# RESTFUL WEB SERVİSLERİ İLE ONTOLOJİ SORGULAMA

**ABDULHAMİT MABOÇOĞLU**

**YÜKSEK LİSANS TEZİ**
**BİLGİSAYAR MÜHENDİSLİĞİ**

**TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

**EYLÜL 2010**
**ANKARA**

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Ünver KAYNAK
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Doç. Dr. Erdoğan DOĞDU
Anabilim Dalı Başkanı

Abdulhamit MABOÇOĞLU tarafından hazırlanan RESTFUL WEB SERVİSLERİ İLE ONTOLOJİ SORGULAMA adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Doç. Dr. Erdoğan DOĞDU
Tez Danışmanı

Tez Jüri Üyeleri

Başkan :Doç. Dr. Erdoğan DOĞDU _____

Üye : Doç. Dr. Bülent TAVLI _____

Üye : Yrd. Doç. Dr. Hüsrev Taha SENCAR _____

## TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Abdulhamit MABOÇOĞLU

**Üniversitesi** : **TOBB Ekonomi ve Teknoloji Üniversitesi**
**Enstitüsü** : **Fen Bilimleri**
**Anabilim Dalı** : **Bilgisayar Mühendisliği**
**Tez Danışmanı** : **Doç. Dr. Erdoğan DOĞDU**
**Tez Türü ve Tarihi** : **Yüksek Lisans – Eylül 2010**

**Abdulhamit MABOÇOĞLU**

**RESTFUL WEB SERVİSLERİ İLE ONTOLOJİ SORGULAMA**

## ÖZET

Semantik veya Anlamsal Web, günümüzde web alanında yaşanan en önemli gelişmelerden birisidir. Web'de anlamsız denen türde bilgiler arama motorlarınca kelime bazlı arama ve web belgeleri arasındaki anlamsız bağlantılar takip edilerek bulunabilmektedir. Bunun yetersizliği başta fark edilmiş ve geliştirilen semantik web teknolojileri bu boşluğu dolduracak görünmektedir.

Semantik web yalnızca web tabanlı sistemler için değil diğer bilgi sistemleri için de önemli bir veri modeli ve bilgi işleme alternatifi oluşturmaktadır. Semantik Web'de kapsamlı veri modelleri ontolojilerde oluşturulmaktadır. Ontoloji belli bir alanda kullanılan veri türleri ve bunlar arasında ilişkileri tanımlayan bir yapıdır. Ontoloji tanımlama dilleri RDF, RDFS, OWL bu konuda geliştirilmiş web standartları olarak akademi ve endüstride kabul görmektedir.

Ontoloji, semantik web tabanlı sistemler için bir tür şema işlevi görmektedir. Ontolojiler bu iş için geliştirilmiş özel araçlar kullanılarak oluşturulabilmektedir ve sorgulanabilmektedirler. Bu tezde ontoloji tanım ve verilerinin sorgulanması ve oluşturulması için geliştirdiğimiz RESTful web servisleri tabanlı bir protokol ve web-tabanlı gerçekleştirimi sunulacaktır.

**Anahtar Kelimeler:** semantik web, anlamsal web, restful, web servisleri, ontoloji, owl, rdf

**University** : **TOBB University of Economics and Technology**

**Institute** : **Institute of Natural and Applied Sciences**
**Science Programme** : **Computer Engineering**
**Supervisor** : **Associate Professor Dr. Erdoğan DOĞDU**
**Degree Awarded and Date** : **M.Sc. – September 2010**

**Abdulhamit MABOÇOĞLU**

**QUERYING ONTOLOGIES USING RESTFUL WEB SERVICES**

## ABSTRACT

Semantic web is the most important technology development nowadays in web technologies area. In the web, information is extracted from the current web via keyword-based search and following the meaningless links between the documents using web search engines. It is noticed that this will be insufficient at the very beginning with the ever growing web and semantic web has the promise to fix this.

Semantic web provides an important data model and information processing alternative not only for the web-based but for all other information processing systems as well. In the semantic web, comprehensive data models are constructed in the ontologies. Ontology is a data definition model for the data types and relationships in a specific domain. Ontology definition languages RDF, RDFS, OWL are the current web standards that are accepted in the academia and the industry.

Ontology constitutes a schema for the semantic web-based systems. Ontologies are constructed and queried using specialized tools developed for his purpose. In this thesis we present a RESTful web services-based protocol and its implementation that is developed to define and query ontology definition and data.

**Keywords:** semantic web, restful, web services, ontology, owl, rdf

## TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren hocam Doç. Dr. Erdoğan DOĞDU'ya yine kıymetli tecrübelerinden faydalandığım arkadaşım Abdullah BATTAL'a teşekkürü bir borç bilirim.

## İÇİNDEKİLER

# ÇİZELGELERİN LİSTESİ

# ŞEKİLLERİN LİSTESİ

## KISALTMALAR

| Kısaltmalar | Açıklama |
|---|---|
| **API** | Application Programming Interface |
| **ASCII** | American Standard Code for Information Interchange |
| **HTML** | Hyper Text Markup Language |
| **OWL** | Web Ontology Language |
| **RDBMS** | Relational Database Management System |
| **RDF** | Resource Description Framework |
| **RDFS** | RDF Schema |
| **REST** | Representational State Transfer |
| **SPARQL** | Sparql Protocol And RDF Query Language |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **WSDL** | Web Service Description Language |
| **XML** | Extensible Markup Language |
| **XSL** | Extensible Stylesheet Language Family |

# 1. GİRİŞ

Internet ortamında bilgiler insan odaklı olarak oluşturulmuştur. 1989 yılında Tim Berners Lee tarafından geliştirilen HTML dili bu amaca uygun şekilde tasarlanmıştır [1]. Günümüzde 5. Sürümü yayınlanan bu dil, görsel tasarım ve metin tabanlı içerik sunmaktadır. Bir bilginin ilişkili olduğu başka bir bilgiye erişilmek istendiğinde insan faktörünün devreye girmesi gerekmektedir. Her ne kadar standart bir yapıya sahip olsa da, HTML sonuçta durağan yapıda bir dildir ve bağlantıların tanımlanmasında çok zayıf kalır.

Metin ya da görsel tabanlı her bilgi aslında web ortamında bir kaynaktır. Ve bu kaynağın verimli bir şekilde tutulması gerekmektedir. Tutarlı ve güncel bir kaynağa sahip olmak için bunların belirli standartlarda tutulması gerekmektedir. Bunun yanında kaynakların insanların dışında bilgisayar yazılımlarının da anlayabileceği bir yapıda olması, aranan bir kaynağın ya da bilginin daha hızlı bulunmasını sağlar. Aynı zamanda bir bilgi kaynağının paylaşılmasına da imkân verir. Bu amaçla Semantik Web kavramı ortaya çıkmıştır.

Bilgi kaynakları her ne kadar standart bir yapıda sunulmak istense de, şu anda var olan yığın haldeki kaynakların standartlaştırılması da kolay değildir. Kaynakların sunulma şekli kadar onların üretilmesi ve yönetilmesi de önemlidir. Bu amaçla tasarlanan XML, internet ortamındaki kaynakların tanımlanması ve dağıtılmasında altyapı görevini üstlenir. RDF, bu yapının üzerinde kendi tanımlamalarını kullanarak özelleşmiş bir kaynak oluşturur.

Konu bu kaynakların yönetilmesine geldiğinde, ontoloji tanımı ortaya çıkar. Aynı bilgi kaynağını ifade eden bir kavram farklı bir alanda, farklı bir kavram olarak kullanılabilir. Eğitim ontolojisinde yazar kavramı, film ontolojisindeki yazar kavramı ile örtüşebilir. Ya da eğitim ontolojisindeki bir yazar kavramı, hayvan ontolojisindeki insan kavramından türemiş olabilir. Bu ve buna benzer ilişkilerin tanımlanmasında, ontoloji yapısının kullanılmasını gerektirir. Ontoloji tanımında kullanılan OWL, RDF'i temel alan bir dildir. Ve bu dil kullanılarak bahsedilen ilişkilerin oluşturulmasını sağlar.

Semantic web için kaynak görevini üstlenen RDF içeriği, bir dosyada ya da ilişkisel veri tabanında tutulabilmektedir. Veri tabanlarının güvenlik ya da diğer nedenlerden ötürü internet ortamına açık olmaması, güncelliğini ya da kullanılabilirliğini azaltmaktadır. Diğer yandan xml olarak sunulabilecek bir dosyanın da güncellenmesi problem olmaktadır. Dosyanın paylaşımı ve aynı zamanda güncel tutulması son derece zordur. Web servisleri, ortak bir noktadan bilgi dağıtımı konusunda standart hale geldiğinden, bu bilgi kaynaklarının yönetilmesinde alternatif olmuşlardır.

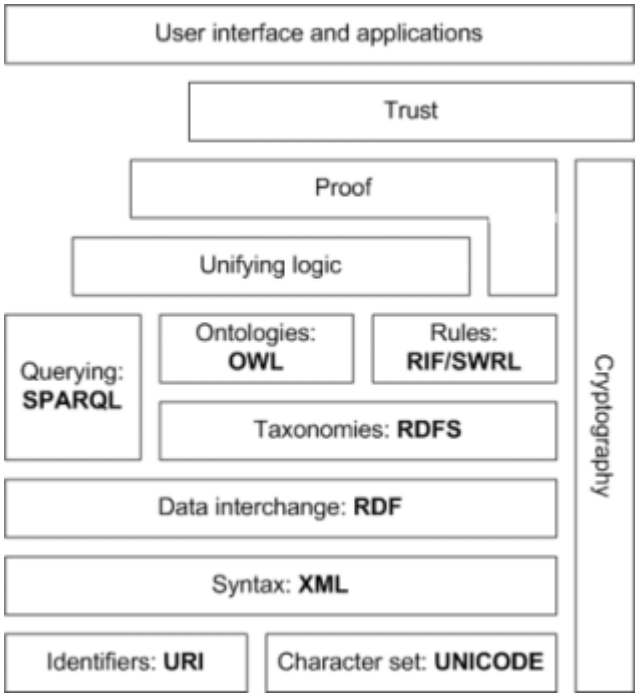
Bu amaçla ortaya konulan bu tez çalışması, Semantik Web' in oluşumunda kullanılacak bilgi kaynağının (şemanın) oluşturulmasını ve yönetilmesini amaçlamaktadır. Bu amaçla RESTful (kaynak bazlı) çalışan web servisi oluşturulacaktır. Sunucu üzerinde çalışacak yazılım, semantik yapının oluşturulmasını ve yönetilmesini sağlayan Jena Framework (yazılım çatısı) kütüphanesini kullanacaktır.

Aşağıda semantik web ile ilgili genel kavramlar ve standart protokoller ile restful web servisleri açıklanacaktır. 2. bölümde tezde ele aldığımız ontoloji oluşturma ve sorgulama problemi açıklanacak, 3. bölümde bu problemin çözümü önerdiğimiz restful web servisleri tabanlı sorgulama protokolü sunulacaktır. 4.bölümde önerdiğimiz protokolün gerçekleştirimi sunulacak ve 5.bölümde sonuç ve gelecek çalışmalara değinilecektir.

