOccurs="1" maxOccurs="1" name="customer_id" type="s:int" />
   <s:element minOccurs="1" maxOccurs="1" name="reserved_place" type="s:int" />
      </s:sequence>
      </s:complexType>
      </s:element>
- <s:element name="Cancel_OrderResponse">
- <s:complexType>
- <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="Cancel_OrderResult" type="s:int" />
      </s:sequence>
      </s:complexType>
      </s:element>
```

<s:element name="**GetFullSeats**">
<s:complexType>
<s:sequence>
<s:element minOccurs="**1**" maxOccurs="**1**" name="**travel_id**" type="**s:int**" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="**GetFullSeatsResponse**">
<s:complexType>
<s:sequence>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**GetFullSeatsResult**" type="**s0:ArrayOfInt**" />
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="**ArrayOfInt**">
<s:sequence>
<s:element minOccurs="**0**" maxOccurs="**unbounded**" name="**int**" type="**s:int**" />
</s:sequence>
</s:complexType>
<s:element name="**GetTravelDetail**">
<s:complexType>
<s:sequence>
<s:element minOccurs="**1**" maxOccurs="**1**" name="**travel_id**" type="**s:int**" />
</s:sequence>
</s:complexType>
</s:element>
<s:element name="**GetTravelDetailResponse**">
<s:complexType>
<s:sequence>
<s:element minOccurs="**0**" maxOccurs="**1**" name="**GetTravelDetailResult**" type="**s0:TravelDetail**" />
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="**TravelDetail**">
<s:sequence>
<s:element minOccurs="**1**" maxOccurs="**1**" name="**vehicle_id**" type="**s:int**" />
<s:element minOccurs="**0**" maxOccurs="**1**" name="**current_location**" type="**s:string**" />
<s:element minOccurs="**0**" maxOccurs="**1**" name="**destination_location**" type="**s:string**" />
<s:element minOccurs="**0**" maxOccurs="**1**" name="**date**" type="**s:string**" />
<s:element minOccurs="**0**" maxOccurs="**1**" name="**time**" type="**s:string**" />
<s:element minOccurs="**1**" maxOccurs="**1**" name="**price**" type="**s:int**" />
</s:sequence>
</s:complexType>
<s:element name="**GetVehicleDetails**">
<s:complexType />
</s:element>
<s:element name="**GetVehicleDetailsResponse**">

```
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="GetVehicleDetailsResult"
    type="s0:ArrayOfVehicleDetail" />
    </s:sequence>
    </s:complexType>
    </s:element>
- <s:complexType name="ArrayOfVehicleDetail">
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="unbounded" name="VehicleDetail" nillable="true"
    type="s0:VehicleDetail" />
    </s:sequence>
    </s:complexType>
- <s:complexType name="VehicleDetail">
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="vehicle_id" type="s:int" />
  <s:element minOccurs="0" maxOccurs="1" name="number" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="model" type="s:string" />
    </s:sequence>
    </s:complexType>
- <s:element name="GetVehicleDetail">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="vehicle_id" type="s:int" />
    </s:sequence>
    </s:complexType>
    </s:element>
- <s:element name="GetVehicleDetailResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="GetVehicleDetailResult"
    type="s0:VehicleDetail" />
    </s:sequence>
    </s:complexType>
    </s:element>
- <s:element name="GetVehicleTravels">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="vehicle_id" type="s:int" />
    </s:sequence>
    </s:complexType>
    </s:element>
- <s:element name="GetVehicleTravelsResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="GetVehicleTravelsResult"
    type="s0:ArrayOfTravelDetail" />
    </s:sequence>
    </s:complexType>
```

```
      </s:element>
- <s:complexType name="ArrayOfTravelDetail">
