İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

WEB SERVİS MİMARİSİNİN BİR UYGULAMASI

YÜKSEK LİSANS TEZİ

Mat. Müh. İbrahim Şahin KEKEVİ

Anabilim Dalı: MÜHENDİSLİK BİLİMLERİ
Program: SİSTEM ANALİZİ

HAZİRAN 2002

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

WEB SERVİS MİMARİSİNİN BİR UYGULAMASI

YÜKSEK LİSANS TEZİ

Mat. Müh. İbrahim Şahin KEKEVİ

(509971408)

Tezin Enstitüye Verildiği Tarih: 10 Mayıs 2002

Tezin Savunulduğu Tarih: 31 Mayıs 2002

Tez Danışmanı: Prof. Dr. Gazanfer ÜNAL

Diğer Jüri Üyeleri Prof. Dr. Galip G. TEPEHAN

Doç. Dr. Metin Orhan KAYA

HAZİRAN 2002

ÖNSÖZ

Hazırlamış olduğum bu tezi bu günlere gelmemde büyük emeği olan, tezimin bitmesini sabır ve merakla bekleyen fakat tamamlanmasını göremeyen babam Mustafa Kemal Kekevi'ye itaf etmek istiyorum.

Tezimin hazırlanması sırasında değerli eleştiri ve önerileriyle oluşumuna katkı sağlayan ve gösterdiği hoşgörü ve destekle güzel bir çalışma ortaya çıkmasına imkan tanıyan değerli hocam sayın Prof. Dr. Gazanfer ÜNAL'a teşekkürlerimi bir borç bilirim. Ayrıca her zaman olduğu gibi birikiminlerinden faydalanmamı esirgemeyen ve tezimin şekillenmesi ve oluşumuna büyük katkıları olan sevgili biraderim Serkan Taş'a, sevgili dostum Uğur Tanrıverdi'yede hiç bir zaman esirgemediği desteği için ve benimher zaman yanımda olan sevgili aileme başta annem Özen, ablam Ayşegül ve kardeşim Gülbin Kekevi'ye ayrı ayrı teşekkürlerimi borç bilirim. Son olarakta sevgili Nilay'a teşekkür etmek istiyorum.

Haziran 2002 **Mat. Müh. İbrahim Şahin Kekevi**

İÇİNDEKİLER

KISALTMALAR

SOAP	: Simple Object Access Protocol
WSDL	: Web Servis Description Language
UDDI	: Universal Description Discovery and Integration
HTTP	: Hyper Text Transfer Protocol
JSP	: Java Server Pages
XML	: Extensible Markup Language
FTP	: File Transfer Protocol
IDL	: Interface Definition Language
HTML	: Hyper Text Markup Language
XSL	: Extensible Stylesheet Language
URL	: Uniform Resource Locator
API	: Application Programming Interface
RPC	: Remote Procedure Call
JDBC	: Java Database Connectivity
ODBC	: Open Database Connectivity
ISP	: Internet Service Provider
JDK	: Java Development Kit
SQL	: Structured Query Language
NAISC	: North American Industrial Classification System
UNSPSC	: Universal Standart Products and Services Classification
GCS	: Geographic Classification System

ŞEKİL LİSTESİ

WEB SERVİS MİMARİSİNİN BİR UYGULAMASI

ÖZET

Bu çalışmanın amacı Web Servis mimarisinin detaylarıyla incelenmesi ve incelenen bu yapı üzerine bir uygulamanın geliştirilmesi ve değerlendirilmesidir. Konu olarak Web Servisleri modelini seçmemin nedeni ise önümüzdeki yıllarda yazılım uygulamalarının %95'inin bu model üzerinde çalışacağının SUN, MS, IBM gibi sektöre yön veren firmalar tarafından öngörülmesidir.

Çalışma üç bölümden oluşmaktadır. Birinci bölümde Web servis mimarisi ve onu oluşturan yapılar, protokoller detaylarıyla incelenmekte genel bir yaklaşımla ele alınmaktadır. Mimarinin gelecekteki uygulama alanlarından ve günümüz yazılım uygulamalarında kullanılmasından entegrasyonuna kadar tüm konu genel bir yaklaşım içinde ele alınmaktadır. İlk bölümde Web servis mimarisine ilaveten projede kullanılan teknolojiler kısa bir özet halinde aktarılmaktadır.

İkinci bölüm ise Web servis mimarisi üzerine geliştirmiş olduğum uygulamadan oluşmaktadır. Geliştirmiş olduğum bu uygulama ise kısaca web servis mimarisini kullanan bir B2B uygulamasıdır. Uygulama hava durumu ve otomobil uygulaması olmak üzere iki ana bölümden oluşmaktadır. Her iki uygulamada Java merkezli bir yaklaşımla geliştirilmiştir. Otomobil uygulamasında sağlanan veriler veritabanında tutulan statik verilerdir ve kullanıcıya web servis mimarisi kullanılarak ulaştırılmaktadır (hava durumu uygulamasında da olduğu gibi). Hava durumu uygulaması ise kullanıcıya hem istatiksel veri (statik veri) hem de anlık dinamik veri sağlamaktadır. Hava durumunun statik verileri otomobil uygulamasında olduğu gibi veri tabanında tutulmakta dinamik veri ise internetten anlık olarak çekilebilmektedir. Program internetten çekilen anlık veriyi istenirse veritabanına kaydedebilmektedir. Kullanıcı kaydedilen bu verilerede web servis mimarisi üzerinden ulaşabilmektedir.

Programlama dili olarak Java ve J2EE teknolojilerini seçmemin nedeni ise Java'nın internet uygulamalarının geliştirilmesinde sağladığı kolaylıklar ve web servis mimarisinin gelişiminde oynadığı rol oldu. Hazırlamış olduğum çalışmada Web Servis mimarisinin ve kullanılan teknolojilerin tanıtımından sonra detaylarıyla verdiğim uygulama 3 katmanlı bir yapıdan oluşmaktadır. Birinci katmanda kullanıcı ayağı yer almakta, ikinci katmanda sistemin aracını ve omurgasını oluşturan web servis yapısı bulunmakta ve son olarakta üçüncü katmanda sistemin genel olarak veritabanı ayağını oluşturmaktadır. Veritabanı olarak SQL Server 7.0'ı seçmemin sebebi ise büyük çapta veri tabanı uygulamalarının geliştirilmesine imkan tanıması ve kolay bir kullanıma sahip arayüze sahip olmasıdır.

Üçüncü bölümde ise çalışmanın genel bir değerlendirilmesi yapılmakta uygulamanın kullanılabileceği örneklerden ve geliştirilebilirliğinden bahsedilmektedir. Böylelikle ilerde bu çalışmadan yararlanacaklar için bazı referanslar verilmektedir.

AN IMPLEMANTATION OF WEB SERVICE ARCHITECTURE

SUMMARY

The aim of this study is to examine Web Service architecture in detail and to build up and evaluate an application on this structure. According to the forecasts of the leading companies of the software sector such as SUN, MS and IBM, in the next few years 95% of software implementations will work on this architecture. This is the main motivation for choosing web service model as my thesis subject.

The study consists of three sections. In the first section Web service architecture and its protocols are examined in detail. From field of applications of the architecture in the future to its integration with current software implementations, the whole subject is handled with a general approach in this section. In addition to Web service architecture, the technologies used in the project are described briefly in this section.

The second section consists of the implementation developed on Web service architecture. This is an B2B application which uses web service architecture. The application has two main parts: weather service application and automobile application. Both of these two applications are developed with a Java based approach. The data used in automobile application is of a static nature and it is stored in a database. This data is transfered to the end user via web service architecture. (as in the weather service application) On the other hand weather service application provides both statistical (static) and dynamic data to the user. Static data of this application is stored in a database as in the automobile application whereas the dynamic data can be transfered instantaneously via internet. Upon request of the user, the program can record the data transfered to the database and the user can reach the data recorded by using web service architecture.

The reason for choosing J2EE technologies and Java as the programming language, is the ease of developing internet implementations with this language and the role that it played in the development of web service architecture. After the introduction of web service architecture and the technologies used, an implemantation with three layers is presented in detail in this study. The first layer is the client side, the second layer consists of web service construction that forms the goal and the backbone of the system and the last layer is the server side that forms the database of the system. SQL Server 7.0 is chosen as the database in this study as it enables developing database applications with large amounts of data and as it has a user friendly interface.

In the last section, some references are given for those who will benefit from this study in the future by providing examples which can be used by the implementation, considering chances of improvement and evaluating the study generally.

1. GİRİŞ

Web servisleri geleceğin uygulama entegrasyonları ve dağınık mimarileri için büyük ümitler vaat eden ve hızla gelişen bir dizi standart ve uygulama teknolojileridir.

Tezimde web servis konseptini tanıtarak, onunla ilişkili çeşitli standartları Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) ve Universal Derscription Discovery and Integration (UDDI) gibi web servis mimarisi içinde genel bir perspektifle ele almaya çalışacağım. Web servislerini genel bir bakışla teknik anlamda inceledikten sonra bir web servis uygulaması geliştirerek kullanılabileceği geniş alanlara bir örnek vericeğiz. Bölüm 1.1'de yer alan ve Web'in gelecek evrimini gösteren model yeni nesil akıllı sistemlere örnek teşkil ederken tezin ana modülü olan hava durumu uygulamasının bu tür sistemlere entegrasyonunu örneklendirmektedir.

Bölüm 2'de ise web servisleri geliştirilirken ihtiyaç duyulacak kavramların ve teknolojilerin anlatımı yapılmaktadır. Bu anlatım 3. Bölümde verilen ve Hava Durumu uygulamasına göre daha basit fakat yine temel yapı üzerinde geliştirilen Oto Guide uygulaması üzerinde örneklendirilerek detaylandırılmaktadır. Konu çok geniş bir alanı ve teknolojiyi kapsadığından sadece uygulamada ihtiyaç duyulan kısımlar ve teknolojiler anlatılmaktadır. Konunun rahat anlaşılabilmesi için web-tabanlı uygulamala mimarisine ve tekniklerine (HTTP gibi) biraz yakınlık gerekmektedir. Kodların anlaşılması için kullandığım Java ve ilişkili Java Server Pages (JSP) ve Java servlet teknolojilerine de aşina olmak kolaylık sağlayacaktır. Web servis yapısının çekirdeğini oluşturan Extensible Markup Language (XML) diline de kısaca değinilecektir.

3. Bölüm'de ise uygulama detaylandırılmakta ve önceki bölümlerin eşliğinde web'in gelecek evrimi ve bu evrimin temel taşı olan web servislerine ışık tutulmaktadır.

Tezde web servis mimarisi açıklanırken ve uygulama geliştirilirken Java merkezli bir yaklaşım seçilmiştir fakat şunu da belirtmeliyim ki web servis mimarisi tamamen

programlama dilinden bağımsız bir yapıdır ve seçilen herhangi bir dile uygulanabilir.

1.1 Neden Web Servisleri ve İnternet?

Bilgisayar kullanımında ki hızlı gelişimi hepimiz gördük ve yaşadık. Delikli kartlardan gelişmiş obje tabanlı programlarına hızlı bir gelişme ve geçiş yaşadık. Tüm bu gelişim programlamaya büyük bir üretkenlirlik kazandırdı.

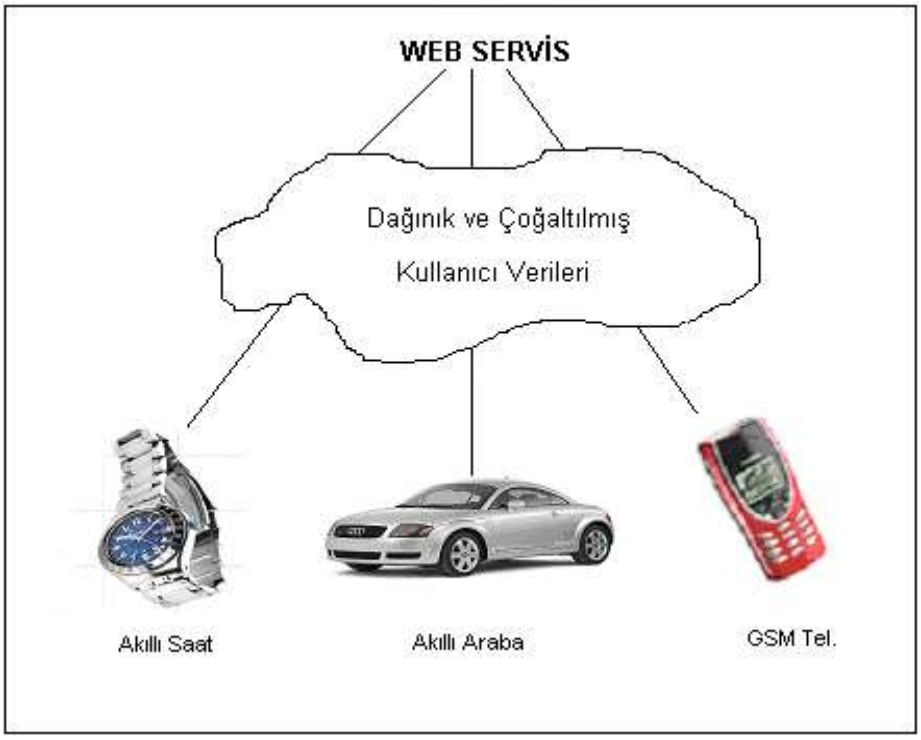
Değişik seviyelerde ki bilgisayar kullanıcılarının deneyimlerinde de büyük bir gelişim yaşandı. İnsanlar PC'lerini kullanarak İnternet'e girebiliyor, e-mail yollayıp alabiliyor, online bankacılıktan ve benzeri bir dizi teknolojiden artık kolaylıkla yararlanabiliyor. Yeni internet protokollerinin 64-bit adreslemeye sahip aletlerle uyumu sonucu tüm bu elektronik aletlerin internet üzerinde tek bir IP'leri olabiliyor dolayısıyla tek bir kimliğe sahip olabiliyorlar. İnternete bağlı bu cihazlara web servisleri yazmak için gerekli altyapı ise sektöre öncülük eden firmalar tarafından sağlanıyor.

Gelecek nesil cihazlar ise açık bir şekilde akıllı hücresel telefonlar, internet ve TV'nin integrasyonu ve daha güçlü kişisel digital asistanlar (PDA). Belki daha az açık bir şekilde de bu yeni cihazlar örneğin takılarımızda veya elbiselerimizde yer alacak. Klasik olarak hep local networkleri ve interneti düşünüyoruz. Bu klasik network görüşü vücutta taşınan cihazların ve gün geçtikçe bir yenisi çıkan cihazların eklenmesiyle daha da büyüyecek. Dolayısıyla şimdiden geleceğin kullanıcı ortamları hedefler doğrultusunda gelecek nesil web servis dizaynları yapılmaya başlandı. Aşağıda sayıcağımız aletlerle günlük yaşantısında kontak kuran bir kullanıcı düşünelim.

- Akıllı bir saat böylece kullanıcıyı tek bir IDile tanımlayabiliriz. Kullanıcı benzin alabilir, alışverişinde kullanabilir (basitçe kredi kartının gelişmişi) ve evinin veya arabasının kapısını kilitleyip açabilir.

- İnternet bağlantılı bir GSM telefonu (aynı zamanda akıllı saatle bağlantılı).

- Akıllı bir araba.



Şekil 1.1 Web'in Gelecek Evrimi

Listeyi akıllı elektrikli ev aletleri, akıllı televizyon üstü gibi cihazlarla genişletebiliriz fakat yukarıda verdiğimiz işlem yeteneğine sahip bu dört cihaz kullanıcı merkezli uygulamalar geliştirmek için gereksinimleri anlamamız için yeterli. Kullanıcının kim olduğunun ve ne yaptığının bilinmesi önemli, bu bilgiye ulaşabilmek için kullanıcının etkileşimde olduğu cihaz kullanılabilir. Şekil 1.1'de akıllı cihazların etkileşimiyle örnek bir senaryoda kullanıcı arabasıyla uzun bir seyahatteyken bulunduğu bölgenin hava durumu ve yol verileri kullanıcıya GSM telefonu ile e-mail olarak oradan da kullanıcının daha rahat görebileceği arabanın gösterge ekranına aktarılabilir. Tezde geliştirilen hava durumu uygulaması ise bu senaryoda yer alan hava durumu verileri için %100 uyumlu bir web servisi sağlamaktadır.

Başka bir senoryaya bakıcak olursak kullanıcı saatini benzin almak için kullanabilir (ödemeyi pompada yapabilir) yapılan işlemler saatten kullanıcının GSM'ine aktarılır. Kullanıcı eve geldiğinde ise GSM verileri PC'ye aktarır (zaten alışverişten hemen sonra internet üzerinden GSM'den PC'ye aktarılmıştır eve gelince işlemlerin bir kontrol ve güncellemesi yapılır). Sağlanan bu web servis sayesinde kullanıcı aylık harcamalarını ve bütçesini kolaylıkla kontrol edebilecektir.

Bu senaryolar kesinlikle bugün sahip olduğumuzdan çok farklı. Web servisleri ise bu noktada devreye girerek tüm bu akıllı ve dağınık sistemlerin haberleşmesi için bir model oluşturmaktadır. Geliştirmiş olduğum ve Bölüm 3'dede detaylarıyla verilen özellikle hava durumu uygulaması böyle senaryolar için web servis hizmeti sunabilmektedir.

2. WEB SERVİSLERİ

Web'in program-kullanıcı için sağladığı etkileşimi, web servisleri programdan programa sağlamıştır. Web servisleri birbirleriyle network üzerinde haberleşmeye çalışan tek bir dağınık bilgisayar mimarisi oluştururlar. Yeni ufkun anahtarı gelişmekte olan Extensible Markup Language (XML) [1], HTTP, Simple Object Access Protocol (SOAP) [2], Web Service Description Language (WSDL) [3] and Universal Description Discovery and Integration (UDDI) [4] standartları üzerine kurulmuş programdan-programa haberleşme modelidir.

Bölüm 1.1'de belirtildiği gibi tezin yazımı sırasında Java [5] merkezli bir yaklaşım izlenmiştir. Fakat bu bölümde verilenler web servis mimarisinin [2] genel yapısını sunmakta ve 3. Bölüm'de detaylandırılan uygulamalar bu mimari üzerine kurulduğundan geliştirilecek başka uygulamalara da kılavuzluk edebileceklerdir. Bu yüzden verilen bu mimari herhangi bir programlama diliyle farklı teknolojiler ve platformlar kullanılarak uygulamalar geliştirilebilir. Zaten web servis mimarisinin geliştirilmesine sebep olan ana ihtiyaç farklı platformlar üzerinde çalışan farklı programlama dilleriyle yazılmış objelerin, uygulamaların birbirleriyle haberleşmesini sağlamaktır. İşte bu noktada web servis mimarisi bir bakıma tercümanlık görevi yapmaktadır. Bu tercüman SOAP protokolüdür dersek doğru olur. SOAP'la beraber WSDL ve UDDI bu mimariyi oluştururlar. Bu üçlüyü oluşturan ve haberleşmeyi sağlayan dilde XML'dir. WSDL ve UDDI'ın yapısının ve SOAP haberleşmesinin anlaşılması için XML bir şarttır. Bölüm 3'de anlatılan uygulamaların kalbinide SOAP server teşkil etmekte ve modüller arası haberleşme XML ile sağlanmaktadır.

Tüm bu nedenlerden dolayı bu bölümde verilen teknolojilerin ve özellikle web servisi oluşturan protokollerin iyi anlaşılmasında fayda vardır çünkü web sevis mimarisi platformdan bağımsızdır. Ayrıca SOAP server başta olmak üzere diğer protokol ve teknolojiler de üçüncü partiler tarafından sağlanmakta ve değişik platformlar için sayıları hergün artmaktadır [6]. Bu yüzden elemanların iyi

anlaşılması zaten uygulamanın büyük kısmını aydınlatacak 3. Bölüm'dede detaylarıyla vereceğimiz uygulama resmin eksik parçalarını tamamlayacaktır.