## a. SEMANTİK WEB

Halihazırdaki web yapısının yeterli gelmemesinden dolayı, webin kurucusu Tim Berners Lee tarafından ortaya atılan bu yapı var olan yapının üzerinde bir geliştirme olarak tanımlanabilir. Bu bölümde Semantik Web' in oluşumunda kullanılan protokoller ve teknolojiler anlatılmıştır.

Varolan teknolojiler kullanılıp günümüz web içeriği işlenmek istendiğinde çok karmaşık ve yapay zeka içerikli bir yöntem izlenmesi gerekirdi [2]. Bunun yerine varolan içeriğin bilgisayarlar tarafından işlenebilecek bir formatta sunulması amaçlanmıştır. World Wide Web yapısında standartları koyan kurum olan W3C, Semantik web'i oluşturan yapıyı Şekil 1'deki gibi tanımlamıştır.




Şekil 1. Semantik Web Yapısı

En alt düzeyde URI (evrensel kaynak tanımlayıcı),'yi temel alan bu yapı XML tabanlı bilgi kaynağın oluşturulup sorgulanmasını ve aynı zamanda şifrelenebilmesini sağlar. Bu yapıda kullanılan bazı protokoller ve teknolojiler aşağıda açıklanmıştır.

**XML**

Konu bilgi transferi olduğunda kullanılan yapı olarak karşımıza çıkan XML (eXtensible Markup Language: Genişletilebilir İşaretleme Dili), bazı yanlış anlaşılmalarda olduğu gibi HTML'e alternatif değildir. XML, XSLT kullanılarak bir arayüz oluşturabilse de, temel amacı web ortamında bilgi taşınmasını sağlamaktır. HTML'in aksine kullanıcılar kendi etiketlerini oluşturabilmektedir [3].

İhtiyaçlar doğrultusunda kullanılabilen etiketler ve bunların özellikleri, Semantik Web'in bu yapıyı temel almasında önemli rol oynamıştır. XML verisinin etiketlerini açıklayan ve kısıtları içeren XML Schema, Belli bir etikete ya da özelliğine erişmek için kullanılan kısayol yapısı olan XML Path, bu etiket içerikleri içerisinde sorgu kullanımı sağlayan Xquery gibi teknolojileri barındırır [3]. Bu teknolojiler tam

anlamıyla kullanılmasa da XML, web servislerinde temel veri dosyası olarak kullanılmaktadır. Her ne kadar etiket isimlerinin tekrar etmesi nedeniyle büyük veri yapılarında çok yer kaplasa da Semantik Web için temel yapı taşı görevini üstlenmiştir.

XML'in bu kadar yaygın kullanılmasının bir diğer sebebi de Unicode karakter kodlamasının desteklenmesidir. Şekil 2'de ontoloji yapısı için oluşturulmuş bir XML örneğini görebilirsiniz [4].

```
<?xml version="1.0" encoding="utf-8"?>
<Ontology about="Animals">
  <versionInfo>$Revision: 1.3 $</versionInfo>
  <comment>
    This is an example ontology expressed in OWL for testing F-OWL
    inference rule.
  </comment>
</Ontology>
```

Şekil 2. XML Dokümanı Örneği

Örnekte görüldüğü üzere ilk satır XML'in temel tanımı olan yapı kullanılmaktadır. Bundan sonra kullanılan bütün etiketler kullanıcıya bırakılmıştır. Ontoloji tanımı için kullanılan bu xml örneğinde "Ontology" etiketi ve ona ait about özelliği eklenmiştir. Dikkat edilirse xml'in en önemli gerekliliği her etiketin kapatılması gerektiğidir. HTML sayfalarında kapatılmayan bir etiket bazen istemcilerde sorun yaratmasa da, XML için bu durum söz konusu değildir.

**RDF**

Kaynak Tanımlama Çerçevesi diye tercüme edilen RDF, triple (üçlü: Object → Predicate → Subject) adı verilen yapılardan oluşmuştur. Üçlü olarak adlandırılan bu yapılar özne, özellik ve değer üçlüsünü barındırır. Özne URI ile tanımlanan bir kimlik, özellik bu öznenin sahip olduğu bağlantının tanımı (standart özellikler (W3C XSD şeması) ya da ontoloji yapısındaki özellikler olabilir), değer ise bu bağlantının içeriğini ifade eder. Değer başka bir özneyi gösterebilmekle beraber ontolojide literal (sabit değer) olarak tanımlanan sabit yapılar (metin, sayı, tarih vb) da içerebilir [5]

RDF, tanımlamasında temel olarak XML kullanılsa da, N-Triple, Notation 3, Turtle gibi bilinen diğer formatları da vardır. XML olarak tanımlanan yapısı RDF/XML

olarak anılır [6]. Şekil 3'de RDF/XML formatında yazılmış bir tanımlamayı görebilirsiniz. RDF dosyası temelde bir XML dosyası olmasından dolayı, ilk satırda xml deklarasyonu içermek zorundadır. Sonrasında rdf dosyası olduğunu belirten rdf etiketi ve bu etiketin özellikleri arasında yeralan isimuzayı bilgileri yer alır.