- <s:sequence>
   <s:element minOccurs="0" maxOccurs="unbounded" name="TravelDetail" nillable="true"
       type="s0:TravelDetail" />
      </s:sequence>
      </s:complexType>
- <s:element name="GetCustomerReserves">
- <s:complexType>
- <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="customer_id" type="s:int" />
      </s:sequence>
      </s:complexType>
      </s:element>
- <s:element name="GetCustomerReservesResponse">
- <s:complexType>
- <s:sequence>
   <s:element minOccurs="0" maxOccurs="1" name="GetCustomerReservesResult"
       type="s0:ArrayOfReserveDetail" />
      </s:sequence>
      </s:complexType>
      </s:element>
- <s:complexType name="ArrayOfReserveDetail">
- <s:sequence>
   <s:element minOccurs="0" maxOccurs="unbounded" name="ReserveDetail" nillable="true"
       type="s0:ReserveDetail" />
      </s:sequence>
      </s:complexType>
- <s:complexType name="ReserveDetail">
- <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="travel_id" type="s:int" />
   <s:element minOccurs="1" maxOccurs="1" name="reserved_place" type="s:int" />
      </s:sequence>
      </s:complexType>
- <s:element name="Login">
- <s:complexType>
- <s:sequence>
   <s:element minOccurs="0" maxOccurs="1" name="username" type="s:string" />
   <s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
      </s:sequence>
      </s:complexType>
      </s:element>
- <s:element name="LoginResponse">
- <s:complexType>
- <s:sequence>
   <s:element minOccurs="0" maxOccurs="1" name="LoginResult" type="s:string" />
      </s:sequence>
      </s:complexType>
      </s:element>
```

<s:element name="**int**" type="**s:int**" />
<s:element name="**ArrayOfTravelResponse**" nillable="**true**" type="**s0:ArrayOfTravelResponse**" />
<s:element name="**ArrayOfInt**" nillable="**true**" type="**s0:ArrayOfInt**" />
<s:element name="**TravelDetail**" nillable="**true**" type="**s0:TravelDetail**" />
<s:element name="**ArrayOfVehicleDetail**" nillable="**true**" type="**s0:ArrayOfVehicleDetail**" />
<s:element name="**VehicleDetail**" nillable="**true**" type="**s0:VehicleDetail**" />
<s:element name="**ArrayOfTravelDetail**" nillable="**true**" type="**s0:ArrayOfTravelDetail**" />
<s:element name="**ArrayOfReserveDetail**" nillable="**true**" type="**s0:ArrayOfReserveDetail**" />
<s:element name="**string**" nillable="**true**" type="**s:string**" />
</s:schema>
</types>
- <message name="**AddTravelSoapIn**">
<part name="**parameters**" element="**s0:AddTravel**" />
</message>
- <message name="**AddTravelSoapOut**">
<part name="**parameters**" element="**s0:AddTravelResponse**" />
</message>
- <message name="**GetTravelSoapIn**">
<part name="**parameters**" element="**s0:GetTravel**" />
</message>
- <message name="**GetTravelSoapOut**">
<part name="**parameters**" element="**s0:GetTravelResponse**" />
</message>
- <message name="**OrderSoapIn**">
<part name="**parameters**" element="**s0:Order**" />
</message>
- <message name="**OrderSoapOut**">
<part name="**parameters**" element="**s0:OrderResponse**" />
</message>
- <message name="**Cancel_OrderSoapIn**">
<part name="**parameters**" element="**s0:Cancel_Order**" />
</message>
- <message name="**Cancel_OrderSoapOut**">
<part name="**parameters**" element="**s0:Cancel_OrderResponse**" />
</message>
- <message name="**GetFullSeatsSoapIn**">
<part name="**parameters**" element="**s0:GetFullSeats**" />
</message>
- <message name="**GetFullSeatsSoapOut**">
<part name="**parameters**" element="**s0:GetFullSeatsResponse**" />
</message>
- <message name="**GetTravelDetailSoapIn**">
<part name="**parameters**" element="**s0:GetTravelDetail**" />
</message>