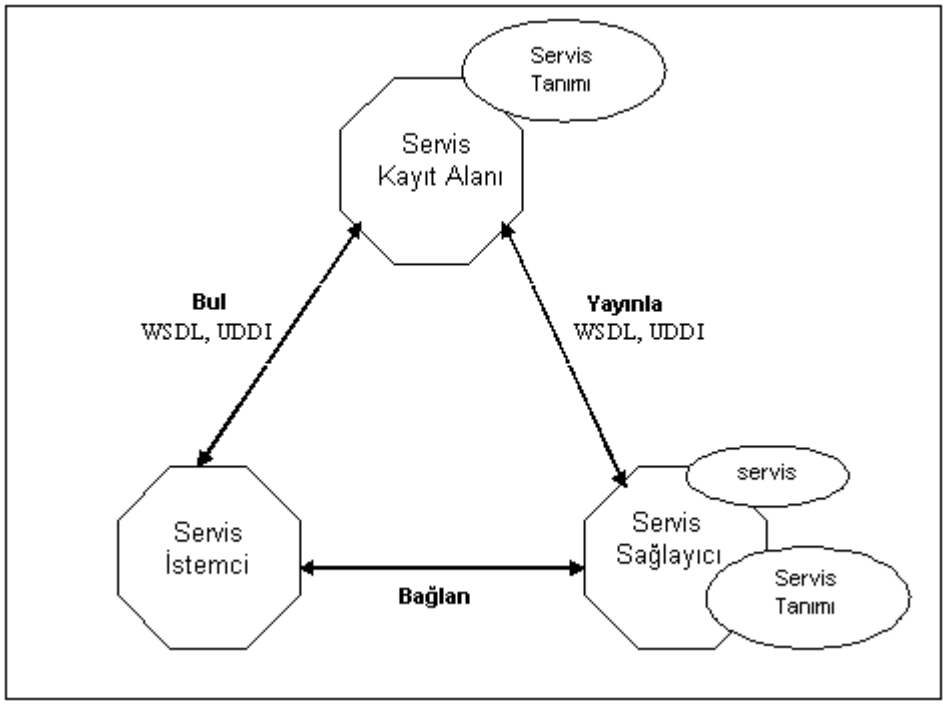
2.1 Web Servislerinin Tanımı

Web Servisleri standartlaştırılmış XML mesajlaşma ile network ulaşımına açık operasyonlar kolleksiyonunu tanımlayan bir arayüzdür [2]. Web servisi standart bir XML notasyonu tarafından tanımlanır ve onun servis tanımı diye adlandırılır [3]. Bu servisle etkileşmek için gerekli mesaj formatları, iletişim protokolleri ve lokasyon dahil olmak üzere tüm detayları kapsar. Arayüz servisin uygulandığı donanım ve yazılım platformu üzerinde vede yazıldığı programlama dilinde bağımsızca kullanılmasına izin vererek uygulama detaylarını saklar. Web servisleri tek başlarına veya diğer web servisleri ile kompleks çözümler için veya business işlemleri için kullanılabilir. Programcılar servisleri oluşturan standartları kullanarak dağınık uygulamalar geliştirebilirler.

2.2 Web Servis Modeli

Web servislerinin mimarisi üç rolün etkileşimi üzerine kurulmuştur; servis sağlayıcı, servis kayıdı ve servis istemcisi [2]. Etkileşimler yayınlama, bulma ve bağlama operasyonlarını kapsar. Birlikte bu roller ve operasyonlar web servis objeleri, web servis software modülleri ve tanımları üzerinde rol oynarlar. Tipik bir senaryoda servis sağlayıcı network ulaşımına açık bir software modülüne ev sahipliği eder. Servis sağlayıcı servis tanımını belli bir web servisi için tanımlar ve bunu bir servis kayıt alanına veya servis istemcisine yayınlar. Servis istemci bir arama operasyonuyla servis tanımını lokal olarak veya servis kayıt alanından çeker ve servis tanımını servis sağlayıcıya bağlanmak ve web servis uygulamasını çalıştırıp veya etkileşmek için kullanır. Servis sağlayıcı ve servis istemci rolleri mantıksal yapılardır ve bir servis ilişkisinin karakteristiklerinide miras edinebilir.

Şekil 2.1 bu operasyonları canlandırıp onları sağlayan komponentleri ve onların etkileşimini göstermektedir.



Şekil 2.1 Web Servis Mimarisi

2.3 Web Servis Mimarisi

Şekil 2.1'i geliştirdiğimiz uygulamalar boyutundan incelersek servis sağlayıcı hava durumu ve araba modelleri hakkında veriler sağlayan Java programlarıdır. Servis istemci web browser kullanan bir kullanıcıdır. Servis kayıt alanı ise IBM Test Registry'e (https:// www-3.ibm.com/services/uddi/testregistry/protect/) Hava Durumu adı altında kaydedilmiş fakat uygulama local olduğu için internet üzerinden erişim olanaklı değildir [4]. Teknik anlamda ise;

Servis Sağlayıcı: İş perspektifinden servisin sahibi.

Servis İstemci: İş perspektifinden, tatmin edilmesi gereken belli fonksiyonlara ihtiyaç duyan işletme. Mimari perspektifden servisi arayan, çalıştıran ve bir etkileşim başlatan uygulama. Servis istemci browser kullanan biri olabileceği gibi kullanıcı arayüzü olmayan bir programda olabilir, örneğin başka bir web servisi.

Servis Kayıt Alanı: Servis sağlayıcıların servis tanımlarını yayınladığı, servis tanımlamalarının aranabildiği bir kayıt alanı. Başka bir açıdan servislerin statik ve dinamik bağlanma için servis istemcilerinin servis ve bağlanma bilgilerine ulaşabildiği bir alandır. Statik bağlanma servis istemlerinde, servis kayıt mimarisinde opsiyonel bir rol oynar çünkü servis tanımı istemciye servis sağlayıcı tarafından

direkt olarak sağlanabilir. Servis istemcisi servis tanımını servis kayıt alanı haricinde local dosya olarak, FTP sitesinden, Web sitesinden, bir reklamdan da elde edebilir.

2.4 Web Servisi Mimarisinde Roller

Bir uygulamanın web servisinin avantajlarını taşıyabilmesi için üç davranışı göstermesi gerekir; servis tanımının yayınlanması, servis tanımının aranabilmesi ve servis tanımı ile servise bağlanılabilmesi ve çalıştırılabilmesi [2]. Bunlar teker teker veya ardışık olarak gerçekleşir. Detayları ile bu operasyonlar:

Yayınla: Servis tanımı yayınlanmalıdır böylece servis istemcisi onu bulabilir. Nerede yayınlanacağı ise uygulamanın gereksinimlerine göre değişebilir.

Bölüm 2.3'de uygulamanın lokal olmasına karşın örnek olarak uygulamayı IBM Test Registry alanına yayınladığımızı söylemiştik.

Bul: Find operasyonunda, servis istemcisi servis tanımını direkt olarak çeker ve yahut servis kayıt alanını aradığı tipteki servise göre sorgulayarak ulaşır. Find operasyonu servis istemcisi için iki şekilde oluşabilir, dizayn sırasında servis arayüz tanımını programın yazılımı için kullanabilir veya runtime sırasında servise bağlanma ve lokasyon tanımlarını çekebilir.

Herhangi bir kullanıcı bu alanı sorguladığı zaman (örneğin Hava Durumu diye) ilgili servis linkine ulaşıcaktır.

Bağlan: Sonuçta bir servise ulaşılması gerekir. Bind operasyonunda servis istemcisi servis tanımındaki detayları servisle runtime esnasında etkileşmek için kullanır.

2.5 Web Servis Nesneleri

Servis: Servis tanımı tarafından belirtilen web servis arayüzünün uygulamasıdır. Servis network ulaşımına açık platformüzerinde servis sağlayıcı tarafından sağlanan bir yazılım modülüdür. Servis istemcisi tarafından çalıştırılmak vaya etkileşmek için tasarlanmıştır. Ayrıca istemci fonksiyon olarak diğer web servis uygulamalarını kullanabilir.

Servis Tanımı: Servis tanımı servisin arayüz ve uygulama detaylarını kapsar. Servisin veri tiplerini, operasyonlarını, bağlanma bilgilerini ve network adresini.

Ayrıca servis istemcisi tarafından keşfedilebilmesi için gerekli kategorizasyon ve metadolojiyide kapsayabilir. Servis tanımının bir servis istemcisine veya servis kayıt alanına yayınlanması gerekir.

2.6 Web Servislerinin Geliştirilme Döngüsü

Web servisi geliştirme döngüsü her rol için dizayn, geliştirme ve runtime gereksinmelerini kapsar: servis kayıt alanı, servis sağlayıcı ve servis istemcisi. Her rol geliştirme döngüsü için spesifik gereksinimlere sahiptir. Geliştirme döngüsü dört safhadan oluşur:

İnşa: Geliştirme döngüsünün inşa safhası web servisi uygulamasının geliştirme ve testini, servis arayüz tanımının ve servis uygulama tanımının tanımlanmasını kapsar. Web servisi uygulamaları yeni web servisleri oluşturarak, varolan uygulamaları web servislere dönüştürerek veya web servisleri diğer web servislerle ve uygulamalarla birleştirerek oluşturulabilir.

Deploy: Deploy safası servis arayüzünün ve uygulamasının tanımının servis istemcisine veya servis kayıt alanına tanıtılmasıyla ve web servisinin çalıştırılabileceği bir icra ortamına yerleştirilmesiyle sağlanır. Bu bir web server olabilir.

Run: Run safasında artık web servisi icra edilmeye hazırdır. Bu safada artık servis sağlayıcı tarafından tamamen deploy edilmiş, operasyonel ve network ulaşımına açıktır. Artık servis istemcisi find ve bind operasyonlarını uygulayabilir.

Manage: Yönetimsafası web servisi uygulamasının süregiden yönetimve denetimini kapsar. Güvenlik, geçerlilik, performans ve iş processleri belirlenmelidir.

2.7 Web Servis Standartları

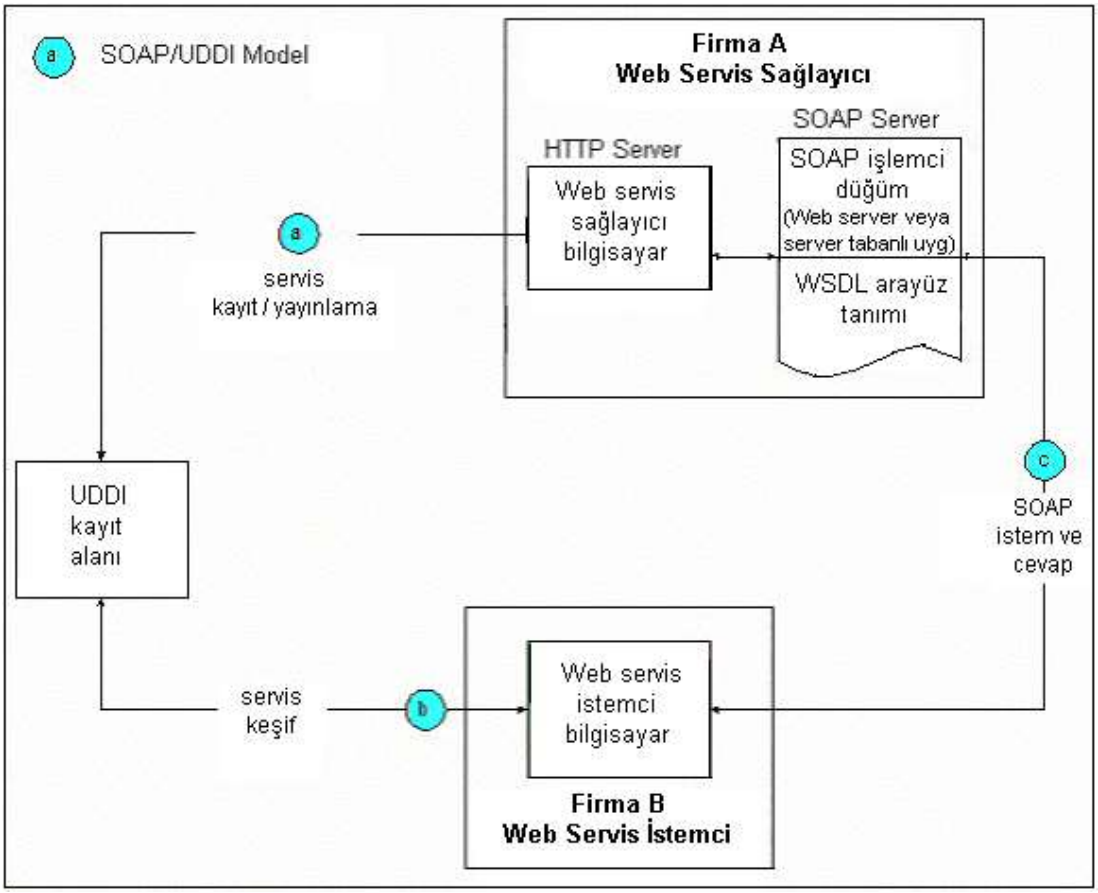
UDDI : Geçerli web servis komponentlerini tanımlayan bir protokoldür [1] . Bu standart işletmelere servislerini bir internet dizinine kayıt edebilme ve reklamını yapabilme olanağı verir böylece firmalar birbirlerini bulup işlerini web üzerinden yapabilirler. Kayıt ve arama işlemleri XML ve HTTP temelli mekanizmalarla yapılır.

SOAP : SOAP farklı server'larda bulunan metodların veya fonksiyonların farklı

uygulamalar tarafından kullanılmasını sağlar [1]. SOAP uygulaması farklı serverlarda bulunan methodların çağrılması için gerekli veriyi ve uygulamanın kendi için gerekli lokasyon gibi datayı XML bloğu şeklinde oluşturur.

WSDL : Web servis tanımı için önerilen standart XML tabanlı bir IDL (Interface Definition Language) servisidir [1]. WSDL servisin arayüzünü ve uygulama karakteristiklerini tanımlar. WSDL, UDDI girişleri tarafından referans alınır ve belirli bir web servisini tarif eden SOAP mesajlarını tanımlar.

Şekil 2.2'de bir web servisinin standartlarıyla şematik gösterimi verilmektedir.



Şekil 2.2 Bir Web Servisi

1. Servis sağlayıcı bir web servis oluşturur.
2. Servis sağlayıcı servisi tanımlamak için WSDL'i kullanır.
3. Servis sağlayıcı servisi bir UDDI'a kayıt eder.
4. Başka bir servis veya kullanıcı UDDI'yı sorgulayarak servisi belirler.
5. İstemci servis veya kullanıcı servise bağlanmak için gerekli uygulamayı SOAP üzerinden haberleşecek şekilde yazar.

6. Data ve mesajlar HTTP üzerinden XML olarak değiştirilir.

WSDL web servisin detaylarını tanımlayan bir XML dökümanıdır. Web servis standartlarına geçmeden XML üzerinde kısaca durmakta fayda var çünkü web servisleri oluşturan standart protokollerin meydana getirdikleri yapının ortak dilini XML oluşturmaktadır.

2.8 XML Nedir?

XML web üzerinde data değişimi için hızla standart haline gelen text temelli bir işaretleme dilidir[1]. HTML olduğu gibi veriyi etiketlerle tanımlarız[8]. HTML'den farklı olarak XML etiketleri verinin nasıl tanımlanacağını gösterir, datanın nasıl gösterileceğini değil. Mesela (.....) HTML etiketinde bu verinin bold gösterilmesini tanımlarken, XML etiketleri programınızdaki bir değişken gibi davranır.

Veri tabanında alan adlarını tanımladığınız gibi XML etiketlerini verilen bir uygulamada kullanabilirsiniz. Doğal olarak çok sayıda uygulamanın aynı XML verisini kullanabilmesi için etiket isimleri üzerinde anlaşmaları gerekir.

Etiket ve sonlandırıcı etiket arasındaki XML verisi bir elemanı tanımlar. Bir etiketin bir diğerini kapsama yeteneği XML'e hiyerarşik veri tabanlarını temsil edebilme yeteneği verir. HTML'den farklı olarak XML etiketleri verileri tanımladıkları için kapsadıkları veriler search edilebilir.

2.9 XML Uygulama İçin Neden Önemli?

Sırf Texttir: XML binary formatında olmadığı için standart text editöründen visual development ortamına kadar herhangi bir araçla oruşturulup, değiştirilebilir. Buda programın debug edilmesini ve küçük miktarda datanın depolanmasını elverişli kılar. Diğer taraftan bir veritabanı büyük miktarda XML verisini depolayabilir. Böylece XML küçük dosyalardan firma genişliğinde verilere kadar ölçeklenebilirlik sağlar.

Veri Tanımlaması: XML ne çeşit veriniz olduğunu anlatır, onu nasıl göstereceğinizi değil. İşaretleme etiketleri bilgiyi belirttiği ve datayı kısımlara ayırdığı için bir e-mail programı onu işleyebilir, bir arama programı belirli kişilere yollanacak mesajları arayabilir ve bir adres defteri adres bilgisini mesajın geri kalanından çekebilir.

Kısaca bilginin farklı bölümleri tanımlandığından bunlar farklı uygulamalarda farklı şekillerde kullanılabilir.

Sergilenebilirlik: Gösterim önemli olduğundan stylesheet standartı XSL verinin nasıl sergileneceğini dikte eder. Bunu HTML'dede yapabiliriz fakat veriyi search programları ile işleyemeyiz. Daha önemlisi XML özgün tarzda olduğundan tamamen farklı stylesheet kullanıp çıktıyı postscript, TEX, PDF veya henüz keşfedilmemiş bir formatta üretebiliriz.

Kolayca İşlenebilir: Düzenli ve kararlı notasyonu programların XML verisini işlemesini kolaylaştırır. HTML'de bir <dt> etiketi bir diğer </dt>, <dt>, <dd> veya </dl> ile son bulabilir. Bu programlama için zorluk yaratır. Fakat XML'de <dt> etiketi </dt> ile son bulmalıdır, bu kısıtlama XML dökümanına iyi oluşturulmuş bir yapı sağlar. Ayrıca XML parserlar arasından işlem yükünüzü azaltacak herhangi birini seçme olanağı vardır.

Hiyerarşik Yapı: Son olarak XML dokümanları hiyerarşik yapılarından da yararlanır. Hiyerarşik döküman yapıları genel olarak işlemek için hızlıdır çünkü ihtiyaç duyulan bölüme kolayca ulaşılabilir bir içindekiler bölümünü kullandığınızda olduğu gibi.