Dokümanın başında XML deklarasyonu bulunmaktadır. Sonrasında kök eleman olarak rdf başlangıcı ve isim uzayı kısaltmalarının tanımları gelmektedir. Başlangıç bölümlerinin ardından ise esas rdf üçlülerinin geldiği görülmektedir.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://my.library/rdf/syntax#">
  <book rdf:about="http://my.library/book/0836217462">
    <isbn>0836217462</isbn>
    <title>Being a Dog Is a Full-Time Job</title>
    <author rdf:resource="http://my.library/author/Charles-M.-Schulz"/>
    <character rdf:resource="http://my.library/character/Peppermint-Patty"/>
    <character rdf:resource="http://my.library/character/Snoopy"/>
    <character rdf:resource="http://my.library/character/Schroeder"/>
    <character rdf:resource="http://my.library/character/Lucy"/>
  </book>
</rdf:RDF>
```

Şekil 3. RDF Dokümanı Örneği[1]

Örnekte Rdf etiketinin altında oluşturulan her alt etiket bir özneyi temsil eder. Özneye ait tüm değerler de bir alt etiket olarak tanımlanmıştır. Örnekte title özelliği tanımlanırken <title> etiketi kullanılmış, aldığı değer ise iki etiket arasında tanımlanmıştır. Literal özelliği taşıyan değer bilgileri etiket içerisinde verilirken, başka bir özneye bağlanan değerler ise rdf:resource niteliği ile tanımlanmaktadır.

Bu örnekte özne bir kere tanımlanıp, bu özneye ait bütün özellik -> değer ilişkileri toplu halde tanımlanmıştır.

**RDFS**

RDF verilerinde nitelik tanımı yapılırken bu şema kullanılır. XML şemalarında olduğu gibi bir rdf dosyasında tanımlanabilecek etiketler ve içeriklerini belirler. İlk

[1] http://www.xml.com/2000/10/04/linking/lib5-triples.html adresinden alıntılanmıştır.

olarak 1998 yılında yayınlanan bu yapı, 2004 yılında son halini almıştır [7]. RDF için kullanılabilen şu etiketler bu şema içerisinde yer almaktadır.

Sınıf bilgileri tanımlanırken;
**rdfs:Resource** literal olmadığı sürece değer olarak kullanılan tüm kaynaklar bu etiket kullanılarak tanımlanır.
**rdfs:Class** bir sınıf tanımı yapılırken kullanılır.
**rdfs:literal** bir değer bilgisinin metin yapısında veri içerdiğini gösterir. Tanımlı bir değer (XSD şemasında tanımlı) ya da sabit bir değer.
**rdfs:Datatype** bir instance (sınıftan türemiş nesne) için kullanılabilecek bir property'nin (özellik) DatatypeProperty (metin tabanlı özellik) mi ObjectProperty (nesne tabanlı özellik) mi olduğunu belirtmek için kullanılır.

Bunun dışında property'ler için kullanılan etiketler de vardır:
rdfs:domain, rdfs:range, rdfs:subClassOf, rdfs:subPropertyOf gibi.

**OWL**

Ontoloji philosopy kelimesinden ortaya çıksa da Semantik Web'de Thomas Gruber'in "ontoloji; bir kavramsallaştırmanın tanımsallaştırılmasıdır" tanımını kullanılmaktadır [2]. OWL, bu alandaki ontolojinin tanımlanmasını sağlayan bir dildir. OWL-Lite, OWL-DL ve OWL-Full adında üç farklı versiyonu bulunur [22]. OWL dili daha çok RDF ve RDFS kullanılarak tanımlanır [8]. Ontoloji kavramları oluşturmak için barındırdığı sınıfları (owl:Class), instance'ları ve bu ontolojide tanımlı özelikleri (owl:Property) barındırır.

Üçlüler instance'lar için tanımlanır. Her instance bir özne, kullandığı property bir nesne ve içerdiği predicate bir değer olarak belirtilir. Burada kullanılan property ontoloji içerisinde DatatypeProperty ya da ObjectProperty olmasına göre literal olup olmadığı anlaşılır.

## SPARQL

SPARQL (SPARQL Protokolü ve RDF Sorgulama Dili), rdf verileri üzerinde SQL sorgusuna benzer şekilde sorgu yapılmasını sağlar. SQL'den farklı olarak yalnızca SELECT sorgusu, yani veri elde etme işlemi yapılabiliyor (SPARQL/Update sonradan bu işlemler için ortaya çıkmıştır) [9]. Bunun yanında ek olarak yine sadece verinin çekilmesi işleminde kullanılabilen ASK, DESCRIBE ve CONSTRUCT sorgu çeşitleri kullanılabilir. SELECT SQL'den de bilindiği üzere veri kümesinin tamamını ya da FILTER kısmında belirtilecek koşula bağlı olarak belirli bir kısmını getirir.

CONSTRUCT, aynı şekilde SELECT sorgusunda olduğu gibi koşula bağlı getirilen veri kümesini, doğru şekilde üçlüleri içerdiği takdirde bir rdf grafiği çıktısı üretmeye yarar.

ASK sorgulanan içeriğin veri kümesi içinde var olup olmadığı "true" ya da "false" olarak cevabını döndürür. Son olarak DESCRIBE ise WHERE kısmında filtrelenip elde edilen bilginin RDF veri kümesi içindeki tanımlamasını döndürür. DESCRIBE daha çok RDF veri kümesinin içeriğinin bilinmediği durumlarda kullanılır [10].

## SPARQL Update

SPARQL/Update RDF verisinde güncelleme yapmak için kullanılır. SPARQL/Update SPARQL diline yakın bir dil olup, SPARQL sorgu dili ile birlikte kullanılması öngörülmüştür [11].

SPARQL/Update aşağıdaki olanakları sağlar:

- RDF grafiğine yeni bir üçlü (triple) ekleme
- RDF grafiğine kümesinde bir üçlünün silinmesi
- Tek adımda bir grup güncelleme işlemi yapabilme
- RDF grafik ambarına yeni bir RDF grafiğinin eklenmesi
- RDF grafik ambarından bir RDF grafiğinin silinmesi

## b. RESTFUL WEB SERVİSLERİ

REST, temsili durum transferi olarak adlandırılan bu yapı 2000 yılında Roy Fielding tarafından doktora tezi olarak yayınlanmıştır [12]. Burada amaç web üzerindeki bir kaynağın istenilen yapıda sunulabilmesi ve değiştirilebilmesidir. Aslında sistem olarak tanımlanan bir yapının REST yapıda olabilmesi için (RESTFUL) aşağıdaki kısıtları barındırması gerekir [13];

- Sunucu-istemci yapıda olması
- Durum bilgisi içermemesi- Her isteğin birbirinden bağımsız olması
- Önbelleklemeyi desteklemesi – Ağ yapısının buna göre düzenlenmiş olması
- Eşit oranda erişilebilir olması – Her kaynağın geçerli ve ünik bir adres sahip olması

Aslında var olan web yapısında statik içerikler REST yapısını desteklemektedir. Herhangi bir durum bilgisi barındırmadığı gibi (kullanıcı doğrulama, erişim izinleri vs.) kaynak adreslerinin değişmesi de söz konusu değildir. Değişseler dahi, linkler güncelleneceğinden dolayı, var olan HTML içeriğinden istemciler tarafından çözülebilir.

Fakat günümüzde daha çok dinamik web içerikleri kullanılmaktadır. Bu da REST yapısının kısıtlarını destekleyememesine neden olmaktadır. Kaynağa erişimde kullanıcı bazlı değişim sergilenmektedir. Bu da doğal olarak REST yapısına aykırı düşer.

RESTful yapısı genellikle web servisleri ile kullanılır. Web servisleri, bilindiği gibi tek bir amaca hizmet eden, stateless (çalışma esnasında herhangi bir durum barındırmayan) sunucu uygulamalarıdır. Web servislerinin stateless yapısı RESTful bir sistemi mümkün kılar.

Aşağıda RESTful yapıyı destekleyen bir sistem için kullanılan kavramlar açıklanmıştır.

**RESOURCE**

Kaynak bir sunucu üzerinde adreslenebilir herhangi bir şey olabilir. Burada kaynak istemci ve sunucu arasında kullanılmaktadır.

**REPRESENTATION**

Kaynağın hangi yapıda sunulacağını belirtir. Kaynak kaydedildiği ortamdaki orijinal halinden farklı şekilde istemciye ya da istemciden sunucuya aktarılabilir.

**URI**

Tekdüze kaynak tanımlayıcı olarak tercüme edebileceğimiz bu yapı, kaynağın adreslemesi için kullanılır. Günümüzde web adresleri köprü (hyperlink) kullandığı için URI daha çok köprü şeklinde kullanılır.

Sunucu üzerinde REST yapısının desteklenmesi için web sunucuları üzerinde asıl uygulama kodu çalıştırılmadan önce yorumlayıcı görevi gören bir REST çerçevenin (Framework) olması gerekir. Bu çerçevenin görevi gelen URI bilgisinden istenilen kaynağı algılayıp asıl uygulamaya aktarmaktır.
Bu kavramların yönetilmesi kısmında ise HTTP komutları devreye girer. Çizelge 1'de gösterildiği gibi kaynaklar üzerindeki işlemler için kullanılır.

| HTTP Komutu | Görevi |
|---|---|
| **GET** | Kaynağı alır |
| **POST** | Kaynağı oluşturur. |
| **PUT** | Kaynağı günceller. |
| **DELETE** | Kaynağı siler. |

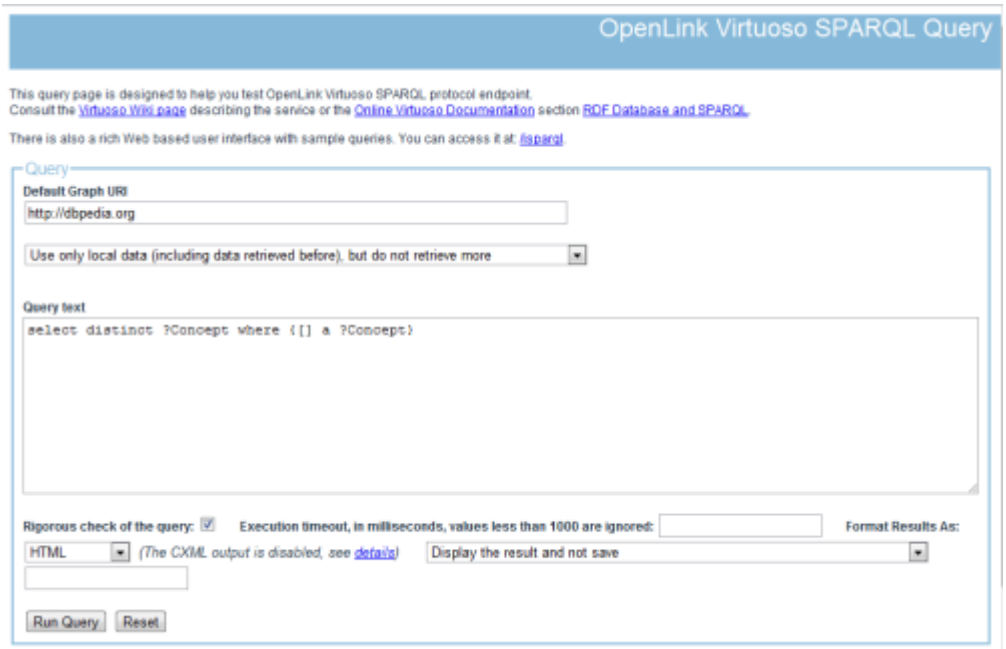
Çizelge 1. HTTP Komutları

## 2. ONTOLOJİ OLUŞTURMA VE SORGULAMA PROBLEMİ

Ontoloji belli bir etki alanını (domain) temsil eden tüm kavramların bir arada tutulduğu oluşumdur. Her ontoloji kendi alanındaki kavramları barındırabileceği gibi, başka ontolojilerdeki kavramları da referans gösterebilir. Web ortamındaki her kavramın belli bir etki alanında temsil edilmesi ve ilişkili olduğu diğer kavramlarla bağlantıları doğru bir şekilde tanımlanmalıdır. OWL bu işi üstlenebilecek bir format olduğu halde, bu dille yazılmış veri kümelerinin oluşturulması ve yönetilmesi çok fazla zaman almaktadır.

DBPedia ve Freebase gibi iki büyük oluşumun Wikipedia[2] ve benzeri internet bilgi depolarını kullanarak oluşturdukları veri kümelerine sahiptir. Semantik Web veri tabanı olarak görev yapan bu kaynaklar üzerinde sorgulama SPARQL veya MQL gibi sorgulama dilleri kullanılarak kullanılabilmektedir. Freebase kendi üzerinde ek olarak adres satırından kaynak çağırma desteklense (mqlread) de henüz standart ve basit düzeyde bir sorgulama yöntemi geliştirilmemiştir.

### DBPedia




Şekil 4. DBpedia Sorgu Arayüzü

[2] Wikipedia, http://www.wikipedia.org

Web dünyası için büyük bir kaynak oluşturan Wikipedia, DBPedia gibi semantik bir veri ambarının oluşumda rol almıştır. Ontolojinin oluşturulmasında dışarıdan herhangi bir etkileşim olmamakla beraber SPARQL kullanılabilen web arayüzü[3] sayesinde ontolojilerin sorgulanması mümkün kılınmıştır [10].

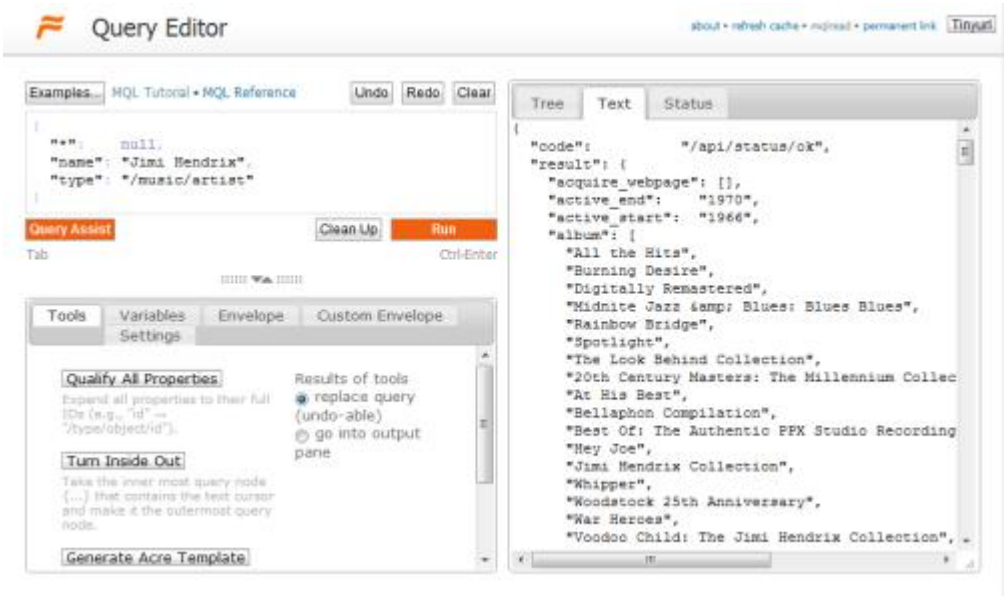
**Freebase**

Freebase[4], bu alanda biraz daha etkin bir kullanıma sahiptir. Kayıtlı kullanıcıların var olan ontoloji yapısına müdahale edebilmesi sağlanmıştır. Yine aynı şekilde insan etkileşimi kullanılması gerekmektedir. Şema odaklı bu yapıda, domain olarak tanımlanan ontoloji yapısı çeşitli türler (sınıflar) ve bunların nitelikleri şeklinde bir yapıya sahiptir. MQL olarak adlandırılan sorgulama ile veriler ya da şema içeriği görüntülenebiliyor [14].

Şekil 5'de de gösterildiği gibi dâhili bir çözüm olarak bir sorgu editörü sunmaktadır. mqlread ya da Freebase' in sunduğu sorgu editörü (query editor) kullanılarak bu sorgulama gerçekleştirilebiliyor. DBPedia'da elde edilen veri kümesine nazaran JSON formatındaki çıktı yapısı, uygulamaların kullanımına daha uygun bir çözüm sunabilmektedir [15]. Yine de ontolojilerin tanımlanması işlevi daha çok insanların elle girişleriyle mümkün olabiliyor. Bu kısım için geliştirilmiş bir protokol bulunmamaktadır.

[3] DBPedia SPARQL Endpoint, http://www.dbpedia.org/sparql

[4] Freebase, http://www.freebase.com




Şekil 5. Freebase Query Editor

Örnek olarak verebileceğimiz Music domainindeki (ontolojisindeki) bir artist sınıfından türenmiş Jimi Hendrix instance'ı sorgulanırken kullanılan MQL dili, mqlread yapısı tercih edildiğinde adres satırında parametre olarak yer alır;