- <message name="**GetTravelDetailSoapOut**">
<part name="**parameters**" element="**s0:GetTravelDetailResponse**" />
</message>
- <message name="**GetVehicleDetailsSoapIn**">

<part name="**parameters**" element="**s0:GetVehicleDetails**" />
</message>
- <message name="**GetVehicleDetailsSoapOut**">
<part name="**parameters**" element="**s0:GetVehicleDetailsResponse**" />
</message>
- <message name="**GetVehicleDetailSoapIn**">
<part name="**parameters**" element="**s0:GetVehicleDetail**" />
</message>
- <message name="**GetVehicleDetailSoapOut**">
<part name="**parameters**" element="**s0:GetVehicleDetailResponse**" />
</message>
- <message name="**GetVehicleTravelsSoapIn**">
<part name="**parameters**" element="**s0:GetVehicleTravels**" />
</message>
- <message name="**GetVehicleTravelsSoapOut**">
<part name="**parameters**" element="**s0:GetVehicleTravelsResponse**" />
</message>
- <message name="**GetCustomerReservesSoapIn**">
<part name="**parameters**" element="**s0:GetCustomerReserves**" />
</message>
- <message name="**GetCustomerReservesSoapOut**">
<part name="**parameters**" element="**s0:GetCustomerReservesResponse**" />
</message>
- <message name="**LoginSoapIn**">
<part name="**parameters**" element="**s0:Login**" />
</message>
- <message name="**LoginSoapOut**">
<part name="**parameters**" element="**s0:LoginResponse**" />
</message>
- <message name="**AddTravelHttpGetIn**">
<part name="**vehicle_id**" type="**s:string**" />
<part name="**current_location**" type="**s:string**" />
<part name="**destination_location**" type="**s:string**" />
<part name="**date**" type="**s:string**" />
<part name="**time**" type="**s:string**" />
<part name="**price**" type="**s:string**" />
</message>
- <message name="**AddTravelHttpGetOut**">
<part name="**Body**" element="**s0:int**" />
</message>
- <message name="**GetTravelHttpGetIn**">
<part name="**current_location**" type="**s:string**" />
<part name="**destination_location**" type="**s:string**" />
<part name="**date**" type="**s:string**" />
</message>
- <message name="**GetTravelHttpGetOut**">
<part name="**Body**" element="**s0:ArrayOfTravelResponse**" />
</message>
- <message name="**OrderHttpGetIn**">

<part name="**travel_id**" type="**s:string**" />
<part name="**customer_id**" type="**s:string**" />
<part name="**reserved_place**" type="**s:string**" />
</message>
- <message name="**OrderHttpGetOut**">
<part name="**Body**" element="**s0:int**" />
</message>
- <message name="**Cancel_OrderHttpGetIn**">
<part name="**travel_id**" type="**s:string**" />
<part name="**customer_id**" type="**s:string**" />
<part name="**reserved_place**" type="**s:string**" />
</message>
- <message name="**Cancel_OrderHttpGetOut**">
<part name="**Body**" element="**s0:int**" />
</message>
- <message name="**GetFullSeatsHttpGetIn**">
<part name="**travel_id**" type="**s:string**" />
</message>
- <message name="**GetFullSeatsHttpGetOut**">
<part name="**Body**" element="**s0:ArrayOfInt**" />
</message>
- <message name="**GetTravelDetailHttpGetIn**">
<part name="**travel_id**" type="**s:string**" />
</message>
- <message name="**GetTravelDetailHttpGetOut**">
<part name="**Body**" element="**s0:TravelDetail**" />
</message>
<message name="**GetVehicleDetailsHttpGetIn**" />
- <message name="**GetVehicleDetailsHttpGetOut**">
<part name="**Body**" element="**s0:ArrayOfVehicleDetail**" />
</message>
- <message name="**GetVehicleDetailHttpGetIn**">
<part name="**vehicle_id**" type="**s:string**" />
</message>
- <message name="**GetVehicleDetailHttpGetOut**">
<part name="**Body**" element="**s0:VehicleDetail**" />
</message>
- <message name="**GetVehicleTravelsHttpGetIn**">