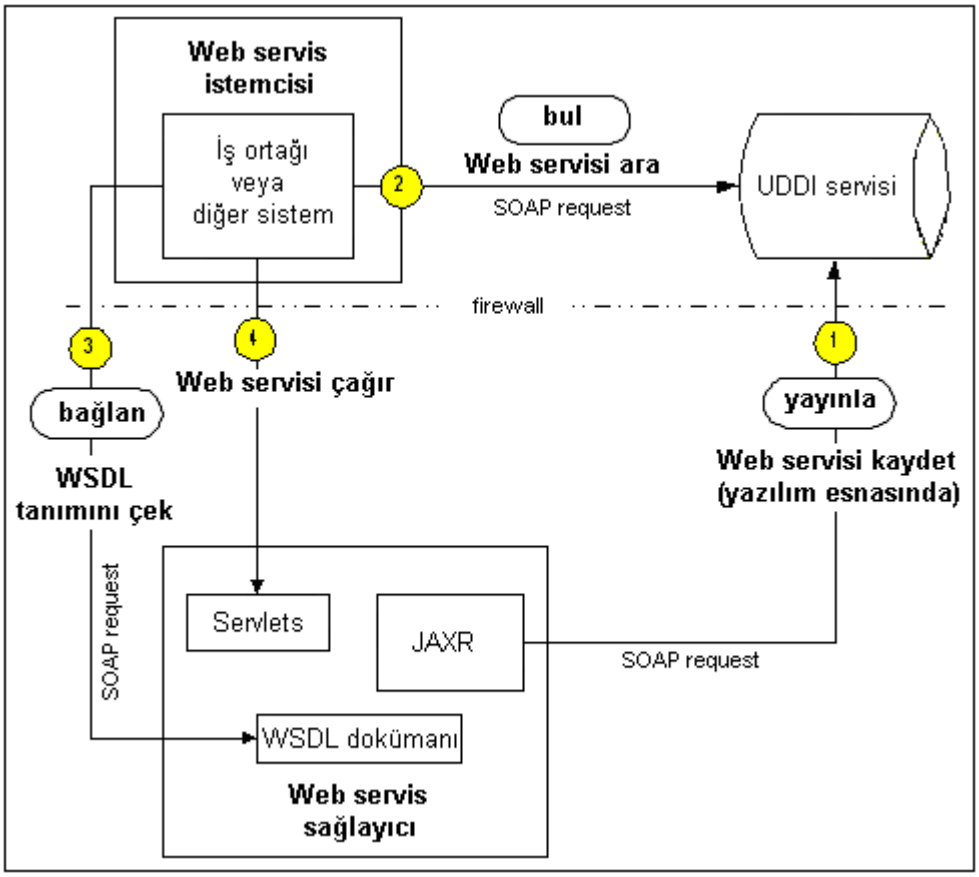
2.10 WSDL

Web service description language, web servislerinin teknik çağırım syntaxını tanımlamak üzere önerilmiş bir standarttır[3]. Bir WSDL servis tanımı, WSDL şema tanımına uygun bir XML dokümanıdır. Daha önce belirttiğimiz gibi WSDL dokümanı tam bir servis tanımı değildir fakat servis tanımının alt seviyelerini kapsar (servis arayüzünün teknik tanımlamasını). WSDL web servisleri için bir IDL'dir (Interface Definition Language). Aslında WSDL bir web servisinin 3 temel tanımını belirtir;

- Bir servis ne yaptığı, servisin sağladığı operasyonlar (metodlar).
- Bir servis nasıl çağrılır, servis operasyonlarını çağırmak için gerekli data formatları ve protokoller.
- Servisin nereye yerleştirildiği URL adresi gibi, protokole bağlı network adres detayları.

2.10.1 WSDL ve java teknolojisi

Şekil 2.3 bir web servisin Java teknolojisi ile nasıl kayıt edildiğini, bulunduğunu ve çağrıldığını göstermektedir;



Şekil 2.3 WSDL Mekanizması

Diyagramda bir web servisi Java API'leri ile UDDI'a kaydedilmekte böylelikle bir iş ortağı veya diğer sistemler servisin UDDI'da ki kayıt bilgisiyle çağırım detaylarını içeren WSDL dökümanına ulaşabilmektedir. WSDL'i elde ettikten sonra programcı web servisine ulaşabilecek Java proxy objesi türetebilir veya daha basitçe alt seviyeli bir SOAP-API kullanabilir [2].

2.10.2 WSDL enformasyon modeli

WSDL Enformasyon modeli soyut spesifikasyonlar ile bu spesifikasyonların ayrı uygulamalarının ayrımının tüm avantajlarını yansıtır. Bu, servis arayüz tanımının ve servis uygulama tanımının ayrımını yansıtır. Tümüyle XML uygulamalarında olduğu gibi, WSDL şeması dildeki ana veya birkaç yüksek seviyeli elemanları tanımlar. Web servisin WSDL'deki ana elemanlarına bakarsak, bunlar;

PortType: Bir web servisin ne yaptığının en net tanımıdır. Kalan elemanlar portType'ı detaylandırır.

Message: Method ve operasyonlar tarafından gereksinim duyulan parametreleri tanımlar.

types: Web servisde kullanılan ve message elemanları tarafından referans alınan tüm data tiplerini tanımlar.

binding: Port type elemanlarının (soyut arayüzlerin) nasıl data format ve protokolleri kombinasyonuna çevrileceğinin detaylarını kapsar (HTTP üzerinden SOAP).

port: Network son noktasında bağlanmanın ne şekilde gerçekleşeceğini tanımlar (HTTP URL'i tanımladığımız yer olarak düşünebiliriz). Web servise ev sahipliği yapan network adresini tutar.

service: isimlendirilmiş portlar kolleksiyonu.

Aşağıda verilen XML dökümanı Bölüm 3.2.2'de verilen Octo Guide uygulamasının CarModelCon isimli web servisinin WSDL dökümanıdır. İyi tanımlanmış bir XML dökümanıdır ve yukarda açıkladığımız bölümlerden oluşmaktadır. Kısaca "portType" servisin ismini, bu servisten çağrılabilecek metodun adını (burada getModels'dir) ve bu metodun alıcağı ve geriye döndüreceği parametreleri tanımlamaktadır. "message" etiketleri ise bu parametrelerin tiplerini tanımlamaktadır. "service" etiketi ise servisin bulunduğu network adresini vermektedir. Şekil 2.3'de gösterildiği gibi kullanıcı bu dökümanı elde ettikten sonra web servise bağlanabilmek için gerekli kodu üretebilir. Yani bir WSDL dökümanı tek başına bir web servisini tanımlamak için yeterlidir.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions targetNamespace=
    "http://localhost:80/axis/CarModelCon.jws"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:serviceNS="http://localhost:80/axis/CarModelCon.jws"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <message name="getModelsRequest">
```

```
    <part name="arg0" type="xsd:string" />
  </message>
  <message name="getModelsResponse">
    <part name="getModelsResult" type="xsd:string" />
  </message>
  <portType name="CarModelConPortType">
    <operation name="getModels">
      <input message="serviceNS:getModelsRequest" />
      <output message="serviceNS:getModelsResponse" />
    </operation>
  </portType>
  <binding name="CarModelConSoapBinding"
    type="serviceNS:CarModelConPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getModels">
      <soap:operation soapAction="" style="rpc" />
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="" use="encoded" />
      </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="" use="encoded" />
      </output>
    </operation>
  </binding>
  <service name="CarModelCon">
    <port binding="serviceNS:CarModelConSoapBinding"
      name="CarModelConPort">
      <soap:address location="http://localhost:80/axis/CarModelCon.jws" />
    </port>
  </service>
</definitions>
```

2.11 SOAP

Genel anlamda SOAP, herhangi bir platformda herhangi bir dilde herhangi bir objenin karşılıklı haberleşmesine izin verir. Günümüzde SOAP 60'ın üzerinde dilde 20'nin üzerinde platformda uygulanmaktadır. Birdenbire objeler heryerde lokal veya uzak, büyük veya küçük etkileşebilmektedir [1].

Web servis anlamında bir protokol olarak SOAP, UDDI ile birlikte işletmeler arasında kayıt ve haberleşme servislerini sağlamaktadır. SOAP uzak objelerin methodlarını çağırabilen uygulamalar inşa etmemize imkan verir. SOAP, iki sistemin aynı platformda çalışması veya programın aynı dilde yazılması ihtiyacını ortadan kaldırır. Metodları bir dizi binary protokolle çağırmaktansa SOAP method çağırımları için XML'i kullanır. İstekte bulunan uygulamayla istekte bulunulan obje arasında ki tüm veri XML data etiketleri şeklinde HTTP üzerinden gönderilir. Web servis bakış açısından SOAP hem server hemde client olarak uygulanabilir.

Bir SOAP client uzak dağınık bir sistemdeki metodu çağırmak için gerekli veriyi, XML dökümanını oluşturan bir programdır. Bir SOAP client web server olabileceği gibi server-tabanlı bir uygulamada olabilir. Mesajlar ve istemler SOAP clientları tarafından tipik olarak HTTP üzerinden gönderilir. Sonuç olarak SOAP dökümanları herhangi bir firewall'ı geçebilerek, çeşitli platformlarla veri alışverişine olanak sağlarlar.

Bir SOAP server ise basitçe SOAP mesajlarını dinleyen ve bir dağıtıcı ve SOAP doküman tercumanı gibi davranan özel bir koddur. SOAP serverlar HTTP bağlantısı üzerinden gelen dökümanın diğer uçtaki objenin anlıyabileceği bir dile dönüştürülmesini sağlarlar. Tüm haberleşme XML formunda yapıldığı için herhangi bir dildeki objeler SOAP ile başka bir dillede haberleşebilirler. Diğer ucun haberleşmeyi anladığı ve işlemleri yerine getirdiği SOAP serverin görevidir.

2.11.1 SOAP gerçekten nedir?

SOAP birbirleri ve platformları hakkında öncü bilgiye sahip olmayan farklı partilerin (objelerin) komünükasyonuna olanak veren genişletilebilir, text-tabanlı bir çalışmadır. Net üzerindeki objeler açısından SOAP uçbir görücü usulüdür. Client uygulamalar servislerle daha önce aralarında herhangi bir kabul olmadan gevşekçe-eşleştirilmiş ortamlarda dinamik olarak birbirlerini keşfeder ve bağlanırlar.

SOAP genişletilebilir çünkü SOAP clientları, serverları ve protokolu uygulamayı kırmadan değişip gelişebilir. SOAP, ara katmanları ve katlı mimarileri destekleme açısından zengindir. Bunun anlamı işlenen düğümler isteğin client ve server arasında gittiği bir dizinde oturur. Bu ara düğümler SOAP tarafından belirtilen headerların kullanımıyla mesajların bölümlerini işlerler. Bu headerlar hangi düğümün mesajın hangi bölümünde çalıştığını clientın tanımlamasına izin verir. Bu tip ara header işlemi client uygulamayla ara düğüm arasındaki özel kontratla uygulanır. SOAP headerlar için mustUnderstand özelliği sağlar, buda client'a işlemin zorunlu veya opsiyonel olduğunu tanımlamasına izin verir. SOAP ayrıca veri kodlama kurallarını tanımlar. Son olarak SOAP client ve serverların SOAP kullanarak uzak prosedür çağırmasına olanak veren bir dizi kurallar belirtir.

2.11.2 Mesaj formatı

SOAP tüm bunları standartlaştırılmış bir mesaj formatı içeriğinde yapar. Bu mesajın birinci kısmı "text/xml" MIME tipine sahiptir ve SOAP zarfını kapsar [6]. Bu zarf bir XML dökümanıdır ve opsiyonel bir header ve zorunlu bir body öğesi taşır. Zarfin body bölümü her zaman mesajın final kısmıdır, header girişleri ise ara işlemleri uygulayan düğümleri hedefler.

2.11.3 SOAP zarfının anatomisi

```
<SOAP-ENV:Envelope SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Header>
     <t:Transaction xmlns:t="some-URI">
          SOAP-ENV:mustUnderstand="1"  5
     </t:Transaction>
```

```
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <ns1:getModels xmlns:ns1="CarModelCon">
      <arg1 xsi:type="xsd:string">DAEWOO</arg1>
       </ns1:getModels>
    </SOAP-ENV:Body>
</SOAP-Envelope>
```

Bu örnekte bölüm 3.2.2'de verilen Oto Guide uygulamasını SOAP server'a yolladığı SOAP envelope yer alıyor. getModels istemi network üzerinde herhangi bir noktadaki Car Model Con servisine yollanıyor. İstek parametre olarak bir string alıyor ve SOAP yanıtında yine bir string değeri geri döndürülüyor. SOAP zarfı SOAP mesajını temsil eden XML dökümanının top elemanıdır. XML namespaceleri SOAP tanımlayıcıları uygulamaya özel tanımlayıcılardan ayırmak için kullanılır.

Namespaceler: Örnekteki ilk namespace SOAP mesajındaki eleman ve özellikleri tanımlayan SOAP şemayı referans alıyor, ikinci namespace ise SOAP kodlamalarını.

Header: SOAP zarf headerında ki ilk eleman bir namespace özelliği ve 1 değerli mustunderstand özelliği ile eşlik edilen bir işlem elemanı. "mustUnderstand" 1 değerine sahip olduğu için server bu mesajı işlem düğümü üzerinde ara işlemleri yapmak olarak kabul ediyor. Bunu server ve client önceden header elemanının nasıl işleneceği konusunda anlaşmışlar olarakta algılayabiliriz dolayısıyla server elemanın içeriğiyle ne yapması gerektiğini bilmektedir.

Bu mesajı kabul eden server headerın semantikleri ile ne yapacağını anlamazsa istemi tamamen reddedip bir hata göndermesi gerekir. Hata elemanı SOAP body'nin özel ve iyi tanımlanmış mekanizması olup client'a hata verisini geri gönderir.

Bunun gibi ara işlem düğümleri SOAP genişletilebilirliğinin örnekleridir. Clientlar bu gibi düğümleri SOAP mesajı içine koyarak mesaj bodysinin işlenmesinden önce özel işlemlerin yapılmasını belirtirler.

Body: Body kısmında gördüğümüzün tümü bir namespace tarafından tanımlanmış birkaç XML elemanı. Doğru döküman semantiklerini anlayıp doğru şeyleri yapmak ise SOAP servera kalıyor. Server etkin bir şekilde XML yüküyle başedebilmek için anlamlı şekilde bir çalışma çerçevesi sağlıyor. Anlamlı derken server bazı arka plan veritabanında uzak prosedür çağırımı ile mesaj body'de bulunan hisse senedi için fiyat verisinin çekilmesini vurguluyor. Tüm olay SOAP-RPC arkasında yer alıyor.

2.11.4 SOAP-RPC

SOAP mesajları temel olarak yollayandan alıcıya tek yönlü aktarımlardır akat sıklıkla SOAP mesajları istek/cevap mekanizmasının kombinasyonu olarak uygulanırlar. SOAP kullanarak RPC yapmak için birkaç çeviri mtakib edilmelidir. İstek ve cevap mesajları yapılar olarak kod haline getirilmelidir. Operasyonun her input parametresi için parametreyle aynı isimde bir eleman ve her çıktı parametresi için elemanla uygun bir isim olmalıdır. Aşağıda yer alan istem ve yanıt yine bölüm 3.2.2'deki araba uygulamasının SOAP server'a yolladığı ve aldığı SOAP Envelope'ların Body öğelerini gösteriyor.

İstem

```
<SOAP-ENV:Body>
    <ns1:getModels xmlns:ns1="CarModelCon">
    <arg1 xsi:type="xsd:string">DAEWOO</arg1>
    </ns1:getModels>
 </SOAP-ENV:Body>
```

Yanıt:

```
<SOAP-ENV:Body>
   <ns1:getModelsResponse xmlns:ns1="CarModelCon">
    <getModelsResult xsi:type="xsd:string">MATIZ SE/KLM LANOS 1.5
      SE/4K LANOS 1.5 SE/5K NUBIRA 1.6 SX/4K NUBIRA 1.6
      SX/4K/OTM LEGANZA CDX/4K LEGANZA CDX/OTM CHAIRMAN 600
      SX KORANDO 2.3 KORANDO 2.9 TD MUSSO 2.9 TD
   </getModelsResult>
  </ns1:getModelsResponse>
</SOAP-ENV:Body>
```

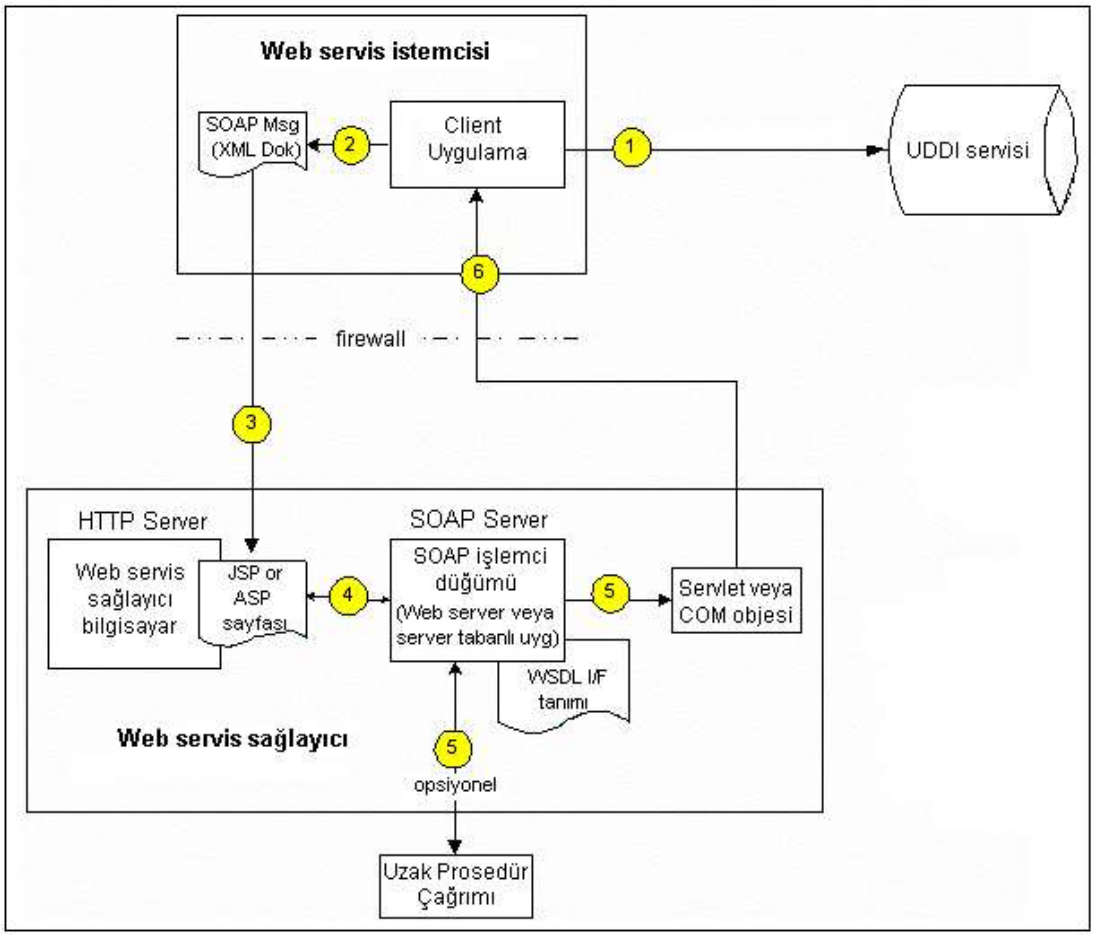
İstek getModels methodunu çağırıyor, yanıt ise getModelsResponse operasyonunu tanımlıyor. Bu çıktı yapısı getModelsResult diye adlandırılan bir elemanı kapsıyor, buda metod çağrımı sonucu değerleri döndürüyor.

2.11.5 Bir SOAP kullanım durumu

SOAP zarfını da detayları ile verdiğimize göre biraz geriye çekilip SOAP'a durum kullanım perspektifinden bakmakta Web ortamında olayın nasıl gerçekleştiğini anlamak açısından fayda var.

1. Bir SOAP client UDDI kayıt alanını bir web servisi bulmak için kullanıyor.

2. Client uygulama istek/cevap operasyonunu gerçekleştirebilecek XML dökümanını SOAP mesaj olarak oluşturuyor.
3. Client SOAP mesajı bir web serverdaki SOAP istekleri dinleyen JSP veya ASP sayfasına yolluyor.
4. SOAP server SOAP paketindeki dökümanı parse ederek elde ettiği parametrelerle kendi domain'indeki uygun obje metodunu çağırıyor.
5. Çağrılan obje gerekli fonksiyonları icra ettikten sonra istenilen verileri SOAP servera SOAP zarfı şekline getirilmek üzere geri gönderiyor ve istemciye paket döndürülüyor.
6. Client objeyi alarak zarfı açıp cevabı istemde bulunan programa geri gönderiyor.



Şekil 2.4 SOAP Mekanizması

2.12 UDDI

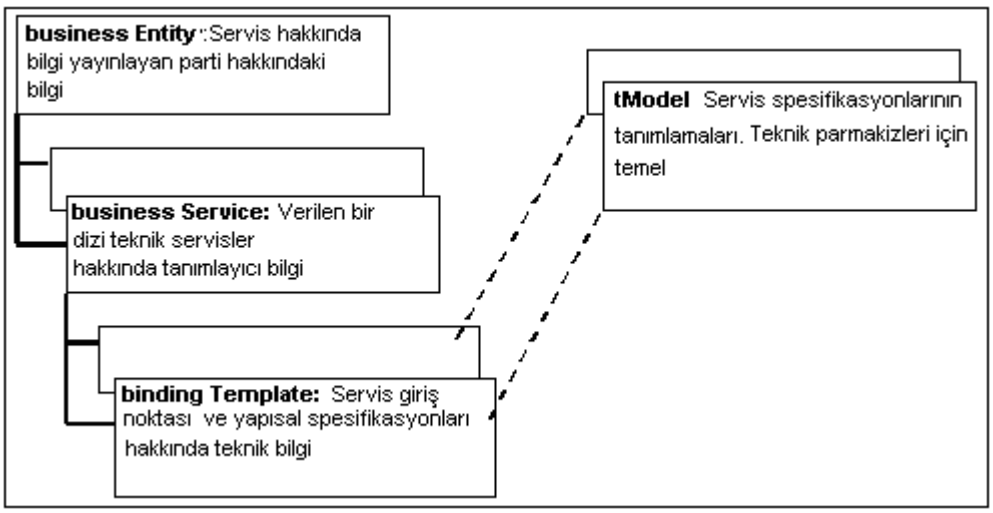
Adının da belirttiği gibi evrensel tanımlama, keşif ve entegrasyon (universal description, discovery and integration) standartı, işletmelerin kendilerini, sağladıkları servis tiplerini tanımlamalarına, UDDI kayıt alanına kayıt olmalarına ve

yayınlamalarına izin veren bir mekanizma sağlar. Bu şekilde yayınlanan işletmeler search edilebilir, sorgulanabilir veya SOAP mesajlarını kullanan diğer işletmeler tarafından keşfedilebilir. Birbirlerini bulan işletmeler ortaklık için servislerini entegre edip müşterilerine servis sağlayabilirler [6].

Kavramsal olarak, UDDI işletme kayıtlarında sağlanan bilgi üç bölümden oluşur. White Pages daha çok telefon eşdeğeri gibidir ve adres, kontak ve tanımlayıcılar bulundurur. Yellow Pages telefon eşdeğeriyle paraleldir ve standart taxonomies üzerine kurulu endüstri kategarizasyonlarını taşır NAICS, UNSPSC, GSC kod gibi. Bilgiyi endüstri kodu, coğrafi kod ve işletme tanımlama kodu gibi kaydetmek diğer arama servislerine bu klasifikasyon bilgisini daha iyi indeksleme ve klasifikasyon için iyi bir başlama noktası sağlar. Green Pages, işletmeler tarafından servisler hakkında belirtilen teknik bilgiyi kapsar. Bunlar çeşitli dosyalara pointer desteği kapsadığı kadar web servisler için olan spesifikasyonlara referansları gerekli olursa da URL tabanlı keşif mekanizmalarını kapsarlar. UDDI kayıt düğümleri önceki bilgiyi kopyalarlar ve veriyi aralarında kopyalayarak aynı dizin bilgisini herhangi bir düğüm bilgisinden sağlarlar.

2.12.1 UDDI'ın ana yapıları

Bir kayıdı oluşturan bilgi 4 veri yapı tipinden oluşmaktadır. Şekil 2.5 UDDI spesifikasyonunda belirtilen 4 yapıyı göstermektedir..



Şekil 2.5 UDDI'ın Çekirdek Yapıları

businessEntity: Bu yapı bir işletme veya alan hakkında bilgiyi alır, işletme tarafından kendi hakkında tanımlayıcı bilgi ve servisleri yayınlamak için kullanılır. XML açısından businessEntity işletme veya alan hakkında açıklayıcı bilgi tutan yüksek seviyeli veri yapısıdır. Servis tanımlamaları ve teknik bilgi businessEntity içinde açıklanır. businessEntity yapısı UDDI'ın Yelow Pages ve White Pages'lerinde bulunan işletme ve veriyi tanımlayan tek anahtara sahiptir.

businessService: Bu yapı businessEntity tarafından sağlanan servisleri ve işletmelerin işlemlerini temsil eder. Servisi temsil eden unique key servis adı, opsiyonel bir tanımlama ve de teknik bilgiyi tutan bindingTemplate yapılarını kapsar.

bindingTemplate: Bu yapı verilen servis uygulamasının teknik karakteristiklerini tanımlayan önemli veriyi temsil eder. Genelde servisi çağırmak için gerekli veriye sahiptir. Her bindingTemplate tek bir mantıksal parent businessService'e oda tek bir parent businessEntity'e sahiptir. Her bindingTemplate tek bir unique anahtara, ilişkili anahtar servise en önemlisi icra noktasına ve tModelInstanceDetails'a sahiptir. accessPoint verilen bir web servisi çağırmak için gerekli adresi temsil eder. Bu bir URL, bir e-mail adresi hatta telefon numarası olabilir.

tModel: tModel'in oynadığı birinci rol teknik spesifikasyonu temsil etmektir. UDDI kayıdı içinde tModel'in amacı soyutlamaya dayalı bir referans sistemi sağlamaktır. Bir tModel anahtara, isme, opsiyonel tanımlamaya, veri hakkında daha fazla bilginin çekilebileceği bir URL'e sahiptir. Bir tModel örneği bir tel protokolü çerçevesini çizen spesifikasyon olabilir.

2.13 JSP

JavaServer Page (JSP) teknolojisi web developer ve dizaynırlarına mevcut iş sistemleriyle etkileşimli dinamik, zengin, bakımı kolay, hızlı geliştirilebilen web sayfaları oluşturmalarına izin verir [9]. Java ailesinin bir bölümü olarak JSP teknolojisi platformdan bağımsız web-tabanlı uygulamaların hızlı geliştirilebilmesini sağlar. JavaServer Pages teknolojisi kullanıcı arayüzünü koddan ayırarak dizaynıra, altta yatan yapıyı değiştirmeden sayfayı yeniden düzenleme imkanı sağlar.

JavaServer Pages teknolojisi Java programlama dilinde yazılmış XML benzeri etiketler ve scriptler kullanarak sayfayı oluşturan kodu çerçeveler. İlaveten uygulama server-tabanlı kaynaklarda bulunabilir ve sayfa bunlara bu etiket ve scriptlerle ulaşabilir. Tüm formatlı etiketler (HTML veya XML) direkt olarak yanıt sayfasına geçirilir. Sayfa mantığını dizayn ve sergilenmesinden ayırarak JSP teknolojisi daha hızlı ve kolay web-tabanlı uygulamalar oluşturur.

JavaServer Pages teknolojisi Java servlet teknolojisinin genişletilmiş bir uzantısıdır. Servletlar platformdan bağımsız, web-server yapısına uygun %100 saf server-tabanlı Java modülleridir ve web server'in kapasitesini minimum bakım ve destekle artırırlar. JSP teknolojisi ve servletlar birlikte diğer dinamik web programlama/scripting tiplerine çekici bir alternatif sağlamakta ve platform bağımsızlığı, geliştirilmiş performans, kullanıcı yüzünün koddan ayrılması, yönetim kolaylığı ve en önemlisi kullanım kolaylığı sağlamaktadır.

2.14 Java Servlet'lar

Java Servletlar küçük, platformdan bağımsız web server'in fonksiyonalitesini çeşitli yollardan geliştiren Java programlarıdır [9]. Applet'lar client için neyse servletlarda server için odur. Appletlar client'a dinamik olarak yüklenen ve host'un kapasitesini artıran küçük Java programlarıdır.

Servletlar appletlardan farklılık gösterirler, web browserda veya grafik kullanıcı arayüzü ile çalışmazlar. Bunun yerine servletlar web server üzerinde koşan servlet motoru ile istemler ve cevaplar yoluyla etkileşirler. İstem-cevap paradimi HyperText TransferProtocol (HTTP) davranışı üzerinde modellenmiştir.

Bir client program web server'a ulaşır ve istemde bulunur bu program internet üzerinden bağlanan bir web browser veya başka bir program olabilir. Bu istem web server ile koşan servlet motoru tarafından işlenir ve yanıt servlet'a döndürülür. Sonra servlet'ta yanıtı HTTP formunda istemciye döndürür.

2.14.1 Java servletların özellikleri

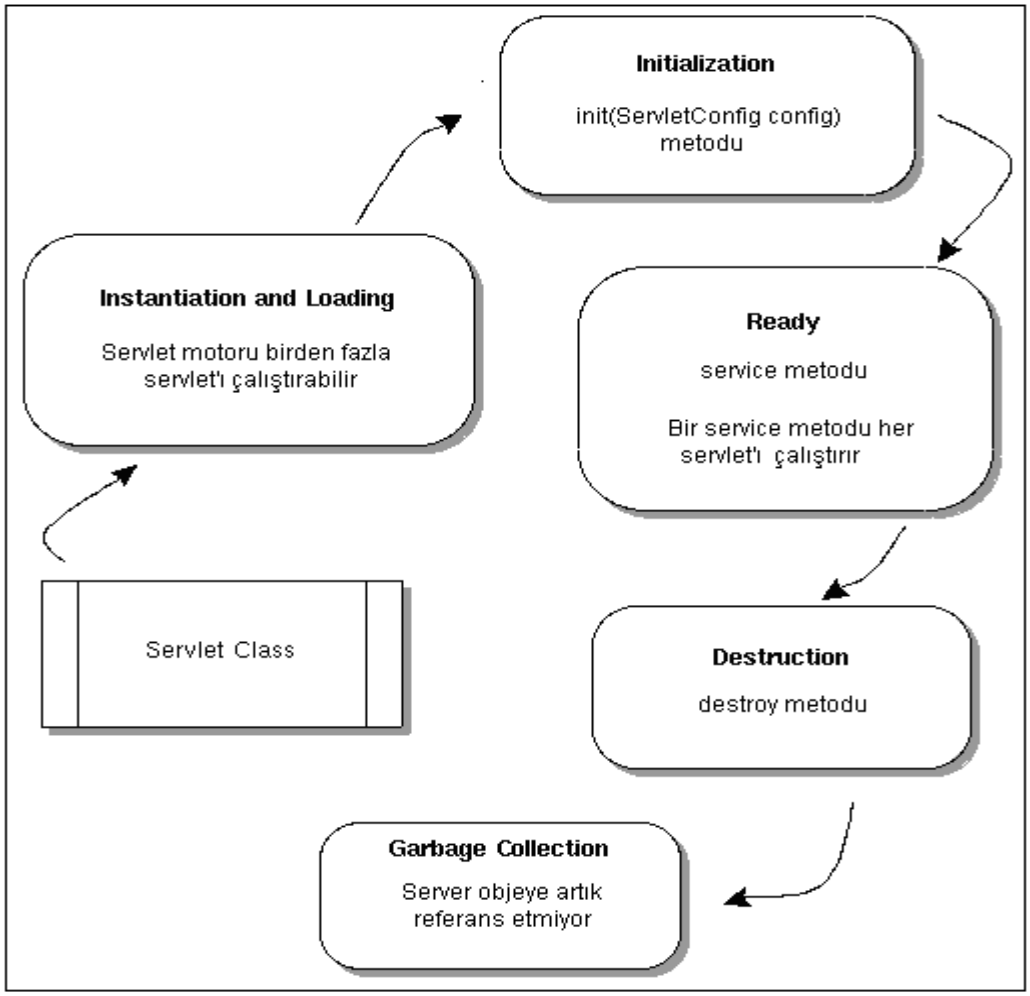
Servletların genel olarak diğer server mekanizmalarına göre avantajları şunlardır..

- CGI scriptlerinden daha hızlıdırlar çünkü farklı bir işlem modeli kullanırlar.
- Çoğu web server tarafından desteklenen standart bir API kullanırlar.

- Java dilinin tüm avantajlarını taşır, platform bağımsızlığı ve kolay development gibi.
- Java platformunda geçerli geniş bir API grubunu kullanabilirler.

2.14.2 Servlet yaşam döngüsü

Bir Java servlet, servletin nasıl yüklendiği initialize edildiği, istemleri nasıl kabul ettiği ve cevapladığı ve servis dışına nasıl çıkarıldığını tanımlayan bir yaşam döngüsüne sahiptir. Kod içinde servlet yaşam döngüsü javax.servlet.Servlet arayüzü tarafından tanımlanır.



Şekil 2.6 Servlet Yaşam Döngüsü

Tüm Java servletlar direkt veya indirekt olarak javax.servlet.Servlet arayüzü kullanırlar böylece servlet motoru içinde çalışabilirler. Servlet motoru servlet'ları işleyebilmek için web server'a özelleştirilmiş bir ilavedir. Servlet motoru network servisleri sağlar, MIME işlemlerini anlar ve servlet container'larını çalıştırır.

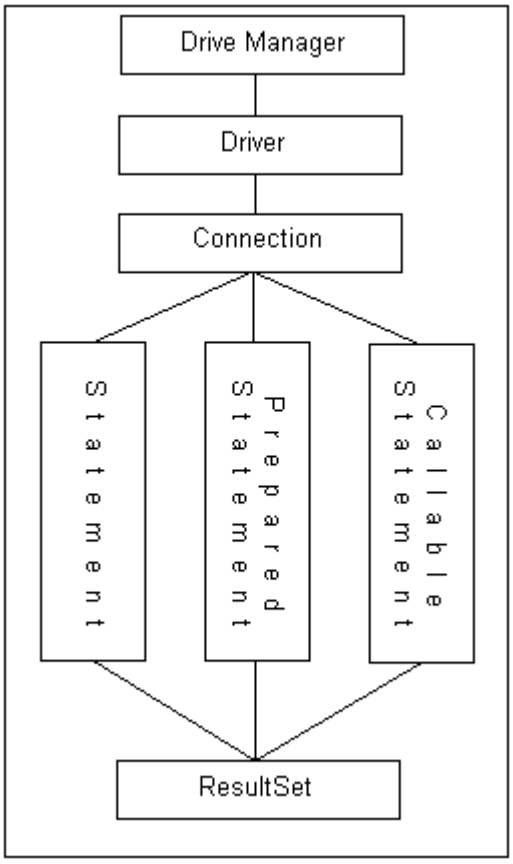
javax.servlet.Servlet arayüzü servlet yaşam döngüsü sırasında spesifik zaman ve durumlarda çağrılan metodlar tanımlar.

2.15 JDBC

Java Database Connectivity (JDBC) API işletme düzeyinde development için en önemli API'lerden birisidir çünkü her zaman bir veri tabanını işleme ihtiyacı duyulur. JDBC veri tabanından bağımsız standart bir API sunar fakat gerek duyulursada veri tabanına spesifik özelliklerin kullanılmasına imkan verir [4].

Bir çok veri tabanı, veritabanı ile haberleşmek için farklı API'ler kullanırlar. Windows platformunda hatta bazı UNIX platformlarında ODBC standart bir veritabanı API'si sunarak bir çok veri tabanı ile çalışmaya olanak verir. JDBC'de problemi ODBC gibi çözer, oda standart bir veritabanı API'si sunar.

ODBC'de olduğu gibi JDBC paketi tek başına nasıl bir veri tabanına bağlanacağını bilemez. Bu diğer paketlerde bulunan ve uygulamayı sağlayan API framework'üdür. Hangi veri tabanı olursa olsun ona uygun bir JDBC sürücüsü vardır.



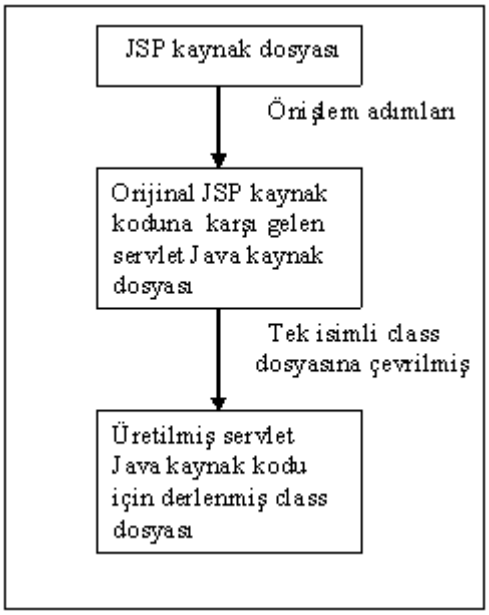
Şekil 2.7 JDBC API

2.16 Servlet &JSP Container

Servlet ve JSP container olarak projede kullanılan Tomcat'in başlıca komponentleri şunlardır [6];

- Dahili HTTP server, çoğunlukla development aktivitelerini desteklemek için,
- Haberleşme adaptörleri, client istemlerini kabul eder, istemleri uygun container'a geçirir ve sonra cevapları client'a geri gönderir. Bu istem ve yanıtlar HttpServlet ve HttpServletResponse classlarının bölümleridir. Bu classlar JSP içinde ayrıca kullanılmaktadır.
- Servlet ve JSP container komponentleri.
- Seans yönetim komponentleri, geçerli seans obje havuzlarını yaratarak ve yöneterek seansları yönetirler.

Şekil 2.8 JSP'lerin Java kodunu sonrada derleme işlem adımlarını göstermektedir. Çoğu JSP container JSP sayfasını özel olarak isimlendirilmiş Java servlet kaynak dosyasına çevirir. Farklı versiyonları farklı isimli class dosyalarına çevirir. Doğru class dosyasını işlemek container'ın sorumluluğudur.



Şekil 2.8 JSP&Servlet Container

Servlet ve JSP konteynırı, servlet ve JSP'lerin yaşam döngüsünü kontrol etmekten sorumludur. Etkinlik için, container genellikle ayrılmış servlet durum havuzunu yönetmektedir.

Servlet ve JSP container'ı aynı zamanda client seansının durum bilgisinin güncellenmesinin yönetilmesinden de sorumludur. Container aynı zamanda seans bilgisinin sonlanmasındanda sorumludur. Sonlanma parametresi servlet veya JSP bir Java Application Server'a yerleştirildiği zaman set edilmektedir.

Bir servlet veya JSP container'ı web kaynaklarının indirekt kullanımında da desteklemelidir. Bir container aynı zamanda bir dağıtım kapasitesini desteklemelidir. Bir servlet veya JSP bir istemin işlenmesini gerçekleştirir ve istem, yanıt objelerini bir diğer web komponentine dağıtır. Dağıtımdan sonra yeni web komponenti container'a modifiye edilmiş komponenti kullanıcıya döndürmesi için kullanılmasından sorumludur.

2.17 AXIS Mimarisi

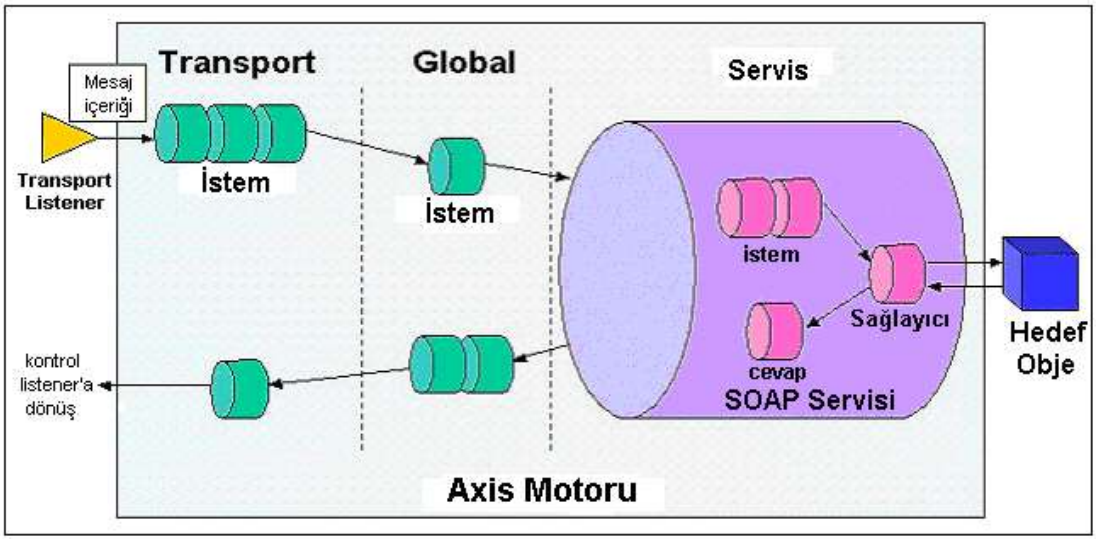
Axis birlikte çalışan birkaç alt sistemden meydana gelmektedir [1]. Bu bölümde paketin nasıl çalıştığı hakkında kısa bir özet vericeğiz.

Axis'de Handler'lar ve Message Path;

Basit şekilde Axis'in tümü mesajların işlenmesi prosedürüdür. Merkezi Axis işlemcisi mantığı çalıştığı zaman bir dizi handler sırasıyla uyandırılır. Bu sıra 2 faktör tarafından belirlenir, deploy konfigürasyonu ve Axis motorunun istemci veya hizmetçi MessageContext'i olması. MessageContext birkaç önemli bölümden oluşan bir yapıdır: 1) bir "istek" mesajı, 2) bir "yanıt" mesajı, 3) bir grup özellikler.

Axis'in uyandırılmasının 2 basit yolu vardır.

1) Server olarak, TransportListener bir MessageContext yaratır ve Axis işlemci yapısını uyandırır.
2) Client olarak, uygulama kodu bir MessageContext üretir ve Axis işlemci yapısını uyandırır. Her 2 durumdada Axis yapısının görevi MessageContext'i dizayn edildiği şekilde işlenecek olan Handler'lara MessageContext'i geçirmektir.



Şekil 2.9 Axis

Message Transport Listener'a (protokole özel bir şekilde) ulaşıyor. Bu durumda bir HTTP servlet olduğunu kabul edelim. Protokole özel veriyi Message objesinin içine paketlemek ve Message bir MessageContext'in içine koymak Listener'in görevidir. MessageContext'e ayrıca Listener tarafından çeşitli özelliklerde yüklenir.. bu durumda örnek olarak SOAP Action HTTP header özelliği "http.SOAPAction" değeri olarak. TransportListener ayrıca MessageContext'teki transportName'i bu durumda "http" olarak yükler. MessageContext yollanmaya hazır olduğu zaman, Listener onu Axis motoruna yollar.

Axis motorunun ilk görevi transport'a isimile bakmaktır. Bu bir "istem akışı", bir "cevap akışı" veya ikisini birden kapsayan bir objede sonuçlanabilir. Eğer bir transport istem akışı mevcutsa, istem MessageContext çağrılma metoduna aktarılarak uyandırılır. Bu işlem akış konfigürasyonunda tanımlı tüm Handler'ların çağrılmasıyla sonuçlanır.

Transport istem handler'larından sonra, motor bir global istem akışı belirler eğer konfigüre edildiyse ve sonrada belirlenmiş handler'lar uyandırılır.

Client'ta Mesaj Path;

Client tarafında Message Path'de kapsam sırasının tersi olması şeklinde benzer şekildedir.

Eğer varsa servis handler'ı önce çağrılır, servis uzak bir düğümt arafından sağlandığı için client tarafında bir "servis sağlayıcı" yoktur fakat hala istem yanıt akışı olasılığı vardır. Servis istem ve yanıt akışları sistemin dışına doğru mesajın servise özel işlemlerini yapma hizmetini verir ve aynı şeyi yanıt mesajının çağrıcıya dönüş yolunda da yapar.

Servis istem akışından sonra eğer varsa global requestFlow Transport'u uyandırılır. Görevi protokole özel gerekli operasyonları hedef SOAP server'a mesajı ulaştırıp, almak olan Transport Sender mesajı göndermek için uyandırılır. Cevap MessageContext'in responseMessage alanına yerleştirilir ve MessageContext yanıt akışında yoluna devam eder.. ilk transport, sonra global ve sonunda servis.

3. WEB SERVİS UYGULAMASI

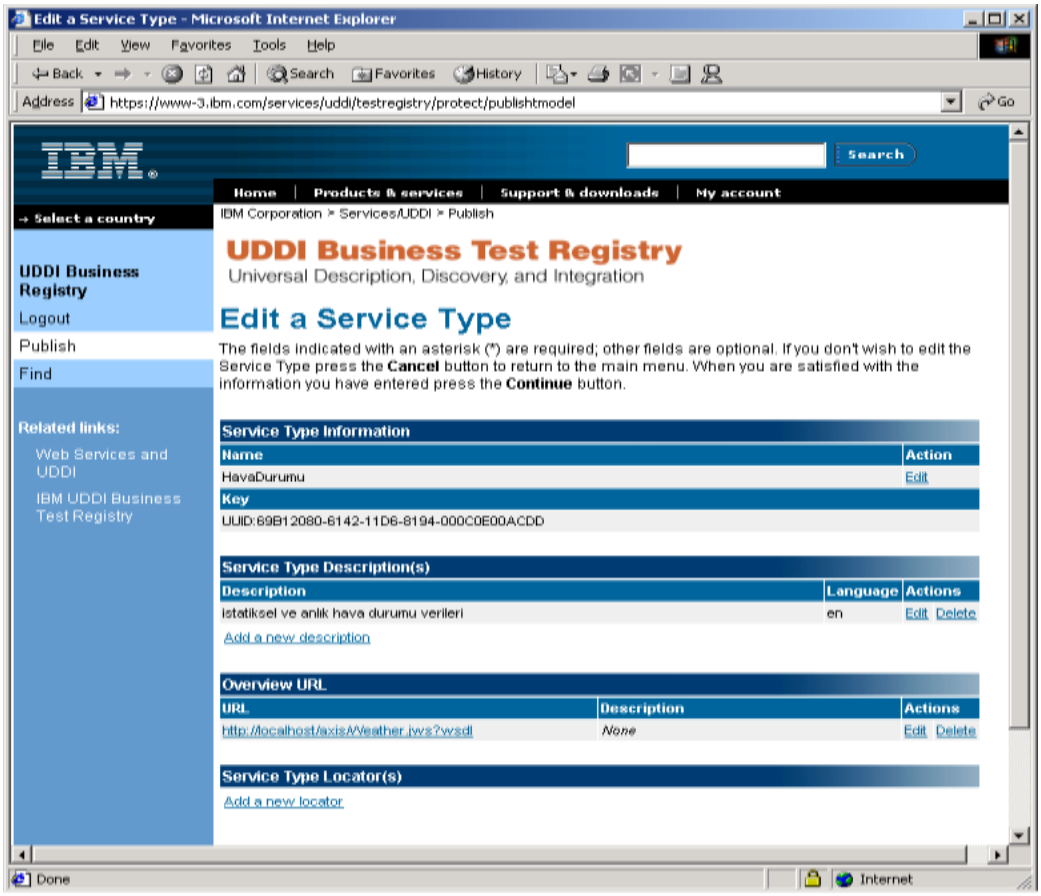
Web servis uygulaması hava durumu ve araba uygulaması olmak üzere iki bölümden oluşmaktadır. Her iki uygulamada web servis mimarisi üzerine kurulmuştur, bölüm 2.3'dede bahsettiğimiz gibi UDDI biraz genel yapıda anlatılan durumdan farklı olarak kullanılmıştır. Bunun nedeni ise uygulamanın bir internet sitesi üzerinde değilde local bir networkde çalışmasıdır. Uygulama istenirse bir web server'a yerleştirilerek internet kullanımına açılabilir. Hardcopy (disket, CD, vb), ftp, e-mail yoluyla WSDL [4] dosyası ulaştırılan kullanıcılar servisden direkt olarakta yararlanabilirler tabi burada servise bağlanmak için gerekli kodu kendilerinin oluşturması gerekmektedir.

Sistemin genel özelliklerine bakacak olursak şu maddeler halinde sıralıyabiliriz:

1. Uygulamalarda üç katmanlı bir yapı kullanılmıştır. Prezentasyon katmanı, web servis katmanı ve veri katmanı birbirlerinden ayrılmışlardır böylece herhangi birinin uygulamasının değiştirilmesinin diğerlerine etkisi ortadan kaldırılmıştır. Bu yapı ileride yapılabilecek değişiklikler ve güncellemeler içinde büyük kolaylıklar sağlamaktadır.

2. Bölüm 2.3'de ve bu bölümün girişinde bahsettiğimiz gibi UDDI alanı tam etkin bir şekilde kullanılamamıştır. Bunun nedeni ise uygulamanın local bir uygulama olarak geliştirilmek zorunda kalmasıdır. Uygulamanın bir ISP'ye yerleştirilebilmesi için Bölüm 3.1'de bahsettiğimiz teknolojilerin ISP tarafından desteklenmesi gerekmektedir oysaki bunların bazıları daha beta sürümleri olan teknolojilerdir. Buna rağmen tüm bu server ve teknolojilerin bir ISP'ye yüklenmesi yaklaşık 100 MB'lık bir alan kaplıyacaktır bundan daha önemlisi bu sistemin yönetimi için uzman birininde olması şarttır. Tüm bu nedenlerden servisin internet kullanımına açılması için en doğru çözüm Türk Telekom'dan bir ISP alınmasıdır bu da tez için hayli pahalı bir çözüm olacağından uygulamalar lokal olarak tasarlanıp geliştirilmiştir. Buna rağmen bu uygulamalar üzerinde en ufak bir değişiklik yapmadan istenirse Bölüm 3.1'de

anlatılan server, teknolojilerle birlikte bir ISP server'a yüklenmesi internet üzerinden kullanıma açılmasına yeterli olacaktır. Bu durumda yapılması gereken tek şey Şekil 3.1'de UDDI kayıt alanında görülen servis URL adresinin değiştirilmesidir.



Şekil 3.1 IBM Test Registry

3. Uygulamaların web servis mimarisi üzerine inşa edilmesi onlara bazı üstünlükler sağlamaktadır. WSDL dosyasına sahip bir kullanıcı veya program servisi kullanarak istediği veriye servisin sağladığı method veya fonksiyonu kullanarak direkt olarak ulaşabilir. İlk bakışta aynı şeyi bir internet sayfasının da sağladığını düşünebiliriz fakat web servislerinin uygulamalar arasında platformdan ve kullanılan programlama dilinden bağımsız direkt etkileşim sağladığını düşünürsek yapının üstün taraflarını daha iyi anlayabiliriz.

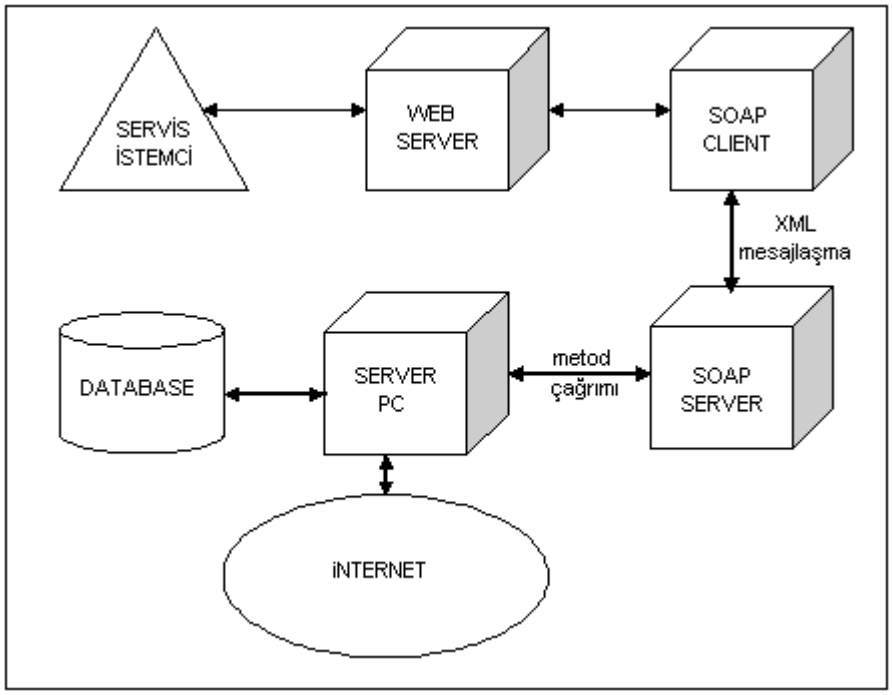
4. Her iki uygulamada Java merkezli bir yaklaşımla geliştirilmiştir. Javanın seçilmesinin ana nedenleri ise Java'nın neredeyse internet uygulamaları için geliştirilmiş olması, platformdan bağımsız taşınabilir olması, J2EE

platformunun sağlamış olduğu teknolojiler, sağladığı olanak ve API'lerle geniş bir kullanım yelpazesine sahip olması, sağladığı bu özelliklerle hızlı development'a olanak tanıması ve ücretsiz olarak sunulmasıdır.

5. Sayfaların oluşturulmasında Java Server Page (JSP) ve Java servlet teknolojileri kullanılmıştır. Kullanılan bu teknolojiler sayesinde sayfalar runtime sırasında dinamik olarak oluşturulmaktadır.

6. Uygulama kullanıcıya istatiksel veriye erişim sağladığı gibi internet üzerinden dinamik veri ulaşımına da imkan tanımaktadır. Kullanıcı isterse bu dinamik veriyi veritabanına kaydedebilmektedir. Daha sonra kaydedilen bu veriler kullanıcıya istatiksel olarak sunulabilmektedir.

Uygulamaların detaylarıyla incelenmesine geçmeden önce sistemin bütünü hakkında bir fikre sahip olmamızda yarar var. Şekil 3.2 sistemin ana modüllerini göstermektedir.



Şekil 3.2 Uygulama Modülleri

Sistemin detaylarına geçmeden önce projede kullanılan server ve teknolojilerin vede sistemin nasıl yüklenip çalışabilir hale getiriliceğinden sözet mek istiyorum.

3.1 Sistemin Yüklenmesi

Şekil 3.1'dede görüldüğü gibi uygulama bazı ana elemanlardan oluşmaktadır. Uygulamanın çalışabilmesi için bunların düzgün şekilde yüklenip konfigüre edilmeleri gerekmektedir. Uygulamada platform olarak Win2000 Pro seçildiği için kullanılan server ve programlarında platforma uygun sürümleri kullanılmıştır.

Java Motoru: Uygulama Java merkezli bir yaklaşımla geliştirildiği için tüm kodların çalışabilmesi için bir Java motoruna ihtiyaç duyulmaktadır [5]. Uygulamada java motoru olarak jdk1.3.1 kullanılmış ve tezin eki olan CD'yede yerleştirilmiştir. JDK sayesinde herhangi bir text editör kullanılarak yazılan Java kodu derlenip, çalıştırılabilir. Yüklenmesi son derece kolay olup gerekli dosyalar kaynak kodunda da yer almaktadır ayrıca verilen referansdanda yararlanılabilir, kitabın elektronik versiyonu internet üzerinde de yeralmaktadır [5].

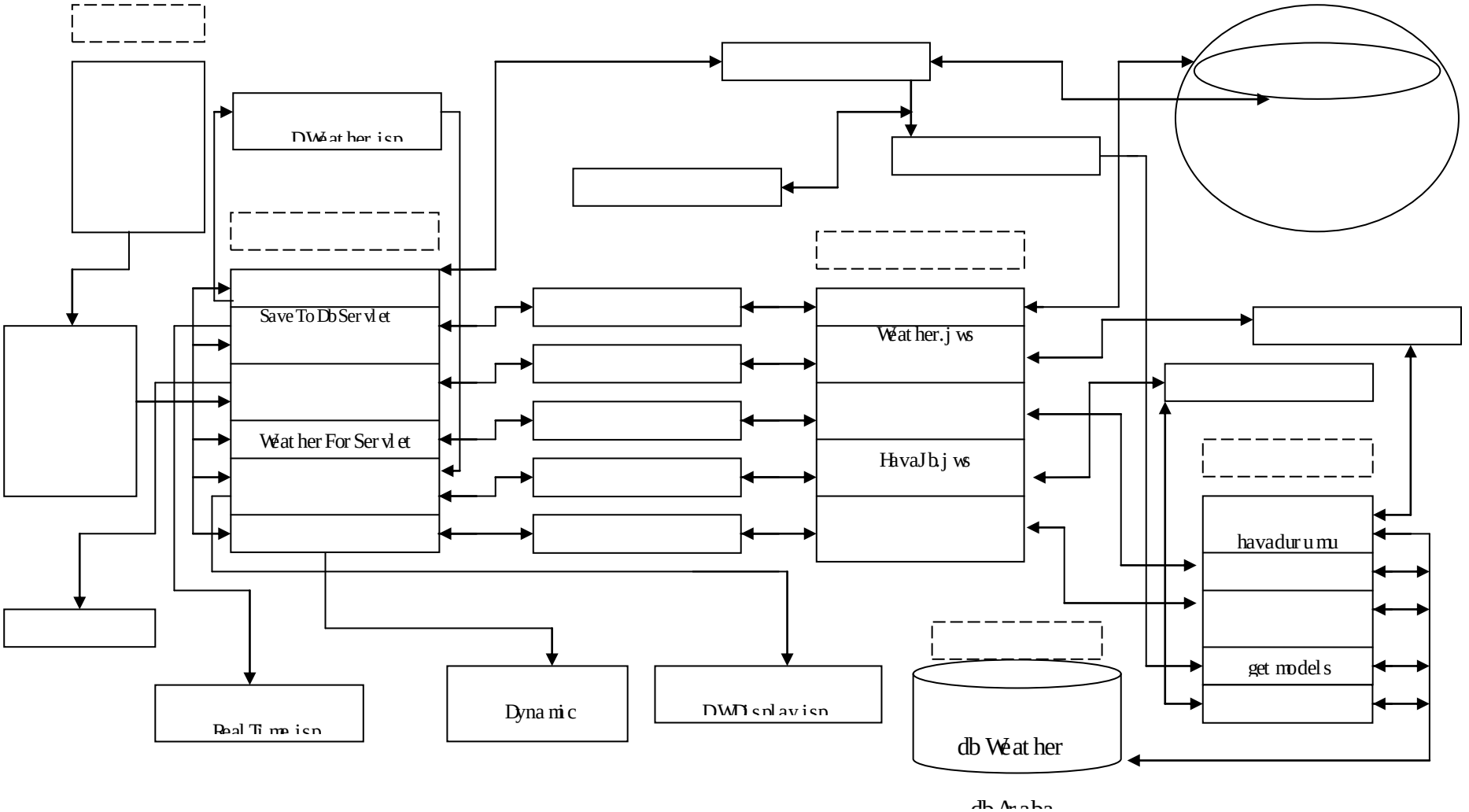
Web Server: Kullanıcının bir web browser kullanarak sisteme bağlanabilmesi için uygulamanın bir web server üzerinde çalışması gerekmektedir. Böylelikle herhangi bir kullanıcı HTTP prokolü ile sisteme bağlanıp JSP sayfalarına ulaşabilecektir. Uygulamada Tomcat 4.0 JSP/Servlet container kullanılmıştır. Uygulamada kullanılan JSP sayfalarının ve servlet'ların çalışabilmesi Java motorunun yanında bir de JSP/servlet container'a ihtiyaç duyulmaktadır [6]. JSP/servlet container sayesinde programcının servlet ve JSP yaşam döngüsü yönetimi, program yükü dengesi, istemcinin GET ve POST istemleri, istemciye HTTP cevabının döndürülmesi gibi konularda [9] endişelenmesi gerekmemektedir. Tüm bunların yönetimiyle container ilgilenmektedir ayrıca runtime sırasında da JSP sayfasını Java servlet koduna dönüştürüp çalıştırmaktadır. Aynı şekilde uygulamada bulunan JSP sayfaları ve servlet kodlarının yönetimini Tomcat 4.0 JSP/Servlet container'ı gerçekleştirmektedir. Bu yüzden uygulamanın çalışması için Tomcat 4.0'ın sistemde yüklü olması gerekmektedir. Uygulamada Tomcat 4.0, Java motoru olarak JDK1.3.1'i kullanmaktadır. Bir JSP/servlet container'ı çalışabilmek için Java motoruna ihtiyaç duymaktadır.

SOAP Server: Bölüm 2.11'de detaylarıyla sözettiğimiz gibi SOAP farklı metodları platformdan ve kullanılan dilden bağımsız olarak XML mesajlaşma ile çağırabilen

bir mekanizma sağlıyordu. Yani bir anlamda Web servis mimarisinin belkemiğini oluşturuyordu. Karşı uygulamaya XML mesajını yollayacak olan program SOAP client gibi davranıyor karşıda mesajı alan SOAP server ise gerekli işlemlerden sonra cevabı SOAP client'a geri yolluyordu [1]. Sonuç olarak uygulamanın çalışabilmesi ve gerekli XML mesajlaşmasının sağlanabilmesi için sistem üzerinde çalışan bir SOAP server'a gereksinim vardır. Uygulamada SOAP server olarak Bölüm 2.12'de anlattığımız Axis kullanılmıştır. Axis'de Tomcat 4.0 ile birlikte Jakarta projesi içinde geliştirilmektedir ve Tomcat'te olduğu gibi internet üzerinden ücretsiz kullanıma sunulmaktadır. Axis Tomcat 4.0 ile uyumlu bir şekilde çalışmakta hatta Tomcat ile birlite yüklenip kullanılmaktadır [1]. Uygulamada SOAP server'a gönderilecek kodu Java program parçaları oluşturmaktadır daha sonra Axis'e yollanan bu mesajların diğer tarafa iletilmesi ve cevabının tekrar istemde bulunan programa döndürülmesi SOAP server'ın yani Axis sorumluluğundadır.

SQL Server: Uygulamanın veritabanı modülünde ise SQL Server 7.0 kullanılmıştır. Kullanımı ve aynı şekilde yüklenmesi son derece kolay şekilde gerçekleşmektedir [10]. Diğerlerinden farklı olarak ücretli bir çözüm sunduğu için ekte veremiyoruz fakat demo versiyonları temin edilip tüm özellikleriyle kullanılabilir.

Bölüm 3.1'de sistemin gerekli bölümlerinden bahsettikten sonra uygulama detaylarına geçebiliriz. Şekil 3.3 uygulamanın bütününü ve modüllerini tek bir şema üzerinde topluyor. Şekil 3.2'de verilen uygulama modülleride programakışı içinde Şekil 3.3'de görülebiliyor. Bunlar istemci olarak bir web browser kullanan bir kullanıcı yani "Client", web server olarak "Application Server", uygulamada objeler arası iletişimi sağlayan Bölüm 2.17 ve 3.1'de anlatılan Axis, SOAP Server olarak görev yapıyor ve SOAP Server'ın diğer tarafında da uygulamanın SQL Server ve İnternet kısımları yer alıyor. Böylelikle Şekil 3.3 bize uygulamanın tam bir resmini sunuyor. Kalan bölümlerde ise Şekil 3.3'dan aldığımız kesitlerle programların ve dolayısıyla web servis mimarisinin detayları uygulamanın farklı bölümleri olarak verilecek.



Şekil 3.3 Uygulama Program Şeması

3.2 Uygulamaya Giriş

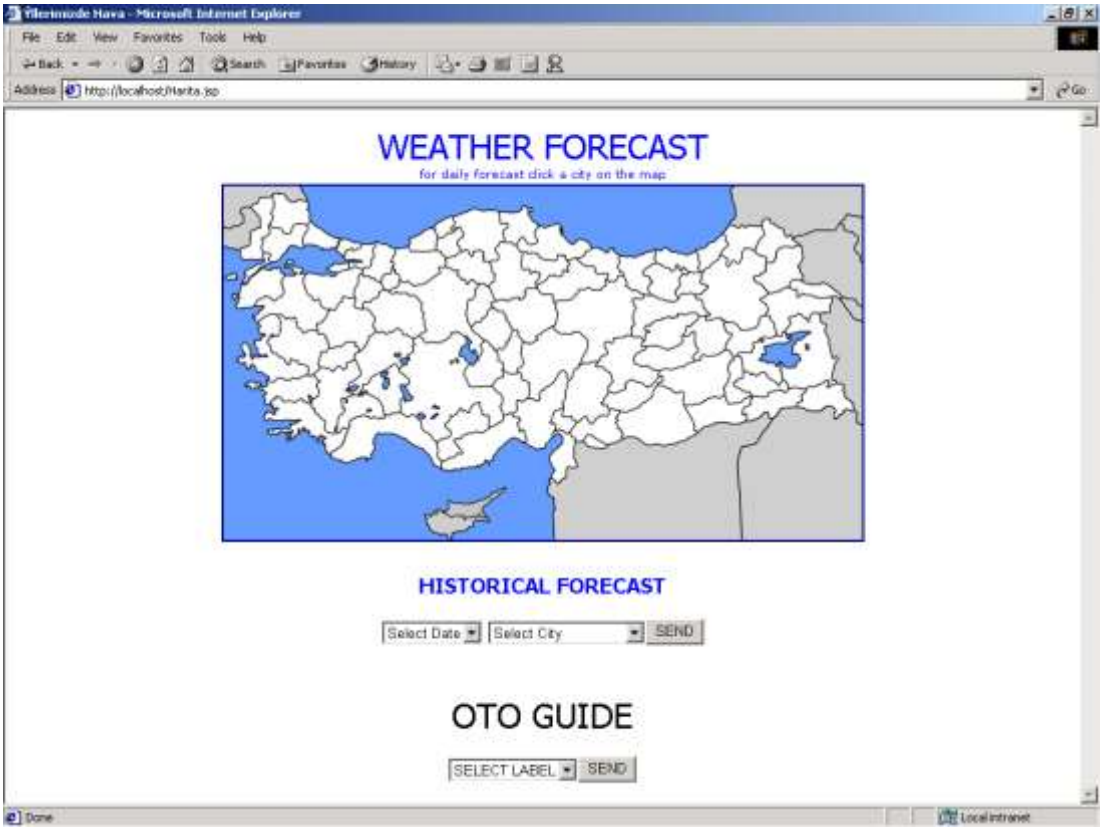
Kullanıcı sisteme Login.jsp sayfası ile giriş yapmaktadır. Bu sayfada kullanıcıdan Login ve Password verilerini girmesi istenmektedir. JSP kodu tarafından üretilen Login sayfası Şekil 3.4'de görüldüğü gibidir.

Şekil 3.4 Şifre Giriş Ekranı

Şekil 3.4'deki sayfayı oluşturan kod ana olarak aşağıdaki gibidir;