```
http://www.freebase.com/api/service/mqlread?query={%20%22query%22:%20{%20%22*%22:%20null,%20%22name%22:%20%22Jimi%20Hendrix%22,%20%22type%22:%20%22/music/artist%22%20}%20}
```

Çizelge 2. mqlread Kullanımı

Bu sorgu editörü olarak kullanılmak istendiğinde, sorgu aşağıdaki yapıda oluşturulur;

```
{
 "*":  null,
 "name": "Jimi Hendrix",
 "type": "/music/artist"
}
```

Çizelge 3. MQL Sorgulama

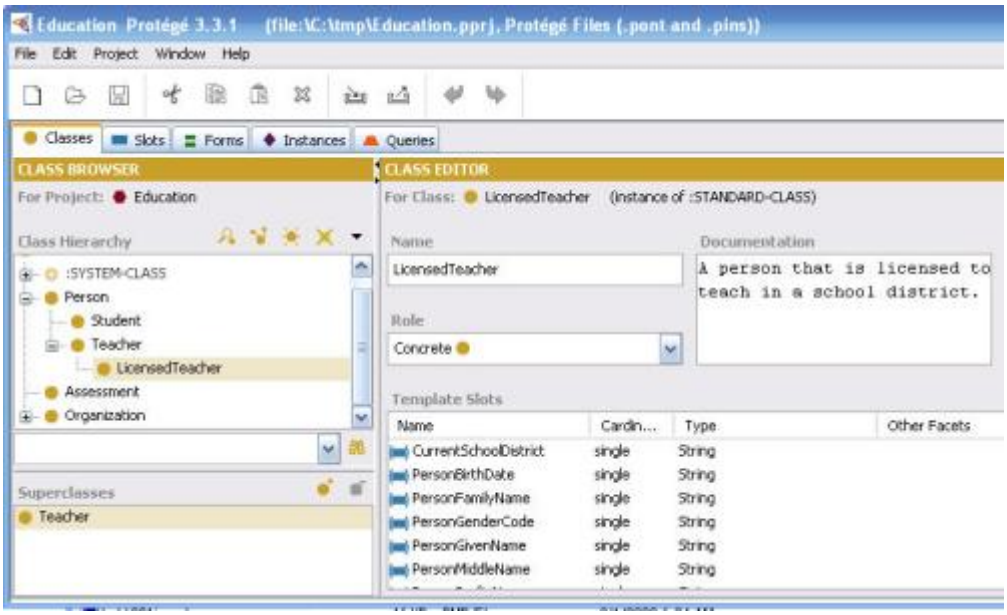
Freebase var olan bilgi kaynağının sorgulanması haricince kullandığı şema yapısının da sorgulanmasına izin vermektedir. Bu yapı uygulama geliştiricilerin bu veri tabanını kullanmak isterken oluşturmaları gereken şemayı çözümlemelerini sağlar. Çizelge 4'de Freebase veri tabanından artist sınıfının nitelikleri listelenmek istendiğinde kullanılması gereken sorgu örneği verilmiştir.

```
[{
  "id":         "/music/artist",
  "properties": [{}],
  "type":       "/type/type"
}]
```

Çizelge 4. Örnek MQL Sorgusu

**Protégé**

Stanford Üniversitesinin geliştirmiş olduğu bu uygulama, ontolojilerin tanımlanması ve bu ontoloji içeriğinin veri dosyası (.owl) olarak kaydedilmesini sağlar [16]. Editör görevi gören bu uygulama aynı zamanda grafik üretimini de sağlar (İlişkilerin gösterimi için). Amaç olarak ontolojilerin üretilmesi görevini üstlense de bunu insan etkileşimi kullanarak gerçekleştirir. Bu da oluşturulacak verilerin güncelliğinin korunması ve yayınlanmasında bir katkı sağlayamamaktadır.




Şekil 6. Protégé Arayüzü

Ontoloji yapısı çoğunlukla RESTful web servislerinin açıklanmasında kullanılmıştır. RESTful web servislerinin içerdikleri kaynakların birbirleriyle olan ilişkileri ve URI bilgileri ontoloji yapısının sunduğu içerikle sağlanmak istenmiştir. Bu konuda yapılan çalışmalar aşağıda listelenmiştir.

**SA-REST**

RESTful web servislerinin standart html ve XML metadata (tanımlayıcı veri) açıklamasına ek olarak, hangi domainde olduğu (domain-rel), başka bir kaynakla olan ilişkisi (sem-rel) ya da hangi sınıftan türediği (class-rel) gibi fazladan bilgileri içerir. W3C tarafından 2010 yılında bir üye olarak kabul edilen SA-REST, metadata verileri için bir zenginleştirme yapısı olarak kullanılmaktadır [17].

**EXPRESS (EXPressing REstful Semantic Services)**

SA-REST' in tersine EXPRESS, HTML veya XML verisini zenginleştirmek yerine, OWL formatında bir açıklama bilgisi sunmaktadır. Bir kaynağın hangi türden oluştuğu ve hangi özellikleri barındırdığını OWL ile açıklar. Bunun dışında bu yapıda bir kaynak üzerinde hangi HTTP komutlarının kullanılabileceğini belirten bilgiler de içerir [18].

**ReLL (Resource Linking Language)**

Biçimleme dili içerisinde var olan kaynak ve bunların path (yol) bilgilerini içeren etiketleri kullanır. Bu sayede uygulamaların XPath ya da regex (regular expression) bilgileri kullanarak kaynağa ulaşmasını sağlayabilmektedir [19].