<part name="**vehicle_id**" type="**s:string**" />
</message>
- <message name="**GetVehicleTravelsHttpGetOut**">
<part name="**Body**" element="**s0:ArrayOfTravelDetail**" />
</message>
- <message name="**GetCustomerReservesHttpGetIn**">
<part name="**customer_id**" type="**s:string**" />
</message>
- <message name="**GetCustomerReservesHttpGetOut**">
<part name="**Body**" element="**s0:ArrayOfReserveDetail**" />
</message>

```
- <message name="LoginHttpGetIn">
  <part name="username" type="s:string" />
  <part name="password" type="s:string" />
    </message>
- <message name="LoginHttpGetOut">
  <part name="Body" element="s0:string" />
    </message>
- <message name="AddTravelHttpPostIn">
  <part name="vehicle_id" type="s:string" />
  <part name="current_location" type="s:string" />
  <part name="destination_location" type="s:string" />
  <part name="date" type="s:string" />
  <part name="time" type="s:string" />
  <part name="price" type="s:string" />
    </message>
- <message name="AddTravelHttpPostOut">
  <part name="Body" element="s0:int" />
    </message>
- <message name="GetTravelHttpPostIn">
  <part name="current_location" type="s:string" />
  <part name="destination_location" type="s:string" />
  <part name="date" type="s:string" />
    </message>
- <message name="GetTravelHttpPostOut">
  <part name="Body" element="s0:ArrayOfTravelResponse" />
    </message>
- <message name="OrderHttpPostIn">
  <part name="travel_id" type="s:string" />
  <part name="customer_id" type="s:string" />
  <part name="reserved_place" type="s:string" />
    </message>
- <message name="OrderHttpPostOut">
  <part name="Body" element="s0:int" />
    </message>
- <message name="Cancel_OrderHttpPostIn">
  <part name="travel_id" type="s:string" />
  <part name="customer_id" type="s:string" />
  <part name="reserved_place" type="s:string" />
    </message>
- <message name="Cancel_OrderHttpPostOut">
  <part name="Body" element="s0:int" />
    </message>
- <message name="GetFullSeatsHttpPostIn">
  <part name="travel_id" type="s:string" />
    </message>
- <message name="GetFullSeatsHttpPostOut">
  <part name="Body" element="s0:ArrayOfInt" />
    </message>
- <message name="GetTravelDetailHttpPostIn">
```

<part name="**travel_id**" type="**s:string**" />
</message>
- <message name="**GetTravelDetailHttpPostOut**">
<part name="**Body**" element="**s0:TravelDetail**" />
</message>
<message name="**GetVehicleDetailsHttpPostIn**" />
- <message name="**GetVehicleDetailsHttpPostOut**">
<part name="**Body**" element="**s0:ArrayOfVehicleDetail**" />
</message>
- <message name="**GetVehicleDetailHttpPostIn**">
<part name="**vehicle_id**" type="**s:string**" />
</message>
- <message name="**GetVehicleDetailHttpPostOut**">
<part name="**Body**" element="**s0:VehicleDetail**" />
</message>
- <message name="**GetVehicleTravelsHttpPostIn**">
<part name="**vehicle_id**" type="**s:string**" />
</message>
- <message name="**GetVehicleTravelsHttpPostOut**">
<part name="**Body**" element="**s0:ArrayOfTravelDetail**" />
</message>
- <message name="**GetCustomerReservesHttpPostIn**">
<part name="**customer_id**" type="**s:string**" />
</message>
- <message name="**GetCustomerReservesHttpPostOut**">
<part name="**Body**" element="**s0:ArrayOfReserveDetail**" />
</message>
- <message name="**LoginHttpPostIn**">
<part name="**username**" type="**s:string**" />
<part name="**password**" type="**s:string**" />
</message>
- <message name="**LoginHttpPostOut**">
<part name="**Body**" element="**s0:string**" />
</message>
- <portType name="**CompanyWebServiceSoap**">
- <operation name="**AddTravel**">
<input message="**s0:AddTravelSoapIn**" />