```
<FORM action="Harita.jsp" method="post">
<TABLE>
<TR><TD>Login : </TD><TD><INPUT type="text" name="login"></TD> </TR>
<TR><TD>Password: </TD><TD><INPUT type="password" name="password">
</TD></TR>
<TR><TD></TD><TD><INPUT type="submit" value="SEND"></TD></TR>
</TABLE>
</DIV>
</FORM>
```

Login ve Password değerleri SEND düğmesine basılmasıyla "post" metoduyla Harita.jsp (ekte cd'de) isimli JSP sayfasına geçirilmektedir. Burada girilen verilerin doğruluğu kontrol edilip yanlış veri girişi varsa program tekrar Login.jsp (ekte cd'de) sayfasına dönmektedir. Girilen değerlerin doğruluğu program tarafından onaylandıktan sonra kullanıcı Harita.jsp sayfasına ulaşabilmektedir. Uygulamanın ana sayfasını (Şekil 3.5'de görüldüğü gibi) Harita.jsp kodu oluşturmaktadır. Kullanıcının seçimine göre bu sayfadan program seçilen uygulamaya dallanmaktadır.



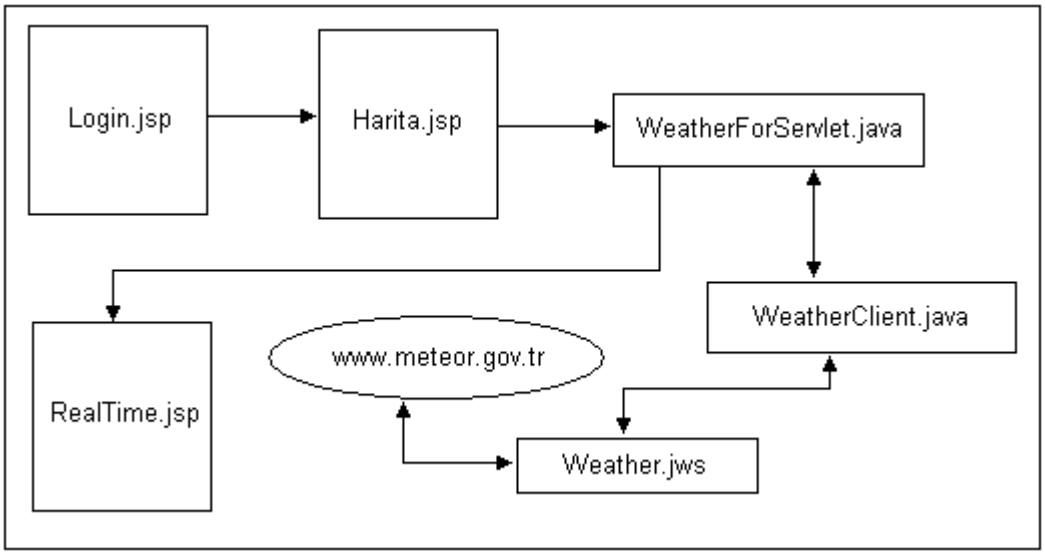
Şekil 3.5 Ana Ekran

Uygulama, Şekil 3.5'dede görüldüğü gibi iki bölüme ayrılmaktadır, uygulamanın ana bölümünü oluşturan WEATHER FORECAST (Hava Durumu) ve OTO GUIDE uygulaması olmak üzere.

3.2.1 Hava durumu uygulaması

Hava Durumu uygulaması kullanıcıya birkaç seçenek sunmakta dolayısıyla birkaç bölümden oluşmaktadır. Hava Durumu kullanıcıya geçmişe dönük istatiksel veri sağladığı gibi internet üzerinden anlık veride sağlayabilmektedir. Ayrıca sağlanan bu anlık veriler veritabanına kaydedilip daha sonra bir JSP sayfası aracılığıyla sorgulanabilmektedir.

İlk olarak uygulamanın internet üzerinden anlık veri sağladığı kısımla başlayacak olursak Şekil 3.6 hava durumu uygulamasının bu parçasında yer alan programlar arasındaki ilişkiyi ve sıralamayı vermektedir. Şekil 3.6 uygulamalarda bulunan tüm programların ilişkilerini gösteren Şekil 3.3'ün bir kesitidir.



Şekil 3.6 Dinamik Veri Şeması

Kullanıcı Şekil 3.5'de yer alan Türkiye haritası üzerinde herhangi bir şehri tıkladığı zaman o şehrin o anki havadurumu verilerine internet üzerinden ulaşılıp sistem üzerinde web servis mimarisi kullanılarak kullanıcıya ulaştırılmaktadır. Programın biraz detaylarına girecek olursak kullanıcı harita üzerinde bir şehri tıkladığı zaman Harita.jsp isimli JSP sayfası WeatherForServlet.java (ekte cd'de) koduna şehrin ismini geçirmektedir.

href="http://localhost/servlet/WeatherForServlet?name=SIVAS"

Bölüm 3.1'da bahsettiğimiz gibi servlet'lar web serverlar üzerinde çalışıp onların kapasitelerini artırmaktaydılar [4]. Dolayısıyla WeatherForServlet.java isimli servlet web server üzerinde kendisine gelecek çağrıları dinlemektedir. Yukarda verdiğimiz kod parçasıda bu servlet'a bir şehir ismi geçirerek onu çağırmaktadır. WeatherForServlet.java daha sonra kendisine geçirilen şehir parametresini alarak WeatherClient.java (ekte cd'de) koduna bu parametreyi geçirmektedir.

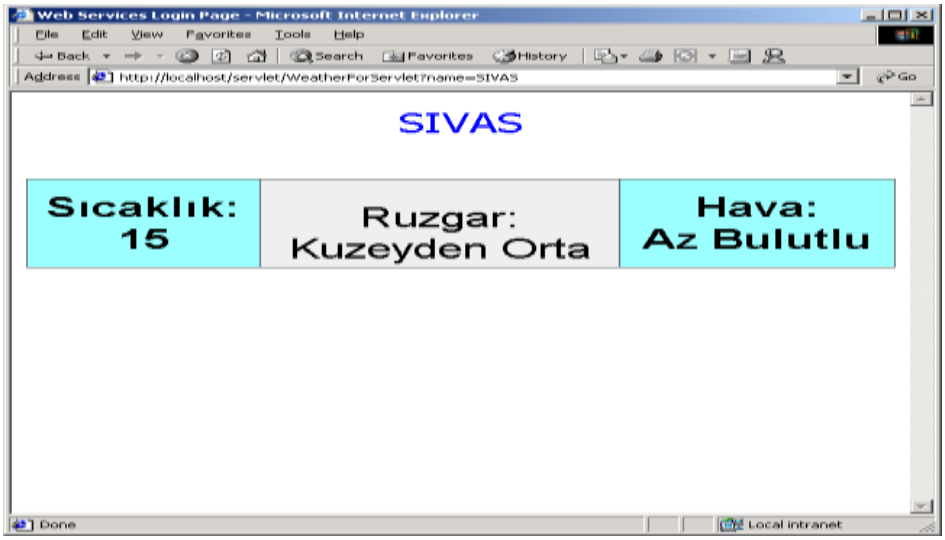
```
WeatherClient hndl = new WeatherClient();
data = hndl.getData(sehir);
```

WeatherForServlet.java şehrinismini SOAP client olarak görev yapan WeatherClient class'ının "getData" isimli metoduna geçirir. SOAP Server'a gönderilecek XML dokümanı WeatherClient.java kodu tarafından üretilmektedir. Program SOAP Server'a yollanacak dökümanı oluştururken parametre olarak şehrin ismini, Server üzerinde çalıştırılacak programın adresini ve metod ismini dökümana yerleştirir.