```
<resource xml:id="person">
  <name>Person Page</name>
  <desc>The home page for a person.</desc>
  <uri match="http://.*?/people/(faculty|students|staff|visitors)/[a-zA-Z]+" type="regex"/>
  <representation xml:id="person-html" type="http://www.iana.org/assignments/media-types/text/html">
    <name>Person HTML Page</name>
    <link xml:id="person-website" type="website">
        <selector select="//div[@class = 'field-field-person-website']//a/@href" type="xpath"/>
    </link>
    <link xml:id="person-course" type="personcourse" target="course">
        <selector select="//span[@class = 'views-field-title']/a/@href" type="xpath"/>
        <protocol type="http">
            <request method="get"/>
            <response media="http://www.iana.org/assignments/media-types/text/html"/>
        </protocol>
    </link>
  </representation>
</resource>
```

Şekil 7. Örnek ReLL İçeriği

**Presto**

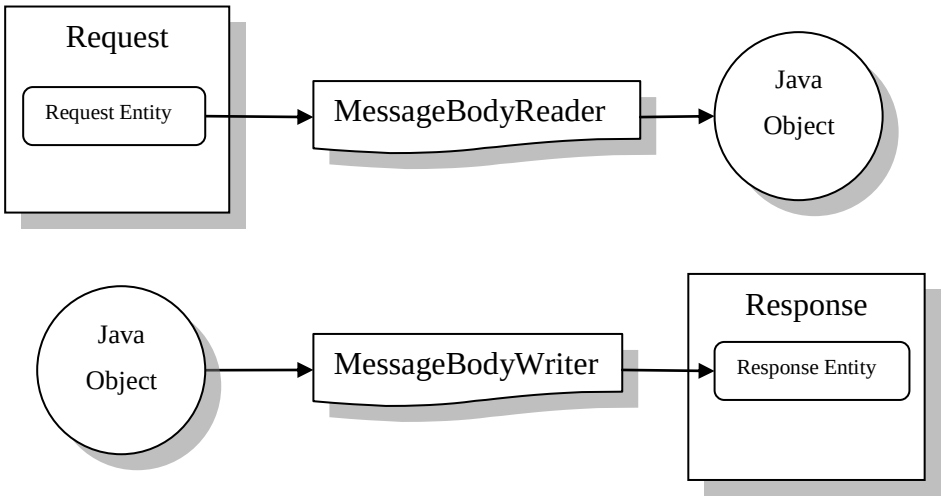
Yapılan çalışmalar bir ontolojinin oluşturulup yönetilmesinden çok, RESTful web servislerinin yayınlanmasında ontoloji bilgilerinin kullanımına yöneliktir. Ontolojilerin sorgulanması kısmında Presto daha çok örnek sayılabilecek bir çalışmadır.

Presto bu alanda sadece ontolojilerin sorgulanmasını amaçlamıştır. Bu tez çalışmasında temel alınmış bazı özelliklere sahiptir. GET komutu istenen kaynağı getirmekle beraber, POST komutu bizim çalışmamızdaki görevi üstlenmez. Presto POST komutunu SPARQL sorgusunu göndermek için kullanmaktadır. Burada amaç adres bilgisini daha sade tutabilmektir [20].

# 3. RESTFUL ONTOLOJİ SORGULAMA PROTOKOLÜ

Bu bölümde veri tabanında kayıtlı bir ontolojideki kavramların sorgulanması, değiştirilmesi ya da yeni bir ontoloji oluşturmak için oluşturulmuş bir protokolün çalışma mantığı anlatılmaktadır. Bu protokolün anlatılmasında web ortamından alınan hazır OWL[5] dosyası kullanılmıştır.

Glassfish web sunucusu üzerinde kullanılan JAX-RS yorumlayıcısı HTTP istekleri (request) ile Java nesneleri arasında köprü görevi görür. Jersey (JAX-RS), bu istekleri işlerken varlık sağlayıcıları (entity providers) kullanır [21]. Bunlar bir kısım string,stream, java.io.Reader objects, java.io.File ve diğer Java nesnelerini işler. Aynı şekilde yanıtlar da bu varlık sağlayıcıları tarafından yönetilir. Java metodunun döndürdüğü tür, bir yanıt (response) olarak üretilir ve buna uygun bir başlık (header) eklenerek istemciye gönderilir.




Şekil 8. Jersey (JAX-RS) Çalışma Prensibi

Kullanılan RESTful web servisi, stateless bir yapıya sahip olduğundan istemciye bir kaynak döndürebileceği gibi başka bir kaynak adresine de yönlendirebilir.

Burada önemli bir görevi üstlenen Jena, ModelFactory nesnesini kullanarak bir ontolojinin oluşturulmasını ya da değiştirilmesini mümkün kılar. Belli bir kaynağa

[5] http://www.atl.lmco.com/projects/ontology/ontologies/animals/animalsA.owl

erişmek istendiğinde (http://www.anonim.com/kitaplar/kitap1 gibi), sorumlu metot bu işi Jena' ya devreder. Jena o an aktif olarak sistemin geçici hafizasında ya da belirlenmiş veri tabanındaki kaynağı istemciye sağlar.

Birden çok kaynağın çekilmesi durumunda kullanılacak SPARQL sorgusu, parametre içerisinde düzyazı olarak sunucuya gönderilir. Uygulama geliştirilirken bu sorgu, Jena'nın metotları ile kaynak yerinden çekilir.

Jena, bu SPARQL sorgularının işlenmesi için ARQ bileşenini barındırır. Standart SPARQL dışında gruplama ve verilerin farklı kaynaklarda barındırılmasını destekler. ARQ, TDB ve SDB olarak iki bileşeni kullanabilir. TDB SQL kullanan ilişkisel veri tabanlarını destekler. TDB geniş çaplı RDF veri kümelerinin standart Java motoru ile sorgulanabilmesini sağlar.

Jena SDB bileşeni sayesinde kısıtlı sayıdaki veri tabanlarına destek sağlar. Uygulamada kalıcı hafıza olarak MySQL veri tabanı kullanılmaktadır. Java içerisinde standart olarak desteklenen JDBC sayesinde yerel ya da ağ üzerinden başka sunucudaki veri tabanına erişim sağlanır.

**Veri tabanındaki Ontolojiler**

Her bir ontoloji farklı bir model tablosu içerisinde tutulmaktadır. Ontolojilere erişmek için yerel isimler kullanılacaktır. Bu bizim RESTful arayüzünde ontoloji isimlerinin basit olarak kullanılmasını sağlar. GET metotlarının veri gönderemedikleri düşünüldüğünde, bu yöntemin kullanımını mecbur kılmaktadır. Ontoloji namespace (isim uzayı) parametre olarak (/ontology?namespace=http://..) gönderilebilseler de, bu adres satırındaki basitliği engellemektedir. REST kullanımında belirli bir hiyerarşi izleneceğinden sadece path (yol) isimlerinin kullanımına izin verilmektedir.

http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı}/{property-adı}
Ontoloji property'leri için sorgulama ve tanımla farklı bir adres yolu ile yapılmaktadır.
http://sunucu/{ontoloji-adı}/property/{property-adı}

Dikkat edilirse burada property bilgisi çekilirken ve tanımlanırken yol kısmında ayrılmış "property" kelimesi kullanılmıştır. Bu işlem verilen adresten bir sınıf nesnesinin mi yoksa bir property'nin mi istendiğini ayırt etmek için kullanılır

http://sunucu/Animals/property/age

http://sunucu/Animals/Person/John

Bu kullanımdan doğan kısıtlama ise, ayrılmış "property" kelimesinin sınıf adı olarak kullanılamamasıdır.

**Ayrılmış Terimler (Typed Keywords)**

Adres satırında kaynak tabanlı sorgulama yapılırken bazı kelimelerin kullanımı engellenmiştir. Bu kelimeler JAX-RS için düzenli ifadeler (regular expression) kısmında ayrıştırılan edilmesi için tanımlanmıştır. Aşağıda verilen bu ifadeler herhangi bir instance ya da sınıf için kullanılamaz.

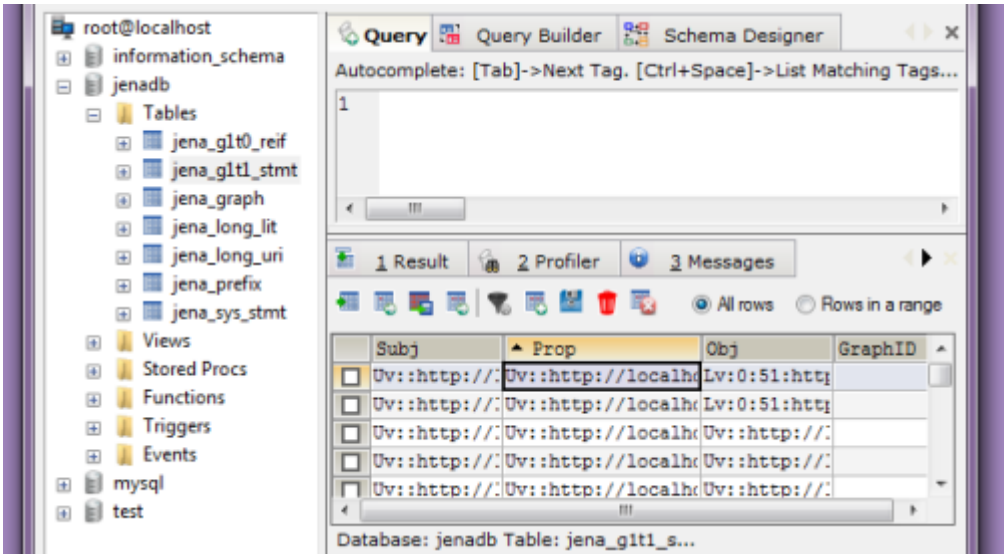
- ontology
- class
- property
- instance

**Veri tabanı Yapısı**

Ontoloji verileri, Jena Framework'ün kullandığı MySQL veri tabanında model olarak tanımlanarak saklanır. Her model için yeni bir tablo kullanılır. Bu tablo statement (Subject→Predicate→Object üçlüsü) bilgilerini içerir. Jena, bu statement verilerini kullanarak hafıza üzerinde modelleme yapar.

| Tablo Adı | Açıklama |
|---|---|
| **jena_graph** | Veri tabanındaki modellerin isimlerini tutar. |
| **Jena_sys_stmt** | Veri tabanın yapısını tutmak için, Framework'ün kullandığı bir tablodur. Uygulamada kullanacağımız xml:base bilgisi de burada tutulmaktadır. |
| **jena_gt..t.._stmt** | Belirli bir model için statement bilgilerini tutar. Her statement aslında bir üçlü bilgi anlamına gelir. |
| **jena_long_uri** | Büyük URI bilgilerini tutar. |
| **jena_long_lit** | Büyük literal bilgilerini tutar. |

Çizelge 5. Jena Framework Veri tabanı Tablo Bilgisi




Şekil 9. Jena Framework'ün Veri tabanı Yapısı

Her statement içerisindeki tanımlama URI bilgisi kullanılarak yapılır. (Uv:: ile başlar). URI bilgileri gereğinden fazla uzun olduğunda, jena_long_uri tablosunda tutulur ve orayı işaret eden ünik bilgi statement tablosunda referans olarak tutulur (Ur: ile başlar).

**HTTP Komutlarının Kullanımı**

Sorgulama sırasında istek bilgisinde "Accept" başlık bilgisi belirtilmesi gerekiyor. Sunucu bu bilgiye göre istemciye veri türünü hazırlayıp gönderir. HTTP paket bilgisindeki bu başlık bilgisi, JAX-RS tarafından çözümlenir (Metodun @Produces nitelik bilgisinde bu bilgi tanımlanmıştır.)

Text/html, web browser (web istemcisi) için oluşturulmuş olup, diğer iki türden farklı olarak daha zengin bir içeriğe sahiptir. XML ve JSON verileri, OWL sözdizimini kullanarak üretilir. JSON verisi üretilmeden önce XML verisi üretilir ve JSON Kütüphanesi kullanılarak XML verisinden dönüştürme (serializing) yapılır. Bu noktada tüm prefix'lerin tanımlı olduğundan emin olunması gerekir.
GET komutu kullanıldığında aşağıdaki çıktı türleri kullanılabilir;

- text/html
- application/xml
- application/json

**Örnek GET komutu kullanımı ve sonucu:**

```
GET /Animals/Person/John HTTP/1.1
Host: http://localhost
Accept: application/xml
```

```
HTTP/1.1 200 OK
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <Person rdf:about="http://localhost/roq-ns#John">
    <age rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">37</age>
    <shirtsize rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">12</shirtsize>
    <shoesize rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">14</shoesize>
    <type rdf:resource="http://localhost/roq-ns#Person"/>
  </Person>
</ResponseData>
```

XML veri türü üretilirken ilk öğe olarak <ResponseData> kullanılır. Bu sayede çoklu öğe gönderiminde, eksik veri alımını engeller. İçerik application/json formatında istendiğinde aşağıdaki sonuç elde edilir.

```
HTTP/1.1 200 OK
{
 "@xmlns:owl": "http://www.w3.org/2002/07/owl#",
 "@xmlns:rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
 "@xmlns:rdfs": "http://www.w3.org/2000/01/rdf-schema#",
 "@xmlns:xsd": "http://www.w3.org/2001/10/XMLSchema#",
 "Person":  {
  "@rdf:about": "http://localhost/roq-ns#John",
  "age":    {
   "@rdf:datatype": "http://www.w3.org/2001/XMLSchema#nonNegativeInteger",
   "#text": "37"
  },
  "shirtsize":    {
   "@rdf:datatype": "http://www.w3.org/2001/XMLSchema#decimal",
   "#text": "12"
  },
  "shoesize":    {
   "@rdf:datatype": "http://www.w3.org/2001/XMLSchema#decimal",
   "#text": "14"
  },
  "type": {"@rdf:resource": "http://localhost/roq-ns#Person"}
 }
}
```

XML ve JSON veri türleri üretilirken tüm XML öğeleri, içerdikleri niteliklerde namespace bilgisini barındırır.

**Ontoloji Sorgulama**

| HTTP Komutu | Kullanılacak yol | Post bilgisi |
|---|---|---|
| **GET** | http://sunucu/ontology | Yok |
| | Tüm ontolojileri listeler. | |
| **POST** | http://sunucu/{ontoloji-adı} | Var |
| | Yeni bir ontoloji ekler | |
| **PUT** | http://sunucu/{ontoloji-adı} | Var |
| | Var olan bir ontolojiyi değiştirir | |
| **DELETE** | http://sunucu/{ontoloji-adı} | Yok |
| | Ontolojiyi siler. | |

Çizelge 6. Ontolojinin Sorgulanması ve Tanımlanması

**Ontoloji Oluşturma**

Ontolojinin oluşturulmasında üç yöntem kullanılır;

Sadece ontoloji adı ve temel özellikleri tanımlanır; Ontoloji adı, URL'in yol bilgisinden alınır. Diğer bilgiler post bilgisi içerisinden ayrıştırılır. Base namespace bilgisi sunucu tarafında oluşturulur (http://localhost/roq-ns#).

**Örnek Post bilgisi**

Her post verisinde XML ağacının ilk elemanı olarak <PostData> öğesinin kullanılması zorunlu kılınmıştır. Sadece ontolojinin kendisi oluşturulurken http:// ile başlayan adres bilgisi (OWL dosyasını içeren), ya da OWL dosyasının kendisi post bilgisi olarak kullanılır (rdf:RDF bilgisi içeren). Post bilgisi PUT komut bilgisi için de kullanılır.