<output message="**s0:AddTravelSoapOut**" />
</operation>
- <operation name="**GetTravel**">
<input message="**s0:GetTravelSoapIn**" />
<output message="**s0:GetTravelSoapOut**" />
</operation>
- <operation name="**Order**">
<input message="**s0:OrderSoapIn**" />
<output message="**s0:OrderSoapOut**" />
</operation>
- <operation name="**Cancel_Order**">
<input message="**s0:Cancel_OrderSoapIn**" />

<output message="**s0:Cancel_OrderSoapOut**" />
</operation>
- <operation name="**GetFullSeats**">
<input message="**s0:GetFullSeatsSoapIn**" />
<output message="**s0:GetFullSeatsSoapOut**" />
</operation>
- <operation name="**GetTravelDetail**">
<input message="**s0:GetTravelDetailSoapIn**" />
<output message="**s0:GetTravelDetailSoapOut**" />
</operation>
- <operation name="**GetVehicleDetails**">
<input message="**s0:GetVehicleDetailsSoapIn**" />
<output message="**s0:GetVehicleDetailsSoapOut**" />
</operation>
- <operation name="**GetVehicleDetail**">
<input message="**s0:GetVehicleDetailSoapIn**" />
<output message="**s0:GetVehicleDetailSoapOut**" />
</operation>
- <operation name="**GetVehicleTravels**">
<input message="**s0:GetVehicleTravelsSoapIn**" />
<output message="**s0:GetVehicleTravelsSoapOut**" />
</operation>
- <operation name="**GetCustomerReserves**">
<input message="**s0:GetCustomerReservesSoapIn**" />
<output message="**s0:GetCustomerReservesSoapOut**" />
</operation>
- <operation name="**Login**">
<input message="**s0:LoginSoapIn**" />
<output message="**s0:LoginSoapOut**" />
</operation>
</portType>
- <portType name="**CompanyWebServiceHttpGet**">
- <operation name="**AddTravel**">
<input message="**s0:AddTravelHttpGetIn**" />
<output message="**s0:AddTravelHttpGetOut**" />
</operation>
- <operation name="**GetTravel**">
<input message="**s0:GetTravelHttpGetIn**" />
<output message="**s0:GetTravelHttpGetOut**" />
</operation>
- <operation name="**Order**">
<input message="**s0:OrderHttpGetIn**" />
<output message="**s0:OrderHttpGetOut**" />
</operation>
- <operation name="**Cancel_Order**">
<input message="**s0:Cancel_OrderHttpGetIn**" />
<output message="**s0:Cancel_OrderHttpGetOut**" />
</operation>
- <operation name="**GetFullSeats**">

<input message="**s0:GetFullSeatsHttpGetIn**" />
<output message="**s0:GetFullSeatsHttpGetOut**" />
</operation>
- <operation name="**GetTravelDetail**">
<input message="**s0:GetTravelDetailHttpGetIn**" />
<output message="**s0:GetTravelDetailHttpGetOut**" />
</operation>
- <operation name="**GetVehicleDetails**">
<input message="**s0:GetVehicleDetailsHttpGetIn**" />
<output message="**s0:GetVehicleDetailsHttpGetOut**" />
</operation>
- <operation name="**GetVehicleDetail**">
<input message="**s0:GetVehicleDetailHttpGetIn**" />
<output message="**s0:GetVehicleDetailHttpGetOut**" />
</operation>
- <operation name="**GetVehicleTravels**">
<input message="**s0:GetVehicleTravelsHttpGetIn**" />
<output message="**s0:GetVehicleTravelsHttpGetOut**" />
</operation>
- <operation name="**GetCustomerReserves**">
<input message="**s0:GetCustomerReservesHttpGetIn**" />
<output message="**s0:GetCustomerReservesHttpGetOut**" />
</operation>
- <operation name="**Login**">
<input message="**s0:LoginHttpGetIn**" />
<output message="**s0:LoginHttpGetOut**" />
</operation>
</portType>
- <portType name="**CompanyWebServiceHttpPost**">
- <operation name="**AddTravel**">
<input message="**s0:AddTravelHttpPostIn**" />
<output message="**s0:AddTravelHttpPostOut**" />
</operation>
- <operation name="**GetTravel**">
<input message="**s0:GetTravelHttpPostIn**" />
<output message="**s0:GetTravelHttpPostOut**" />
</operation>
- <operation name="**Order**">