```
String endpoint = "http://localhost/axis/Weather.jws";

Service  service = new Service();
Call     call    = (Call) service.createCall();
```

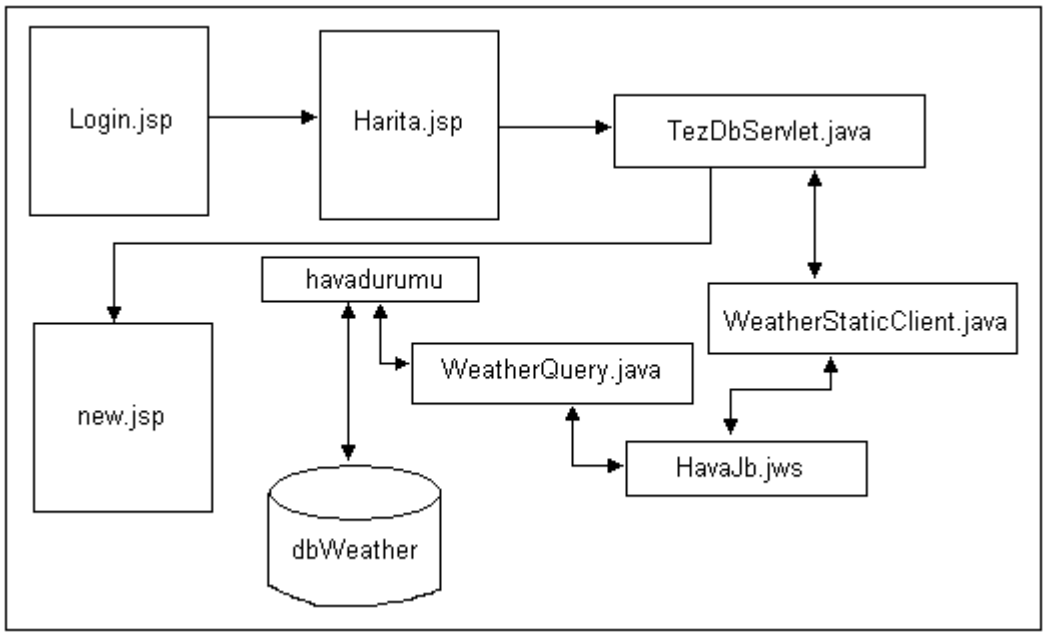
```
call.setTargetEndpointAddress( new java.net.URL(endpoint)
call.setOperationName( "getTemp" );
call.setProperty( Call.NAMESPACE, "Weather" );
call.addParameter( "arg1", XMLType.XSD_STRING,
    Call.PARAM_MODE_IN);
```

Yukarda WeatherClient class'ının kod parçasından anlaşıldığı gibi SOAP server'a yollanacak Bölüm 2.11.3'de anlattığımız SOAP envelope'u program oluşturur. WeatherClient.java XML dökümanını (SOAP envelope) SOAP server'a yollar. SOAP server XML dökumanını alıp açtıktan sonra artık ne yapması gerektiğini biliyordur çünkü zarfın içinde network üzerinde nereye gitmesi gerektiği, hangi metodu çağırması gerektiği ve bu metoda hangi değeri geçirmesi gerektiğine kadar gerekli herşey vardır. Weather.jws (jws: java web service) SOAP server üzerinde çalışan bir class'dır. Axis (SOAP server) bu class'ın "getTemp" metoduna şehir ismini taşıyan parametreyi geçirir. Bu metod internet üzerinde devlet meteoloji işlerinin sitesine bağlanarak kendisine yollanan şehri, sitenin HTML kodunu tarayarak bulur ve şehirli ilgili tümhava durumu bilgilerini parse ederek bir string'e alır. Elde edilen veriler tekrar SOAP server üzerinden XML dökümanı (SOAP envelope) olarak SOAP client'a (WeatherClient) döndürülür. WeatherClient class'ı kendisine gelen diziyi çağırılmış olduğu WeatherForServlet'a geri döndürür, servlet Şekil 3.7 görülen RealTime.jsp isimli JSP sayfasını çağırır.



Şekil 3.7 Dinamik Çıktı Ekranı

Uygulamanın Historical Forecast bölümünde ise kullanıcıya istatiksel hava durumu verileri sağlanmaktadır. Program web servis mimarisi üzerinden server'a ulaşıp veritabanındaki verileri SOAP server üzerinden kullanıcıya döndürmektedir. Şekil 3.8'de Şekil 3.3'deki yapının bir kesitini sunmaktadır.



Şekil 3.8 Statik Veri Şeması

Şekil 3.5'deki sayfada kullanıcının seçtiği tarih ve şehir "get" metoduyla [9] TezDbServlet (ekte cd'de) isimli servlet'a geçirilir. Servlet aldığı parametreleri (tarih ve şehir) WeatherStaticClient.java (ekte cd'de) isimli SOAP client'a geçirir.