```
<PostData>
        <owl:Ontology
…
```

Belirli bir sunucudaki OWL dosyasının kullanılması; Post bilgisi içerisinde kullanılacak OWL dosyasının URI bilgisi tanımlanır. Base namespace bilgisi OWL dosyasındaki xml:base niteliğindeki bilgiden alınır.

**Örnek Post bilgisi:**

http://www.danmccreary.com/presentations/semweb2006/education.owl

Tüm verinin direk olarak post bilgisi içerisinde verilmesi; Ontolojinin tüm içeriği post bilgisindeki tanımlamalara uygun şekilde oluşturulur. Base namespace bilgisi OWL dosyasındaki xml:base niteliğindeki bilgiden alınır.

**Örnek Post bilgisi**

```
<!DOCTYPE owl [
  <!ENTITY animals
"http://localhost/roq-ns#">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
]>

<rdf:RDF
…
```

## SORGULAMA

- **Ontolojilerin Listelenmesi**

Veri tabanında kayıtlı ontolojiler listelenirken, her biri sürüm bilgisi ve yorum kısımlarını da gösterecek şekilde bir yanıt alınır.

**İstek:**

```
GET /ontology HTTP/1.1
Host: http://localhost
Accept: application/xml
```

**Yanıt:**

```
HTTP/1.1 200 OK
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <owl:Ontology rdf:about="Animals">
    <owl:versionInfo>$Revision: 1.3 $</owl:versionInfo>
    <rdfs:comment>
      This is an example ontology expressed in OWL for testing F-OWL
      inference rules. Some ontology concepts are adopted from
      http://
    </rdfs:comment>
  </owl:Ontology>
</ResponseData>
```

- **Yeni bir ontolojinin oluşturulması**

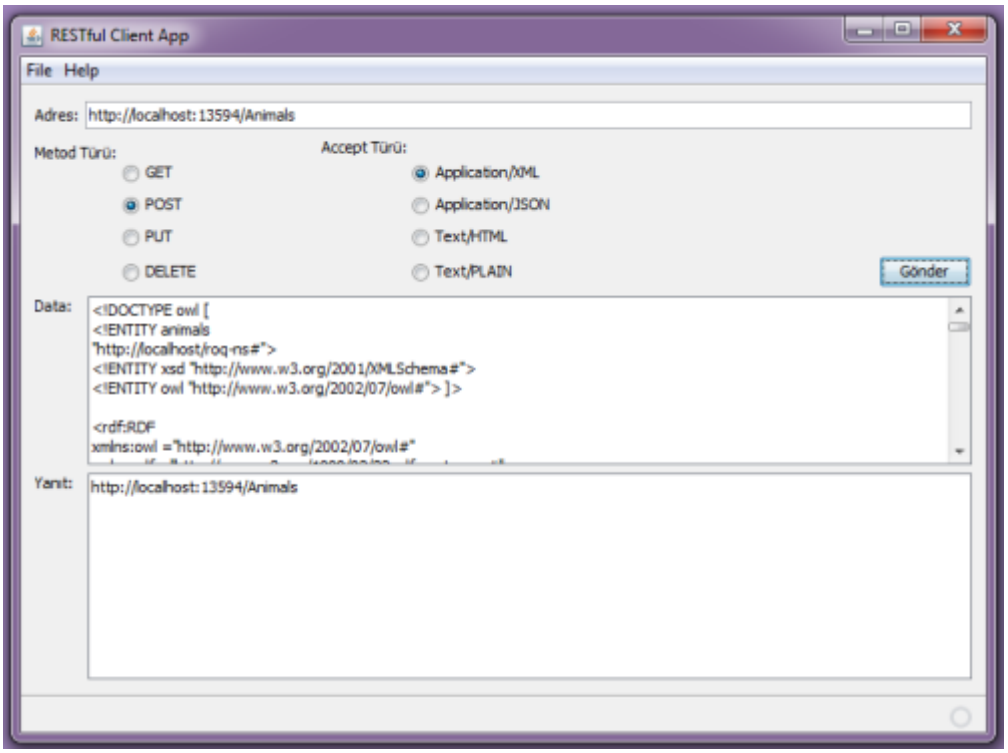
Yeni bir ontoloji oluşturulurken, post bilgisi olarak OWL tanımına uygun ontoloji adı, varsa sürüm bilgisi ve yorum kısımları eklenmiş XML verisi kullanılır.

**İstek:**

```
POST /Animals HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <owl:Ontology
  rdf:about="http://localhost/roq-ns#Animals">
    <owl:versionInfo>$Revision: 1.3 $</owl:versionInfo>
    <rdfs:comment>
      This is an example ontology expressed in OWL for testing F-OWL
      inference rules. Some ontology concepts are adopted from
      http://
    </rdfs:comment>
  </owl:Ontology>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 201 Created
Location: http://localhost/Animals
```




Şekil 10. Uygulamanın testi için derlenmiş bir masaüstü uygulaması.

- **Ontolojinin Değiştirilmesi**

Burada bahsedilen konu, ontolojinin içeriğinin değil, sürüm bilgisi ve yorum kısımlarının değişmesidir.

**İstek:**

```
PUT /Animals HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <owl:Ontology
  rdf:about="http://localhost/roq-ns#Animals">
    <owl:versionInfo>$Revision: 1.3 $</owl:versionInfo>
    <rdfs:comment>
      This is an example ontology expressed in OWL for testing F-OWL
      inference rules. Some ontology concepts are adopted from
      http://
    </rdfs:comment>
  </owl:Ontology>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

- **Ontolojinin Silinmesi**

**İstek:**

```
DELETE /Animals HTTP/1.1
Host: http://localhost
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

Ontoloji isimleri üzerinde değişiklik yapılmamaktadır. Model isimleri ontoloji adı olacak şekilde tanımlandığından, bu şekilde bir kısıtlama oluşmaktadır. Jena, model isimlerinin değiştirilmesine izin vermemektedir. Ayrıca model isimlerinde, ontolojinin yerel ismi kullanılmaktadır (namespace öneki kullanılmaz). Ontoloji isimleri namespace ile birlikte kullanılırsa, istemciden gelen ontoloji isteği sorun olur. GET sorgularında sadece URL bilgisi olduğundan dolayı, namespace bilgisinin alınması mümkün olmamaktadır (Post bilgisi kullanılmaz). Web servisinin de stateless olduğu düşünülürse bir önceki POST isteği ile namespace bilgisinin gönderimi mümkün olmamaktadır.

Ontoloji verisi için aşağıdaki nitelikler PUT komutu ile güncellenebilmektedir.

- versionInfo
- comment

**Ontoloji Property Listeleme ve Oluşturma**

Bir instance için property tanımlamadan önce bu property'nin ontoloji içerisinde tanımlanmış olması gerekir. Adres satırında ontoloji/sınıf hiyerarşisinden farklı olarak ontoloji/property hiyerarşisi kullanılmıştır. Bu yöntem belli bir sınıfın (ontoloji/sınıf/instance-adı/property) property'sinden farklı olmasını sağlar.

| HTTP Komutu | Kullanılacak yol | Post bilgisi |
|---|---|---|
| **GET** | http://sunucu/{ontoloji-adı}/property<br>Ontolojideki tüm property'leri getirir. | Yok |
| **GET** | http://sunucu/{ontoloji-adı}/property/{property-adı}<br>Ontolojideki belirli bir property bilgisini getirir. | Yok |
| **POST** | http://sunucu/{ontoloji-adı}/property/{property-adı}<br>Yeni bir property ekler | Var |
| **PUT** | http://sunucu/{ontoloji-adı}/property/{property-adı}<br>Var olan bir property'i değiştirir. | Var |
| **DELETE** | http://sunucu/{ontoloji-adı}/property/{property-adı}<br>Tanımlı ontoloji property'sini siler. | Yok |

Çizelge 7. Property'lerin Sorgulanması ve Tanımlanması

- **Ontolojiye Ait Property'lerin Listelenmesi**

Adı geçen ontolojide kullanılabilen bütün DatatypeProperty ve ObjectProperty çeşitlerinin içerikleri OWL tanımı şeklinde listelenir.

**İstek:**

```
GET /Animals/property HTTP/1.1
Host: http://localhost
Accept: application/xml
```

**Yanıt:**

```
HTTP/1.1 200 OK
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <owl:ObjectProperty rdf:ID="hasAncestor">
    <rdfs:domain>Animal</rdfs:domain>
    <rdfs:range>Animal</rdfs:range>
  </owl:ObjectProperty>
…
  <owl:DatatypeProperty rdf:ID="age">
    <rdfs:range>nonNegativeInteger</rdfs:range>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="shirtsize">
    <rdfs:range>decimal</rdfs:range>
  </owl:DatatypeProperty>
</ResponseData>
```

- **Belirli bir Ontoloji Property İstemi**

Tüm property'lerin listelenmesi işleminde olduğu gibi, burada da belirli bir property içeriğinin gösterimi sağlanır.

**İstek**

```
GET /Animals/property/age HTTP/1.1
Host: http://localhost
Accept: application/xml
```

**Yanıt:**

```
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <owl:DatatypeProperty rdf:ID="age">
    <rdfs:range>nonNegativeInteger</rdfs:range>
  </owl:DatatypeProperty>
</ResponseData>
```

- **Yeni bir Ontoloji Property Oluşturma**

**İstek:**

```
POST /Animals/property/eyeColor HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <owl:DatatypeProperty rdf:ID="eyeColur">
    <rdfs:domain rdf:resource="#Person"/>
    <owl:equivalentProperty rdf:resource="#hasMother"/>
    <rdfs:comment>
      This property defines the glasses info for a person class
    </rdfs:comment>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:DatatypeProperty>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 201 Created
Location: http://localhost/Animals/property/eyeColor
```

POST işlemi uygulanırken ontoloji-adı/property-adı şeklinde değil ontoloji/property/property-adı şeklinde bir adres yolu kullanılır. Bu ontoloji/sınıf-adı adres yolundan ayırmak için kullanılan bir yöntemdir.
Bir property'nin DatatypeProperty mi yoksa ObjectProperty mi olduğunu post bilgisinde, <owl:DatatypeProperty> ya da <owl:ObjectProperty> öğesinin tanımlanmış olması gerekir.

ObjectProperty, bir nesnenin başka bir nesneyi değer olarak kullandığını belirtir. hasFriend adında bir property, Person domaini içerisinde başka bir Person sınıfından türemiş nesneyi işaret edebilir. Değer bilgisi tutan property'ler DatatypeProperty olarak tanımlanır. Literal bilgileri tutarlar. Bazen de XSD şeması içerisinde tanımlanan decimal, nonNegativeInteger gibi tanımlanmış değerleri de içerebilirler.

| | |
|---|---|
| **rdfs:domain** | Bir property'nin hangi sınıflara tanımlanabildiğini belirler |
| **rdfs:range** | Bir property'nin hangi türleri içerebileceğini belirler |
| **rdfs:comment** | Property ile ilgili notları tutar. |
| **rdf:type** | FunctionalProperty olup olmadığını belirtir. |

Çizelge 8. Desteklenen Property nitelikleri

- **Bir Ontoloji Property'sinin Değiştirilmesi**

**İstek:**

```
PUT /Animals/property/eyeColor HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <owl:DatatypeProperty rdf:ID="eyeColur">
    <rdfs:domain rdf:resource="#Animal"/>
    <owl:equivalentProperty rdf:resource="#hasMother"/>
    <rdfs:comment>
      This property defines the glasses info for a person class
    </rdfs:comment>
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
  </owl:DatatypeProperty>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

Değiştirme işlemlerinde belirtilen property öncelikle silinir. Daha sonra post bilgisi içerisinde tanımlamış bilgilere göre yeniden oluşturulur.