<input message="**s0:OrderHttpPostIn**" />
<output message="**s0:OrderHttpPostOut**" />
</operation>
- <operation name="**Cancel_Order**">
<input message="**s0:Cancel_OrderHttpPostIn**" />
<output message="**s0:Cancel_OrderHttpPostOut**" />
</operation>
- <operation name="**GetFullSeats**">
<input message="**s0:GetFullSeatsHttpPostIn**" />
<output message="**s0:GetFullSeatsHttpPostOut**" />
</operation>

```
- <operation name="GetTravelDetail">
  <input message="s0:GetTravelDetailHttpPostIn" />
  <output message="s0:GetTravelDetailHttpPostOut" />
    </operation>
- <operation name="GetVehicleDetails">
  <input message="s0:GetVehicleDetailsHttpPostIn" />
  <output message="s0:GetVehicleDetailsHttpPostOut" />
    </operation>
- <operation name="GetVehicleDetail">
  <input message="s0:GetVehicleDetailHttpPostIn" />
  <output message="s0:GetVehicleDetailHttpPostOut" />
    </operation>
- <operation name="GetVehicleTravels">
  <input message="s0:GetVehicleTravelsHttpPostIn" />
  <output message="s0:GetVehicleTravelsHttpPostOut" />
    </operation>
- <operation name="GetCustomerReserves">
  <input message="s0:GetCustomerReservesHttpPostIn" />
  <output message="s0:GetCustomerReservesHttpPostOut" />
    </operation>
- <operation name="Login">
  <input message="s0:LoginHttpPostIn" />
  <output message="s0:LoginHttpPostOut" />
    </operation>
    </portType>
- <binding name="CompanyWebServiceSoap" type="s0:CompanyWebServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <operation name="AddTravel">
  <soap:operation soapAction="http://tempuri.org/AddTravel" style="document" />
- <input>
  <soap:body use="literal" />
    </input>
- <output>
  <soap:body use="literal" />
    </output>
    </operation>
- <operation name="GetTravel">
  <soap:operation soapAction="http://tempuri.org/GetTravel" style="document" />
- <input>
  <soap:body use="literal" />
    </input>
- <output>
  <soap:body use="literal" />
    </output>
    </operation>
- <operation name="Order">
  <soap:operation soapAction="http://tempuri.org/Order" style="document" />
- <input>
  <soap:body use="literal" />
```

</input>
- <output>
<soap:body use="**literal**" />
</output>
</operation>
- <operation name="**Cancel_Order**">
<soap:operation soapAction="**http://tempuri.org/Cancel_Order**" style="**document**" />
- <input>
<soap:body use="**literal**" />
</input>
- <output>
<soap:body use="**literal**" />
</output>
</operation>
- <operation name="**GetFullSeats**">
<soap:operation soapAction="**http://tempuri.org/GetFullSeats**" style="**document**" />
- <input>
<soap:body use="**literal**" />
</input>
- <output>
<soap:body use="**literal**" />
</output>
</operation>
- <operation name="**GetTravelDetail**">
<soap:operation soapAction="**http://tempuri.org/GetTravelDetail**" style="**document**" />
- <input>
<soap:body use="**literal**" />
</input>
- <output>
<soap:body use="**literal**" />
</output>
</operation>
- <operation name="**GetVehicleDetails**">
<soap:operation soapAction="**http://tempuri.org/GetVehicleDetails**" style="**document**" />
- <input>
<soap:body use="**literal**" />
</input>
- <output>
<soap:body use="**literal**" />
</output>
</operation>
- <operation name="**GetVehicleDetail**">
<soap:operation soapAction="**http://tempuri.org/GetVehicleDetail**" style="**document**" />
- <input>
<soap:body use="**literal**" />
</input>
- <output>
<soap:body use="**literal**" />
</output>

</operation>
- <operation name="**GetVehicleTravels**">
<soap:operation soapAction="**http://tempuri.org/GetVehicleTravels**" style="**document**" />
- <input>
<soap:body use="**literal**" />
</input>
- <output>
<soap:body use="**literal**" />
</output>
</operation>