```
WeatherStaticClient hndl = new WeatherStaticClient();
param = hndl.getStaticData(seh,day);
```

WeatherStaticClient aşağıdaki kod parçası ile SOAP envelope'u oluşturur (XML dökümanı) ve SOAP server'a yollar.

```
String endpoint = "http://localhost/axis/HavaJb.jws";

Service  service = new Service();
Call     call    = (Call) service.createCall();

call.setTargetEndpointAddress( new java.net.URL(endpoint)
call.setOperationName( "getForecast" );
call.setProperty( Call.NAMESPACE, "HavaJb" );
call.addParameter( "arg1", XMLType.XSD_STRING,
     Call.PARAM_MODE_IN);

String ret = (String) call.invoke( new Object[] {var} );
```

SOAP server zarfı açarak network üzerinde "http://localhost/axis/HavaJb.jws" adresindeki HavaJb (ekte cd'de) class'ının "getForecast" metodunu "arg1" isimli parametreyi geçirerek çağırmaktadır. "getForecast" aşağıda verilen satırlar ile kendisine gelen parametreyi WeatherQuery.java isimli JDBC [9] koduna geçirerek çağırır.

```
WeatherQuery hndl = new WeatherQuery();
path = hndl.getData(seh);
```

WeatherQuery (ekte cd'de) class'ı veritabanında aradığı verilere ulaşabilmek için ODBC [5] yardımıyla SQL server'a bağlanır ve aradığı dataları bulabilmek için elindeki parametreleri "havadurumu" isimli stored procedure'a [10] geçirir. WeatherQuery class'ı tüm bu anlatılanları sırasıyla aşağıdaki kod yardımıyla gerçekleştirir.

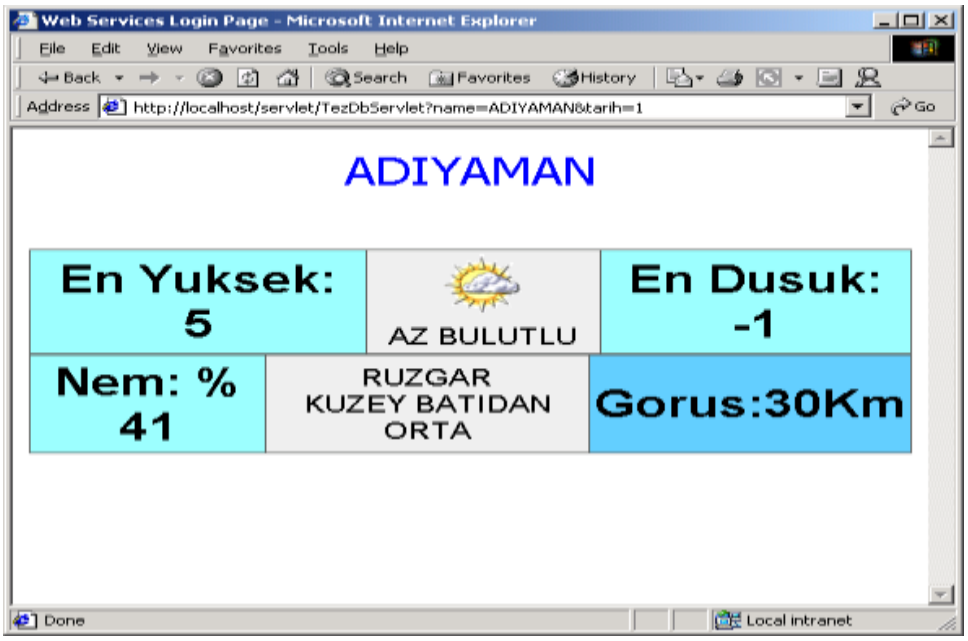
```
String que = new String("havadurumu @sehir=" + seh);
String query = new String(que + ", @tarih=" +day);

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
conn = DriverManager.getConnection("jdbc:odbc:weather");
Statement stmt = conn.createStatement();
ResultSet results = stmt.executeQuery(query);
```

WeatherQuery yukarıdaki kod ile gerekli parametreleri "havadurumu" (ekte cd'de) isimli stored procedure'a geçerir. "havadurumu" aldığı veriler ile SQL server üzerinde bulunan dbWeather isimli veritabanını sorgular.

```
CREATE PROCEDURE havadurumu  @sehir text,  @arih int
 AS

SELECT HV.TARIH, S.SEHIR_DESC, R.RUZGAR_DESC, H.HAVA_DESC,
        HV.EN_YUKSEK_SIC, HV.EN_DUSUK_SIC, HV.NEM,
        HV.HISSEDILEN, HV.GORUS, H.HAVA_CODE

FROM HAVA AS H, HAVA_DURUMU AS HV, RUZGAR AS R, SEHIR AS S

WHERE HV.HAVA_CODE = H.HAVA_CODE
        AND HV.RUZGAR_CODE = R.RUZGAR_CODE
        AND HV.SEHIR_CODE = S.SEHIR_CODE
        AND S.SEHIR_DESC like @sehir
        AND HV.TARIH = @arih;
```

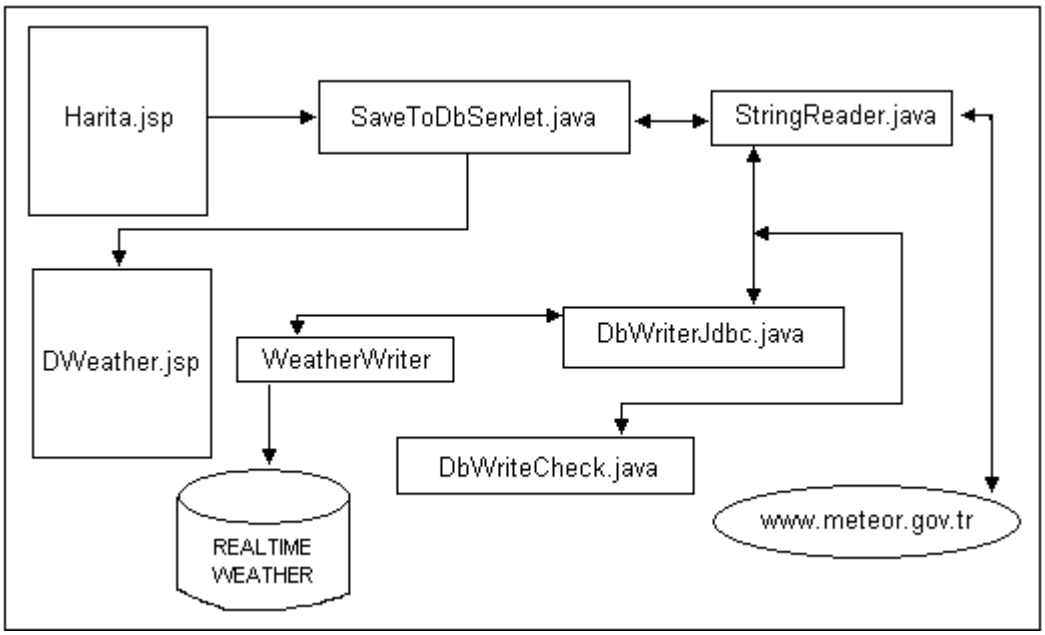
Sorgulama sonucu elde edilen veriler WeatherQuery tarafından alınır ve tekrar SOAP server üzerinden TezDbServlet'a döndürülür, servlet kendisine gelen verileri newjsp (ekte cd'de) isimli JSP sayfasına iletir. JSP gelen verilerin tipine göre yani

havanın açık, yağmurlu, karlı oluşu gibi durumlara göre ekrana basacağı resimleri belirledikten sonra kodu yine dinamik olarak aşağıda görüldüğü gibi oluşturur.



Şekil 3.9 Statik Veri Çıkış Ekranı

Kullanıcı bir başka şeçenekte isterse Şekil 3.5'deki harita üzerinde yeralan tüm şehirlerin o anki hava durumu verilerini "for daily forecast click a city on the map" yazısına tıklayarak veritabanına kaydedebilir ve daha sonra da Şekil 3.11'de görülen JSP sayfasını kullanarak herhangi bir tarihte kaydetmiş olduğu verilere ulaşabilir.



Şekil 3.10 Dinamik Veri Kayıt Şeması

Kullanıcının Şekil 3.5'deki "for daily forecast dick a city on the map" yazısını tıklamasıyla program SaveToDbServlet (ekte cd'de) isimli servlet'a dallanır. Bu servlet StringReader (ekte cd'de) isimli class'ın getData isimli metoduna çağrı mda bulunur.

```
db.StringReader hndl = new db.StringReader();
page = hndl.getData();
```

StringReader class'ı internet sitesine bağlanarak şehirlerin hava durumunun bulunduğu tablonun HTML kodunu string tipinde bir değişkene atar.

```
URL url = new URL("http://www.meteor.gov.tr/pages/sonhatrt.htm");
InputStream is = url.openStream();
BufferedReader nis = new BufferedReader(new InputStreamReader(is));
```

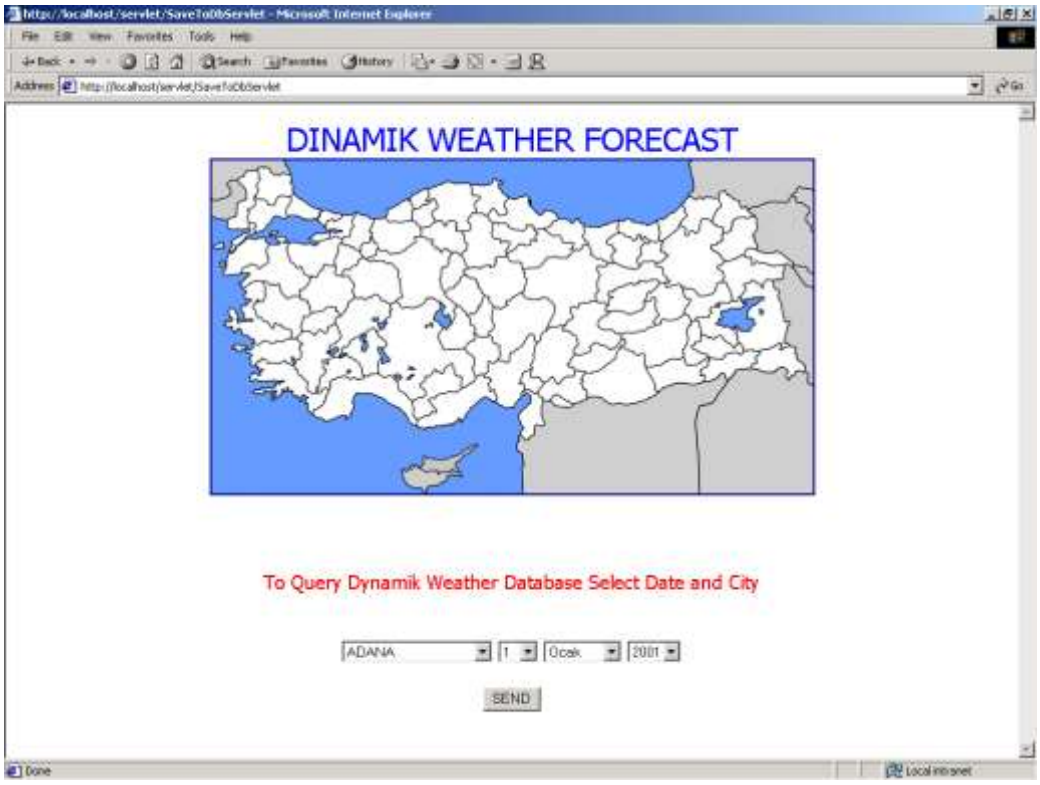
Artık tüm şehirlerin hava durumu verileri elde edilmiştir fakat veriler HTML koduna gömülü haldedir. Bu verilerin kod içinden ayıklanması gerekmektedir. SaveToDbServlet class'ına dönen program bu defa verilerin atandığı değişkeni PageParser (ekte cd'de) isimli class'ın "parser" metoduna geçirerek buraya yönlenir. Bu class'da HTML kodu içinden her şehrin verilerinin ayrılması işlemi gerçekleştirilir. Tam burada Bölüm 2'de verdiğimiz web servis mimarisinin üstünlüğü ortaya çıkmaktadır. Eğer bu veriler bu mimariyle XML formatında sunuluyor olsaydı tüm bu fazladan proglamlamaya gerek kalmayacaktı. Veriler veritabanına kaydedilmeden önce DbWriterCheck (ekte cd'de) class'ı kaydedilmeye başlayacak olan verilerin veritabanında olup olmadığını kontrol eder. Eğer bu veriler mevcutsa kayıt işlemine geçilmez ve programın icrası durdurulur. Veriler kayıtlı değilse DbWriterJdbc (ekte cd'de) class'ı her şehrin verilerini tek tek veritabanına kayıt etmeye başlar.

```
query = "WeatherWriter @cit=\"" + val[1] + "\",@dat=\"" +
      val[2] + "\",@wind=\"" + val[4] + "\",@sight=\""
      + val[5] + "\",@weather=\"" + val[6] +
      "\",@tempetuar=\"" + val[7] + "\",@humidity=\""
      + val[8] + "\",@sistemzamanı=\"" + tarih + "\"";

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection conn =
            DriverManager.getConnection("jdbc:odbc:weather");

PreparedStatement stmt = conn.prepareStatement(query);

stmt.executeUpdate();
```

Kayıt işlemi için ODBC üzerinden veritabanına bağlanarak aşağıda verilen "Weather Writer" isimli stored procedure tarafından alınan değerler dbWeather isimli veritabanına kaydedilir.

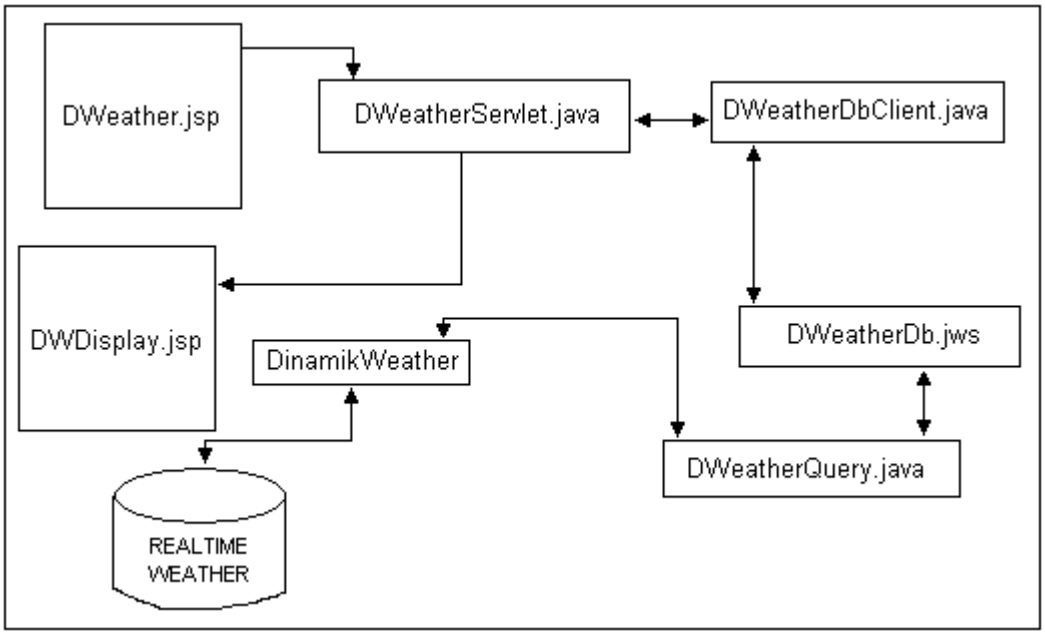
```
CREATE PROCEDURE WeatherWriter @city text, @date text, @wind text, @sight text,
@weather text, @tempetuar text, @humidity text, @sistemzamanı text
 AS

INSERT INTO REALTIMEWEATHER (CITY, TARIH, WIND, SIGHT, WEATHER,
TEMPETUAR, HUMIDITY, SISTEMTIME)
VALUES (@city, @date, @wind, @sight, @weather, @tempetuar, @humidity, @sistemzamanı)
```

DbWriterJdbc Class'ı her şehir için aynı işlemleri stored procedure'ı çağırarak yapar ve her şehir veritabanına tek tek kaydedilir. Programın başarıyla tamamlanmasından sonra program Şekil 3.11'de görülen Dweather.jsp (ekte cd'de) isimli JSP sayfasına yönlendirilir. Kullanıcı bu sayfadan o gün veya daha öncede kaydetmiş olduğu verileri sorgulayabilir.



Şekil 3.11 Dinamik Sorgulama Ekranı

Kullanıcı Şekil 3.11'de görülen JSP sayfasında kaydettiği herhangi bir günün tarihini ve şehir ismini seçerek o güne ait verileri görüntüleyebilir. Programın icrası yine web servis mimarisi içinde gerçekleşir.



Şekil 3.12 Dinamik Sorgu Şeması

Kullanıcının Şekil 3.11'de görülen JSP sayfasında tarih ve şehir seçmesiyle seçilen bu değerler "get" metoduyla [1] DWeatherServlet.java (ekte cd'de) isimli servlet'a aktarılır. Bu servlet aldığı bu değerleri DWeatherDbClient (ekte cd'de) isimli class'a geçirir.

```
DWeatherDbClient hndl = new DWeatherDbClient();
data = hndl.getData(sehir,tarih);
```

DWeatherDbClient aldığı bu parametreleri SOAP server'a göndermek üzere hazırladığı SOAP envelope'a yerleştirir. DWeatherDbClient bu şekilde Bölüm 2.11'de anlatıldığı gibi SOAP client olarak görev yapar.

```
String endpoint = "http://localhost/axis/DWeatherDb.jws";

Service  service = new Service();
Call     call    = (Call) service.createCall();

call.setTargetEndpointAddress( new java.net.URL(endpoint) );
call.setOperationName( "getData" );
call.setProperty( Call.NAMESPACE, "DWeatherDb" );
call.addParameter( "arg1", XMLType.XSD_STRING,
     Call.PARAM_MODE_IN);

Integer ret = (Integer) call.invoke( new Object[] { city } );
```

DWeatherDbClient bu şekilde oluşturduğu XML dökümanını SOAP server'a yollar. SOAP server gelen XML dökümanını Şekil 2.4'de gösterildiği gibi parse ederek gerekli parametreleri elde eder. Artık "http://localhost/Axis/DWeatherDb.jws " adresinde

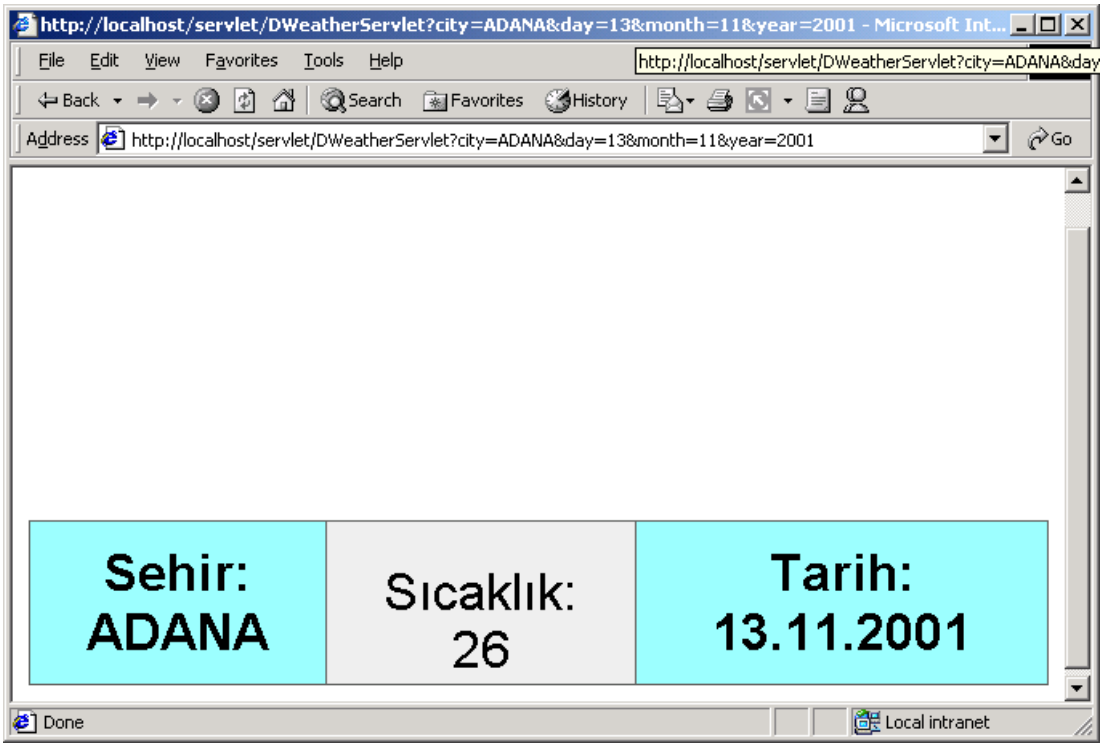
"getData" metoduna geçireceği parametreyi biliyordur. DWeatherDb.jws DWeatherQuery class'ına (ekte cd'de) parametreleri geçirerek çağrıda bulunur.