- **Ontoloji Property'sinin Silinmesi**

**İstek:**

```
DELETE /Animals/property/eyeColor HTTP/1.1
Host: http://localhost
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

**Ontoloji Sınıfları**

| HTTP Komutu | Kullanılacak yol | Post bilgisi |
|---|---|---|
| **GET** | http://sunucu/{ontoloji-adı}/class<br>Ontolojideki tüm sınıfları getirir. | Yok |
| **GET** | http://sunucu/{ontoloji-adı}/{class-adı}<br>Ontolojideki belirli bir sınıf bilgisini getirir. | Yok |
| **POST** | http://sunucu/{ontoloji-adı}/{class-adı}<br>Yeni bir sınıf ekler | Var |
| **PUT** | http://sunucu/{ontoloji-adı}/{class-adı}<br>Var olan bir sınıfı değiştirir. | Var |
| **DELETE** | http://sunucu/{ontoloji-adı}/{class-adı}<br>Tanımlı bir sınıfı siler. Burada alt sınıflar da bulunarak silme işlemi uygulanır. | Yok |

Çizelge 9. Sınıfların Sorgulanması ve Tanımlanması

- **Bir ontolojideki Sınıfların Listelenmesi**

**İstek:**

```
GET /Animals/class HTTP/1.1
Host: http://localhost
Accept: application/xml
```

**Yanıt:**

```
HTTP/1.1 200 OK
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <owl:Class rdf:ID="Animal"></owl:Class>
  <owl:Class rdf:ID="Male">
    <rdfs:subClassOf>Animal</rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Female">
    <rdfs:subClassOf>Animal</rdfs:subClassOf>
  </owl:Class>
…
  <owl:Class rdf:ID="TwoLeggedPerson"></owl:Class>
</ResponseData>
```

- **Belli Bir Sınıf Bilgisinin Alınması**

**İstek:**

```
GET /Animals/Person HTTP/1.1
Host: http://localhost
Accept: applicatino/xml
```

**Yanıt:**

```
HTTP/1.1 200 OK
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <owl:Class rdf:ID="Person">
    <rdfs:subClassOf>Animal</rdfs:subClassOf>
  </owl:Class>
</ResponseData>
```

- **Yeni Bir Sınıfın Oluşturulması**

**İstek:**

```
POST /Animals/Mammals HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <owl:Class rdf:ID="Mammals">
    <rdfs:subClassOf rdf:resource="#Animal"/>
    <rdfs:label>Mammals</rdfs:label>
    <rdfs:comment>A type of animal</rdfs:comment>
  </owl:Class>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 201 Created
Location: http://localhost/Animals/Mammals
```

- **Sınıfın Değiştirilmesi**

**İstek:**

```
PUT /Animals/Mammals HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <owl:Class rdf:ID="NewTypeMammals">
    <rdfs:subClassOf rdf:resource="#Animal"/>
    <rdfs:label>Mammals</rdfs:label>
    <rdfs:comment>A type of animal</rdfs:comment>
  </owl:Class>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

PUT işleminde ontoloji property ve ontolojiden farklı olarak, yeniden isimlendirme işlemi uygulanır. Var olan sınıf adı URL içinde kullanılır. Yeni sınıf adı ise rdf:ID bilgisinde post bilgisi ile gönderilir.

- **Sınıfın Silinmesi**

**İstek:**

```
DELETE /Animals/Mammals HTTP/1.1
Host: http://localhost
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

**Instance Oluşturma ve Sorgulama**

Bir instance oluşturulmadan önce, aynı model içerisinde tanımlanmış bir sınıfın olması gerekir. Oluşturulacak veya değiştirilecek instance için bir property tanımlanırken de bu property'nin o ontolojide tanımlı olması gerekir. Bu işlemde instance oluşturulurken, post bilgisinde gönderilen property'ler de ontoloji içerisinde tanımlanır.

| HTTP Komutu | Kullanılacak yol | Post bilgisi |
|---|---|---|
| **GET** | http://sunucu/{ontoloji-adı}/{class-adı}/instance | Yok |
| | Ontolojideki belirli bir sınıfın tüm instance'larını listeler. | |
| **GET** | http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı} | Yok |
| | Ontolojideki belirli instance'ın bilgilerini getirir. | |
| **POST** | http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı} | Var |
| | Yeni bir instance ekler(Adı geçen sınıf için). | |
| **PUT** | http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı} | Var |
| | Var olan instance'ı değiştirir. | |
| **DELETE** | http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı} | Yok |
| | Tanımlı bir instance'ı siler. | |

Çizelge 10. Instance Oluşturma ve Sorgulama

- **Bir Sınıfa Ait Instance'ların Listelenmesi**

**İstek:**

```
GET /Animals/Person/instance HTTP/1.1
Host: http://localhost
Accept: application/xml
```

**Yanıt:**

```
HTTP/1.1 200 OK
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <Person rdf:about="http://localhost/roq-ns#John">
    <age rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">37</age>
    <shirtsize rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">12</shirtsize>
    <shoesize rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">14</shoesize>
    <type rdf:resource="http://localhost/roq-ns#Person"/>
  </Person>
  <Person rdf:about="http://localhost/roq-ns#Mark">
    <age rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">18</age>
    <hasFather rdf:resource="http://localhost/roq-ns#John"/>
    <shirtsize rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">9</shirtsize>
    <shoesize rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">8</shoesize>
    <type rdf:resource="http://localhost/roq-ns#Person"/>
  </Person>
  …
</ResponseData>
```

- **Belli Bir Instance Bilgisinin Çekilmesi**

**İstek:**

```
GET /Animals/Person/Mark HTTP/1.1
Host: http://localhost
Accept: application/xml
```

**Yanıt:**

```
HTTP/1.1 200 OK
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <Person rdf:about="http://localhost/roq-ns#Mark">
    <age rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">18</age>
    <hasFather rdf:resource="http://localhost/roq-ns#John"/>
    <hasFather rdf:resource="http://localhost/roq-ns#JohnSmith"/>
    <shirtsize rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">9</shirtsize>
    <shoesize rdf:datatype="http://www.w3.org/2001/XMLSchema#decimal">8</shoesize>
    <type rdf:resource="http://localhost/roq-ns#Person"/>
  </Person>
</ResponseData>
```

- **Yeni Bir Instance Oluşturma**

**İstek:**

```
POST /Animals/Person/Michael HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <Person rdf:ID="Michael">
    <age rdf:datatype="xsd;nonNegativeInteger">18</age>
<hasFriend rdf:resource=" #John"/>
  </Person>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 201 Created
Location: http://localhost/Animals/Person/Michael
```

- **Instance'ın Değiştirilmesi**

**İstek:**

```
PUT /Animals/Person/Michael HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
< PostData>
  <Person rdf:ID="Michael">
    <age rdf:datatype="xsd;nonNegativeInteger">22</age>
<hasFriend rdf:resource=" #John"/>
  </Person>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

- **Instance'ın Silinmesi**

**İstek:**

```
DELETE /Animals/Person/Michael HTTP/1.1
Host: http://localhost
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

**Instance Property'si**

| HTTP Komutu | Kullanılacak yol | Post bilgisi |
|---|---|---|
| **GET** | http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı}/{property-adı} | Yok |
| | Instance'a ait belirtilen property bilgisi verir. | |
| **POST** | http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı}/{property-adı} | Var |
| | Belirtilen instance için yeni bir property ekler (ontolojide var olan bir property) | |
| **PUT** | http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı}/{property-adı} | Var |
| | Daha önce instance'a atanmış property bilgisini değiştirir. | |
| **DELETE** | http://sunucu/{ontoloji-adı}/{class-adı}/{instance-adı}/{property-adı} | Yok |
| | Belirtilen property, verilen instance bilgisinden silinir. | |

Çizelge 11. Instance Property Ekleme ve Sorgulama

- **Property Sorgulama**

**İstek:**

```
GET /Animals/Person/Mark/age HTTP/1.1
Host: http://localhost
Accept: application/xml
```

**Yanıt:**

```
HTTP/1.1 200 OK
<ResponseData xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/10/XMLSchema#" >
  <age rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">18</age>
</ResponseData>
```

- **Instance'a Yeni Bir Property Ekleme**

**İstek:**

```
POST /Animals/Person/Michael/hasSpouse HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <hasSpouse rdf:resource=" #Eve"/>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 201 Created
Location: http://localhost/Animals/Person/Michael/hasSpouse
```

- **Instance Property'sinin Değiştirilmesi**

**İstek:**

```
PUT /Animals/Mammals/hasSpouse HTTP/1.1
Host: http://localhost
Content-Type: application/x-www-form-urlencoded
<PostData>
  <hasSpouse rdf:resource=" #David"/>
</PostData>
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

- **Instance Property'sinin Silinmesi**

**İstek:**

```
DELETE /Animals/Person/Michael/hasSpouse HTTP/1.1
Host: http://localhost
```

**Yanıt:**

```
HTTP/1.1 200 OK
```

Sunucu üzerinden bir içerik istenirken kullanılan GET komutu ile beraber hangi türde verinin üretilmesi isteniyorsa HTTP paketi içinde tanımlanan "Accept" başlık bilgisinin de istemci tarafından belirtilmesi gerekir.

POST ve PUT komutları kullanıldığında gönderilecek içeriğin hangi türde olduğunu belirtmek amacıyla Content-Type HTTP başlığı kullanılır. Bu uygulamada XML formatı desteklendiğinden ve Unicode bir yapıda olduğundan dolayı “application/x-www-form-urlencoded” tanımlaması yapılmıştır. DELETE metodu kullanılırken herhangi bir veri ya da özel bir HTTP başlığı kullanılmaz.

DELETE ve PUT metotları sadece işlemin başarılı olduğuna dair “OK” yanıtı, GET metodu “OK” yanıtının yanında istenilen içeriği, POST metodu ise “Created” yanıtı ile beraber oluşturulan verinin URI bilgisini döndürür.

# 4. GERÇEKLEŞTİRİM

Uygulamanın çalışma süreci, istemciden isteklerin alınması ile başlayıp, bunların işlenmesi ve gerekli verilerin triple store'dan (Jena Veri tabanı) çekilmesi ile devam eder, en sonunda istenilen formatta verinin istemciye gönderilmesi ile biter. İstemci tarafından kullanılan isteklerle beraber gönderilen post bilgileri ve buna karşılık üretilecek sonuçlar web ontoloji dilinde (OWL) tanımlanmaktadır. Şekil 17'de bir isteğin gerçekleşmesinde hangi yolların izlendiği gösterilmektedir.