- <operation name="**GetCustomerReserves**">
<soap:operation soapAction="**http://tempuri.org/GetCustomerReserves**" style="**document**" />
- <input>
<soap:body use="**literal**" />
</input>
- <output>
<soap:body use="**literal**" />
</output>
</operation>
- <operation name="**Login**">
<soap:operation soapAction="**http://tempuri.org/Login**" style="**document**" />
- <input>
<soap:body use="**literal**" />
</input>
- <output>
<soap:body use="**literal**" />
</output>
</operation>
</binding>
- <binding name="**CompanyWebServiceHttpGet**" type="**s0:CompanyWebServiceHttpGet**">
<http:binding verb="**GET**" />
- <operation name="**AddTravel**">
<http:operation location="**/AddTravel**" />
- <input>
<http:urlEncoded />
</input>
- <output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
- <operation name="**GetTravel**">
<http:operation location="**/GetTravel**" />
- <input>
<http:urlEncoded />
</input>
- <output>
<mime:mimeXml part="**Body**" />
</output>
</operation>

<operation name="**Order**">
<http:operation location="**/Order**" />
<input>
<http:urlEncoded />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**Cancel_Order**">
<http:operation location="**/Cancel_Order**" />
<input>
<http:urlEncoded />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**GetFullSeats**">
<http:operation location="**/GetFullSeats**" />
<input>
<http:urlEncoded />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**GetTravelDetail**">
<http:operation location="**/GetTravelDetail**" />
<input>
<http:urlEncoded />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**GetVehicleDetails**">
<http:operation location="**/GetVehicleDetails**" />
<input>
<http:urlEncoded />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**GetVehicleDetail**">
<http:operation location="**/GetVehicleDetail**" />
<input>
<http:urlEncoded />

</input>
- <output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
- <operation name="**GetVehicleTravels**">
<http:operation location="**/GetVehicleTravels**" />
- <input>
<http:urlEncoded />
</input>
- <output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
- <operation name="**GetCustomerReserves**">
<http:operation location="**/GetCustomerReserves**" />
- <input>
<http:urlEncoded />
</input>
- <output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
- <operation name="**Login**">
<http:operation location="**/Login**" />
- <input>
<http:urlEncoded />
</input>
- <output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
</binding>
- <binding name="**CompanyWebServiceHttpPost**" type="**s0:CompanyWebServiceHttpPost**">
<http:binding verb="**POST**" />
- <operation name="**AddTravel**">
<http:operation location="**/AddTravel**" />
- <input>
<mime:content type="**application/x-www-form-urlencoded**" />
</input>
- <output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
- <operation name="**GetTravel**">
<http:operation location="**/GetTravel**" />
- <input>
<mime:content type="**application/x-www-form-urlencoded**" />
</input>

<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**Order**">
<http:operation location="**/Order**" />
<input>
<mime:content type="**application/x-www-form-urlencoded**" />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**Cancel_Order**">
<http:operation location="**/Cancel_Order**" />
<input>
<mime:content type="**application/x-www-form-urlencoded**" />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**GetFullSeats**">
<http:operation location="**/GetFullSeats**" />
<input>
<mime:content type="**application/x-www-form-urlencoded**" />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**GetTravelDetail**">
<http:operation location="**/GetTravelDetail**" />