```
DWeatherQuery hndl = new DWeatherQuery();
t = hndl.dyQuery(seh,tar);
```

DWeatherQuery class'ı JDBC ve ODBC kullanarak veritabanına bağlanır. Kendisine gelen parametreleride veritabanında bulunan "DinamikWeather" isimli stored procedure'a geçirir. DinamikWeather kendisine gönderilen değerlerle veritabanını sorgulayarak kullanıcının istediği değerleri elde eder.

```
CREATE PROCEDURE DinamikWeather  @sehir text, @tarih text
AS
select rlt.TEMPETUAR
from REALTIMEWEATHER as rlt
where  rlt.CITY like @sehir
       and rlt.SISTEMTIME like @tarih;
```

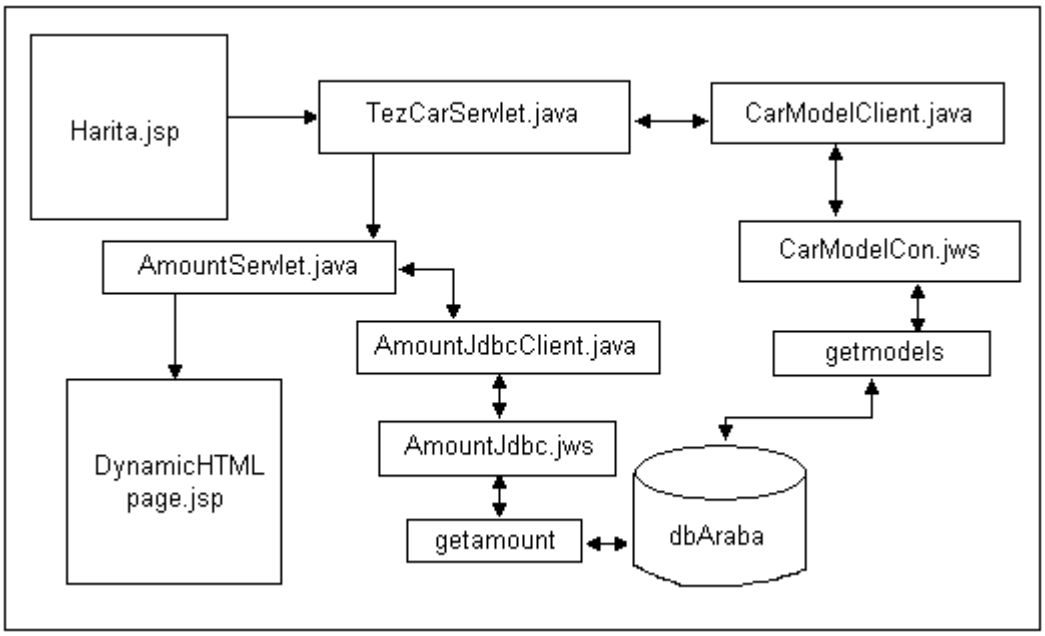
Sorgu sonucu elde edilen değerler tekrar aynı şekilde SOAP server üzerinden geri DWeatherServlet'a kadar döndürülür. Servlet bu değerleri DWDisplay.jsp (ekte cd'de) isimli JSP sayfasına geçirir. DWDisplay aldığı değerleri ekrana dinamik olarak HTML kodunu üreterek Şekil 3.13'de görüldüğü gibi aktarır.



Şekil 3.13 Dinamik Sorgu Çıkış Ekranı

3.2.2 Oto guide uygulaması

Uygulamanın ikinci kısmı olan Oto Guide kullanıcının seçimine göre veritabanında tutulan verileri statik verileri hava durumunda olduğu gibi yine web servis mimarisi içinde kullanıcıya ulaştırır. Biraz detaylarına bakacak olursak;



Şekil 3.14 Oto Guide Program Şeması

Kullanıcının Şekil 3.5'de görülen ana ekranda Oto Guide bölümünde herhangi bir marka seçip SEND düğmesine basmasıyla Harita.jsp (ekte cd'de) isimli JSP sayfası seçilen markayı "get" metoduyla TezCarServlet isimli servlet'a yollar.

```
CarModelClient hndl = new CarModelClient();
models = hndl.getModel(araba);
```

TezCarServlet aldığı bu değeri CarModelClient isimli class'a geçirerek onu çağırır. CarModelClient, SOAP envelope'u oluşturan bir SOAP client olarak görev yapmaktadır. CarModelClient SOAP server'a yollanacak XML dökümanını oluşturur.

```
String endpoint = "http://localhost/axis/CarModelCon.jws";
Service  service = new Service();
Call     call    = (Call) service.createCall();

call.setTargetEndpointAddress( new java.net.URL(endpoint) );
call.setOperationName( "getModels" );
call.setProperty( Call.NAMESPACE, "CarModelCon" );
call.addParameter( "arg1", XMLType.XSD_STRING,
            Call.PARAM_MODE_IN);
```

SOAP server gelen zarfı (SOAP envelope) açarak network üzerinde hangi class'ı, bu class'ın hangi metodunu hangi gerekli parametrelerle çağıracağını elde eder, Bölüm 2.11'de anlatıldığı gibi. Burada CarModelCon.jws SOAP server üzerinden çağrılmakta ve "getModels" isimli metodunada servlet tarafından yollanan parametre geçirilmektedir. Bu metod direkt veritabanına bağlanarak kullanıcının istediği marka modellerine ulaşmaktadır.

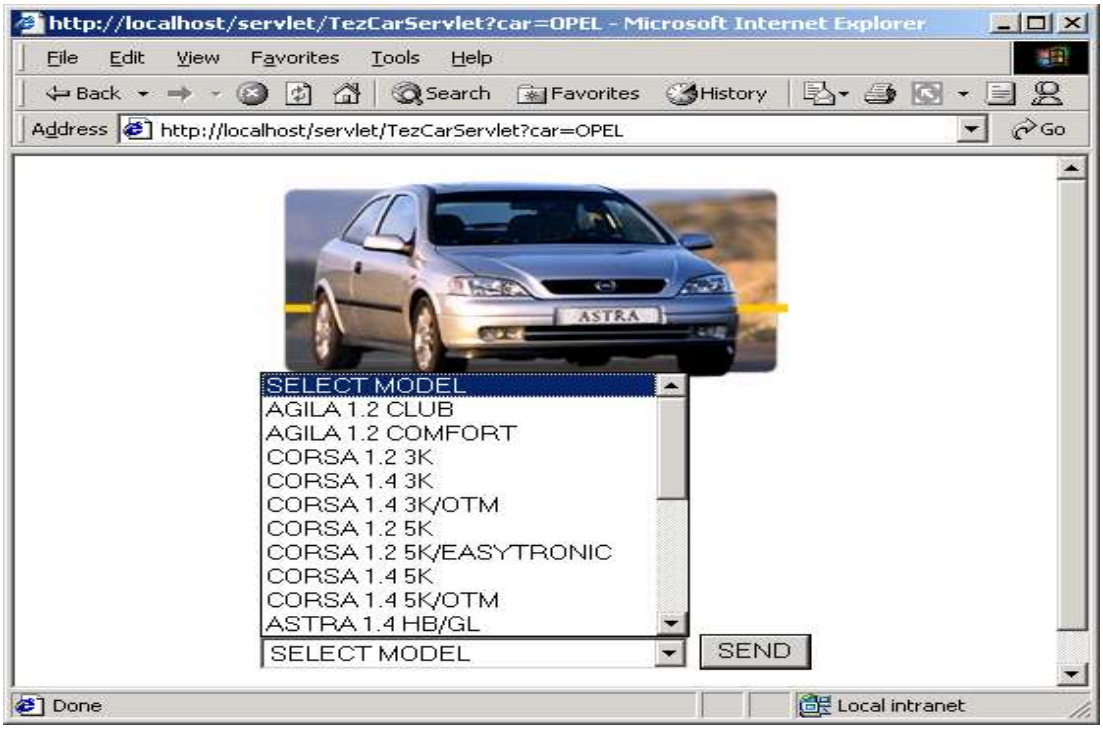
```
String query = new String("getmodels @model="+araba);
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
conn = DriverManager.getConnection("jdbc:odbc:car");
```

Burada JDBC ve ODBC driver'larıyla veritabanına bağlanılarak "getmodels" isimli stored procedure çağrılmaktadır. "getmodels" veritabanını kendisine gelen değerlere göre sorgulayarak elde ettiği değerleri tekrar kendisini çağıran CarModelCon class'ına iletir. CarModelCon aldığı bu değerleri tekrar SOAP server üzerinden kendisini çağıran CarModelClient'a iletir. Geriye döndürülen bu SOAP envelope şu SOAP server tarafından aşağıdaki gibi oluşturulur.

```
HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Content-Length: 647

Date: Tue, 11 Dec 2001 08:45:09 GMT

Server: Apache Tomcat/4.0.1 (HTTP/1.1 Connector)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAP-ENV:Body>
  <ns1:getModelsResponse xmlns:ns1="CarModelCon">
   <getModelsResult xsi:type="xsd:string">MATIZ SE/KLM!LANOS 1.5
SE/4K!LANOS 1.5 SE/5K!NUBIRA 1.6 SX/4K!NUBIRA 1.6 SX/4K/OTM!LEGANZA
CDX/4K!LEGANZA CDX/OTM!CHAIRMAN 600 SX!KORANDO 2.3!KORANDO 2.9
TD!MUSSO 2.9 TD!</getModelsResult>
  </ns1:getModelsResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

CarModelClient kendine döndürülen string tipindeki veriyi alarak TezCarServlet'ailetir. Bu servlet kendisine gelen değerleri bir JSP koduna yollamadan dinamik olarak kendisini Şekil 3.15'de görülen sayfayı oluşturur.



Şekil 3.15 Otto Guide Seçim Ekranı

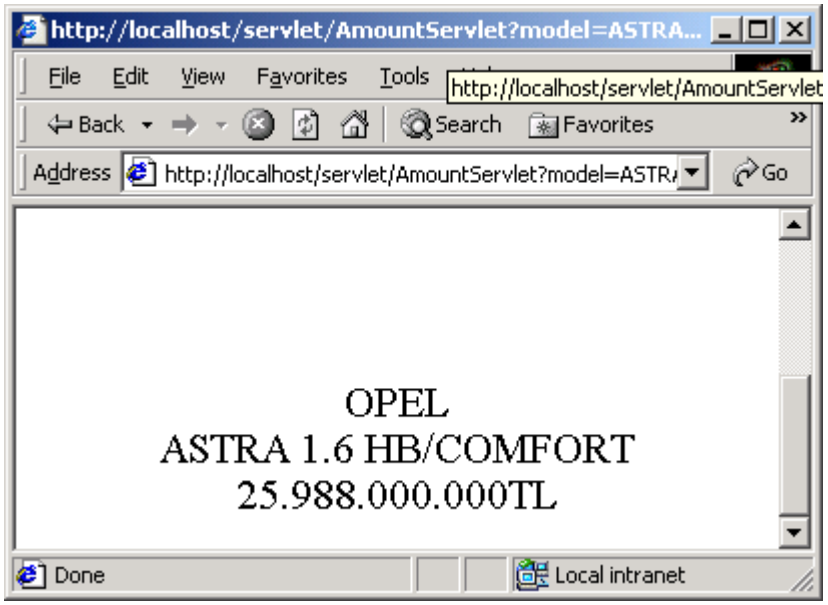
Kullanıcının herhangi bir model seçip yollamasıyla AmountServlet.javaisimli servlet bu değeri alır. Bu servlet SOAP envelope'u üretmesi için AmountJdbcClient isimli class'ı çağırır. AmountJdbcClient XML kodunu üreterek SOAP server'a gönderir.

```
String endpoint = "http://localhost/axis/AmountJdbc.jws";

Service  service = new Service();
Call     call    = (Call) service.createCall();

call.setTargetEndpointAddress( new java.net.URL(endpoint) );
call.setOperationName( "getAmount" );
call.setProperty( Call.NAMESPACE, "AmountJdbc" );
call.addParameter( "arg1", XMLType.XSD_STRING,
       Call.PARAM_MODE_IN);

String ret = (String) call.invoke( new Object[] { car } );
```

AmountJdbcClient SOAP server'a gönderdiği XML dökümanı içine metod çağırımı ile ilgili tüm verileri yerleştirmektedir. Buna göre SOAP server üzerinden AmountJdbc.jws'in "getAmount" isimli metodu çağrılır ve gerekli olan parametre "arg1" diye geçirilir. "getAmount" veritabanına bağlanarak gerekli sorgulamaları yapması için bir stored procedure çağırır ve gerekli değişkenleri onailetir.

```
String query = new String("getamount @model=\"" + model + "\"");

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn = DriverManager.getConnection("jdbc:odbc:car");
Statement stmt = conn.createStatement();
ResultSet results = stmt.executeQuery(query);
```

Çağrılan "getamount" isimli prosedür gerekli sorgulamayı yapıp elde ettiği değerleri AmountJdbc.jws'e döndürür. Bu değerler tekrar SOAP server üzerinden geri AmountServlet'a kadar iletilir. AmountServlet bu değerleri DynamicHTMLPage.jsp isimli JSP sayfasına geçirerek Şekil 3.16'da görüldüğü gibi sergilenmesini sağlar.



Şekil 3.16 Otto Guide Çıkış Ekranı

4. SONUÇLAR VE ÖNERİLER

Tezde web servis mimarisinin genel ve teknik yapısı ele alınmış ve seçilen uygulamalarla örneklendirilmiştir. Çalışmada web servis mimarisi ile geliştirilen uygulamalar mimarinin sağladığı avantajlar ile karşımıza hem B2B hem de B2C uygulamalarının sağladığı avantajlarla çıkmaktadır. Kullanıcıya servisin sağladığı hizmet bakımından bir B2C uygulamasının diğer taraftan web servis mimarisinin diğer programlamlar ve servislerle entegrasyona izin vermesi sonucu bir B2B uygulamasının özelliklerini yansıtmaktadır. Uygulamada kullanıcı, internet, veritabanı modülleri birbirleriyle dinamik olarak etkileşen bir yapı içerisinde yer almaktadır. Ayrıca bu modüllerin birbirlerinden bağımsız dizayn edilip geliştirilmeleri sonucu uygulamanın etkinliğini artırılmış, bakımı kolaylaşmış ve ileride eklenebilecek modüller, programlar için avantajlı bir yapı oluşturulmuştur. Web servis mimarisinin uygulamanın temelinde yer alması onu şimdiye kadar bildiğimiz klasik uygulamaların ötesinde bir yere yerleştirmiştir. Uygulamalar local olarak çalışmalarına rağmen UDDI mekanizması Bölüm 3'de de anlatıldığı üzere IBM Test Registry'ye kaydedilmiştir. Böylece bunun gibi bir servise ihtiyaç duyan her kullanıcı veya program servisle tam bir etkileşim içine girebilecektir. Hatta UDDI mekanizmasını kullanarak bu servisi bulan ve yararlanmak isteyen başka servislerde direkt etkileşimde bulunulabilecektir.

Uygulama ilk bakışta havadurumu ve araba bilgileri sağlayan bir internet sitesi gibi görülebilir fakat ilk bölümde bahsettiğimiz akıllı sistemler için kolay ulaşılabilir veriler sunmaktadır. Varolan uygulamanın tek farkı bu verileri bir akıllı sisteme değilde web browser kullanarak sisteme bağlanan bir kullanıcıya sunmasıdır. Uygulama mimarisinden dolayı bu bilgileri başka bir programada kolaylıkla sağlayabilir. Bu bir başka web servisi de olabilir. Örneğin 1. Bölüm'de bahsettiğimiz senaryo içinde bulunan GSM operatörü böyle bir servisten istemde bulunabilir.

Yakın bir gelecekte teknik olarak yazılım ve donanımdan oluşan web servislerinin birbirlerini içeriklerine göre bulmaları, isteklerini anlamaları, birbirleriyle

uzlaşmaları ve karşılıklı isteklerini karşılamaları mümkün olacaktır. Sonuç olarak web servisleri web'in yeni evrimi olarak karşımıza çıkmaktadır. Web servisleri tüm bu senaryoların gerçekleşebilmesi için gerekli model ve altyapı sağlamaktadır. Dolayısıyla yapılan bu çalışmada geleceğin akıllı sistemlerine model olabilecek bir yapıya yer verilmiş, uygulamalar buna uygun olarak geliştirilmiş ve bu konular üzerinde bundan sonra yapılacak çalışmalara ışık tutabilme amacı güdülmüştür.

KAYNAKLAR

[1] **Graham, S., Simeonov, S., Boubez, T., Davis, D., Daniels, G., Nakamura, Y. and Neyama, R.,** 2002. Building Web Services with Java Making Sense of XML, SOAP, WSDL and UDDI, SAMS Publishing, USA

[2] **Kreger, H.,** 2001. Web Services Conceptual Architecture, IBM Software Group.

[3] **Christensen, E., Curbera, F., Meredith, G., Weerawarano, S.,** 2001. Web Services Description Language, IBM Software Group.

[4] **Brittenham, P., Curbera, F., Ebnebuske, D. and Graham, S.,** 2001. Understanding WSDL in a UDDI Registry, IBM Software Group.

[5] **Eckel, B.,** 2000. Thinking in Java, Prentice Hall, USA

[6] **Watson, M.,** 2002. Sun ONE Services, mandtbooks, USA

[7] **Brittenham, P.,** 2001. Web Services Development Concepts, IBM Software Group.

[8] **Holzschlag, M., E.,** 2000. HTML 4, QUE, USA

[9] **Wutka, M.,** 2002. JAVA 2 Enterprise Edition, QUE, USA

[10] **Wynkoop, S.,** 1999. Special Edition Using Microsoft SQL Server 7.0, QUE, USA

ÖZGEÇMİŞ

Şahin Kekevi 1974 yılında İstanbul'da doğdu. İlk ve orta öğrenimini Malatya Gazi ilkokulu ve Atatürk Orta Okulunda tamamladı. Lise öğrenimine 1989 yılından itibaren Kadıköy Anadolu Lisesinde devam etti. 1992 yılında İTÜ Matematik Mühendisliği Bölümüne girdi 1997 yılında mezun olduktan sonra aynı yıl İTÜ Mühendislik Ana Bilim Dalı Sistem Analizi Programında Yüksek Lisans öğrenimine başladı.