<input>
<mime:content type="**application/x-www-form-urlencoded**" />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>
<operation name="**GetVehicleDetails**">
<http:operation location="**/GetVehicleDetails**" />
<input>
<mime:content type="**application/x-www-form-urlencoded**" />
</input>
<output>
<mime:mimeXml part="**Body**" />
</output>
</operation>

```
- <operation name="GetVehicleDetail">
  <http:operation location="/GetVehicleDetail" />
- <input>
  <mime:content type="application/x-www-form-urlencoded" />
    </input>
- <output>
  <mime:mimeXml part="Body" />
    </output>
    </operation>
- <operation name="GetVehicleTravels">
  <http:operation location="/GetVehicleTravels" />
- <input>
  <mime:content type="application/x-www-form-urlencoded" />
    </input>
- <output>
  <mime:mimeXml part="Body" />
    </output>
    </operation>
- <operation name="GetCustomerReserves">
  <http:operation location="/GetCustomerReserves" />
- <input>
  <mime:content type="application/x-www-form-urlencoded" />
    </input>
- <output>
  <mime:mimeXml part="Body" />
    </output>
    </operation>
- <operation name="Login">
  <http:operation location="/Login" />
- <input>
  <mime:content type="application/x-www-form-urlencoded" />
    </input>
- <output>
  <mime:mimeXml part="Body" />
    </output>
    </operation>
    </binding>
- <service name="CompanyWebService">
- <port name="CompanyWebServiceSoap" binding="s0:CompanyWebServiceSoap">
  <soap:address location="http://localhost/ankara_web/CompanyWebService.asmx" />
    </port>
- <port name="CompanyWebServiceHttpGet" binding="s0:CompanyWebServiceHttpGet">
  <http:address location="http://localhost/ankara_web/CompanyWebService.asmx" />
    </port>
- <port name="CompanyWebServiceHttpPost" binding="s0:CompanyWebServiceHttpPost">
  <http:address location="http://localhost/ankara_web/CompanyWebService.asmx" />
    </port>
    </service>
    </definitions>
```

Ek. 2. Sistem Performansı Ölçüm Sonuçları

Sunucu Sayisi - Zaman (t)				**Parametre Sayisi - Zaman (t)**			
Deney 1				**Deney 1**			
	Sunucu Sayısı	**Süre**			**P. Sayısı**	**Süre**	
	1	0,092934437198024	**sn**		**10**	0,133530099495886	**sn**
	2	0,105815683278182	**sn**		**20**	0,197379555222801	**sn**
	3	0,387819122262746	**sn**		**30**	0,382141026303622	**sn**
	4	1,956551181784280	**sn**		**40**	0,428155813099151	**sn**
Deney 2				**Deney 2**			
	Sunucu Sayısı	**Süre**			**P. Sayısı**	**Süre**	
	1	0,035521833082138	**sn**		**10**	0,027504054286229	**sn**
	2	0,049065733214696	**sn**		**20**	0,112534134924970	**sn**
	3	0,071935120245730	**sn**		**30**	0,182981635934176	**sn**
	4	0,079770194256533	**sn**		**40**	0,394585345344171	**sn**
Deney 3				**Deney 3**			
	Sunucu Sayısı	**Süre**			**P. Sayısı**	**Süre**	
	1	0,010472001329778	**sn**		**10**	0,046226266187462	**sn**
	2	0,056532604004140	**sn**		**20**	0,150297314323468	**sn**
	3	0,083338524868384	**sn**		**30**	0,214921449513835	**sn**
	4	0,100112165093608	**sn**		**40**	0,591231287775402	**sn**
Deney 4				**Deney 4**			
	Sunucu Sayısı	**Süre**			**P. Sayısı**	**Süre**	
	1	0,051655168464148	**sn**		**10**	0,010225601298489	**sn**
	2	0,010470325139089	**sn**		**20**	0,092244405364369	**sn**
	3	0,108970274154955	**sn**		**30**	0,191175973482663	**sn**
	4	1,743132158580500	**sn**		**40**	0,411715176090816	**sn**
Deney 5				**Deney 5**			
	Sunucu Sayısı	**Süre**			**P. Sayısı**	**Süre**	
	1	0,121781120226174	**sn**		**10**	0,022509564763119	**sn**
	2	0,129084003693207	**sn**		**20**	0,135190645738495	**sn**
	3	0,151560324007660	**sn**		**30**	0,236515814160738	**sn**
	4	2,422924701323770	**sn**		**40**	0,341604589410107	**sn**