Bu tez çalışmasında, veri tabanı için MySQL, web sunucusu olarak Glassfish, uygulama geliştirme dili olarak Java, ontoloji modellemesi ve içeriğinin yönetilmesi için Jena Framework, istemcilerden gelen http isteklerinin yönlendirilmesi için Jersey JAX-RS, JSON çıktı alabilmek için json-lib kütüphanesi, kod geliştirme aracı olarak NetBeans uygulaması ve json-lib kütüphanesinin kullandığı bazı Apache[6] kütüphaneleri kullanılmıştır [23].

İstemciden gelen HTTP komutları alınır ve bu komutlar JAX-RS yorumlayıcısı (Jersey JAX-RS) aracılığı ile Java nesnesine iletilir. Bu uygulamada derlenen ROQHandler sınıfı burada, gelen istekleri nitelik bilgilerine göre ayrıştırır ve buna göre gerekli metotların tetiklenmesi sağlar. Bunun için tetiklenecek sınıfa ait Java kodunda niteliklerin tanımlanması gerekir.

JAX-RS için bir nitelik adı kullanılmadan önce, "@" işareti kullanılır.

### HTTP Komut Niteliği

Hangi HTTP komutunun, hangi metot tarafından işleneceğini ayrıştırmamıza olanak sağlar. Şekil 11, GET http komutunun, KaynakGetir metodu tarafından işleneceğini göstermektedir.

```
@GET
Private string KaynakGetir()
```

Şekil 11. GET Komutunun Tanımı

[6] http://commons.apache.org/

**URL Bilgilerinin Çekilmesi**

HTTP isteklerinde kullanılacak URL içeriğinde ayrılmış karakterler kullanılamayacağı için, ":,/" gibi karakterler ASCII karakter kodu karşılığı şeklinde (%20,%23 gibi) sunucuya gönderilebilir ya da o domain içerisinde bu karakterlerin ontoloji, sınıf, instance ya da property isimlerinde kullanılması engellenir.

@Path niteliği sınıflar ve içerisindeki metotlar için tanımlanabilir. Bir sınıftaki metotlar için bu nitelik tanımlanmadan önce, Şekil 12'de de gösterildiği gibi, bu sınıfın bu sınıfın hangi dizinden sorumlu olacağını belirten niteliğin tanımlı olması gerekir.

```
@Path("/kaynaklar")
Public class KaynakHandler{
```

Şekil 12. Java Sınıfının Path Tanımı

```
GET http://localhost/kaynaklar
```

isteği kullanıldığında JAX-RS, KaynakHandler sınıfının kaynaklar dizininden sorumlu olduğunu bilir ve bu sınıfın işlenmesi sağlanır.

Belli bir dizinin alt dizinlerinin işlenmesi görevini sınıfın metotlarına atamak istersek, yine bir @Path nitelik bilgisini yazmamız gerekir. Şekil 13'de bununla ilgili bir örnek verilmiştir.

```
@Path("/kaynaklar")
Public class KaynakHandler{
@Path("/ontoloji")
       Public String ListOntology(){
```

Şekil 13. Java Metodunun Path Tanımı

Bir metot URL bilgisindeki bir dizinin altında tanımlı tüm kaynakları işlemek istediğinde, istemci tarafından gelen herhangi bir kaynak adına ihtiyaç duyabilir. Bunun için nitelik tanımlanırken "{,}" işaretlemeleri kullanılıp, bu kaynak adının metodun parametresi olarak kullanılması sağlanır.

(1) http://localhost/ontoloji/egitim
http://localhost/ontoloji/insaat

1 numaralı örnekte gösterildiği gibi, ontoloji dizinin altındaki herhangi bir ismin belirli bir metot tarafından işleneceği düşünülürse, URL bilgisinde geçen bu ismin metoda aktarılması gerekir. Bunun için Şekil 14'deki gibi bir tanımlama yapılması gerekir. Bu örnekte "ontoloji-adi" olarak tanımlanmış nitelik, metot içerisinde String (dizgi) türünde "ontologyName" adında bir değişken olarak kullanılır.

```
@Path("/ontoloji")
Public class KaynakHandler{
@Path("/{ontoloji-adi}")
       Public String ListOntology(@PathParam("ontology-adi") final String
ontologyName){
```

Şekil 14. Path Bilgisinin Parametre Olarak Alınması

**Desteklenen Veri Türleri**

@Produces niteliği, bir metodun hangi formatlarda veri döndüreceğini tanımlar. Aynı zamanda @Consumes niteliği o metodun hangi formatta POST bilgisi alabileceğini belirtir.

```
@Produces("text/html")
```

Bu tez çalışmasında geliştirilen uygulama, XML (application/xml), HTML (text/html) ve JSON (application/json) formatlarında çıktı oluşturabilmektedir.

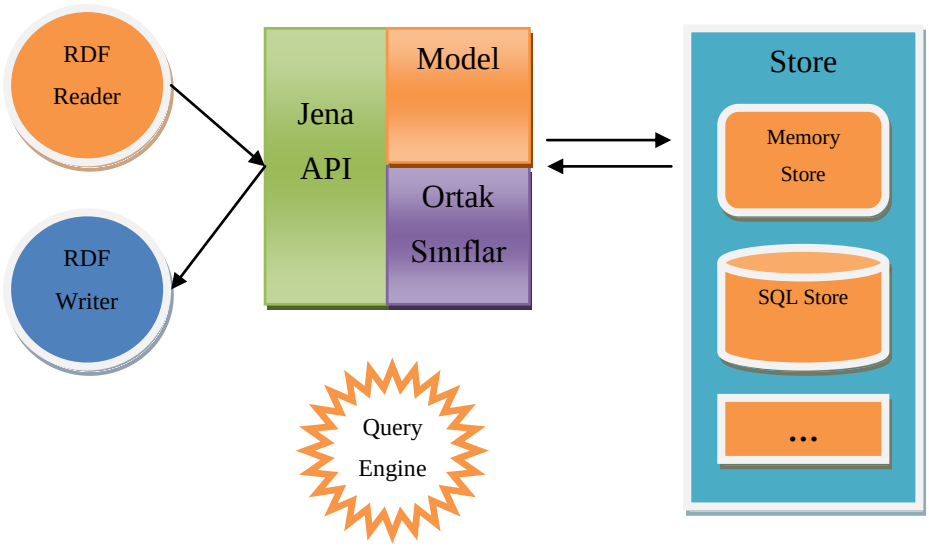
**Jena Framework**

Jena, semantik web için geliştirilen ve Java tabanlı oluşturulmuş bir kütüphanedir. Jena'nin sağladığı API (Uygulama Geliştirme Arayüzü), semantik veri modellemesinde bir altyapı olarak kullanılır. Jena tarafından sağlanan Model sınıfı, bir ontolojinin tanımı ve sorgulanması için oluşturulmuştur [24]. Bir ontoloji oluşturulmak istendiğinde buna uygun bir modelin var olması gerekir. Şekil 15'te ontoloji oluşturmak için gerekli bir Model nesnesinin nasıl tanımlandığı gösterilmektedir.

```
Model base = maker.createModel(ontoModelName, true);

        OntModel ontModel = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM,
base);
```

Şekil 15. Model Nesnesinin Oluşturulması




Şekil 16. Jena Framework

Jena Framework verilerin saklanması için bir veri dosyası ya da veri tabanı kullanabilir. Şekil 16'da genel olarak Framework'ün barındırdığı yapı görülmektedir.

Geliştirilen web servisi 4 adet sınıf görev almaktadır. Her birinin sorumlu olduğu işler aşağıda açıklanmıştır. Şekil 18'de bunlara ait sınıf diyagramı verilmiştir.

**OntModelFactory**

Jena üçlü (triple) yapıları tutmak için model nesnesini kullanır. Bütün sınıflar ve bunlara ait instance'lar bu model nesnesi altında toplanır. Jena'da OWL yapısını destekleyen özelleşmiş model yapısı olan OntModel nesnesi vardır. Bu sınıf OntModel nesnesinin veri tabanı üzerinde tutulmasını ve gerektiğinde dinamik hafızaya getirilmesini sağlar.

**OntologyManager**

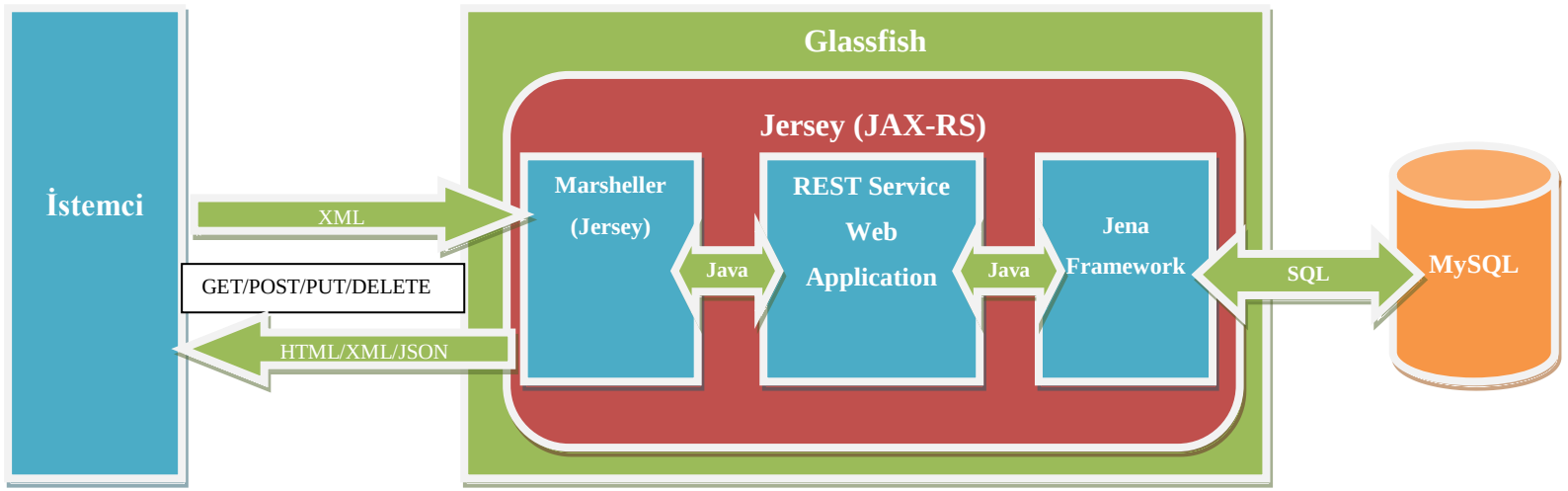
OntModelFactory sınıfının sağladığı model yapısını kullanarak ontolojilerin, bu ontolojiye ait sınıf, property ve instance'ların oluşturulması, listelenmesi, değiştirilmesi ya da silinmesi gibi işlemleri yapar.

**ROQHandler**

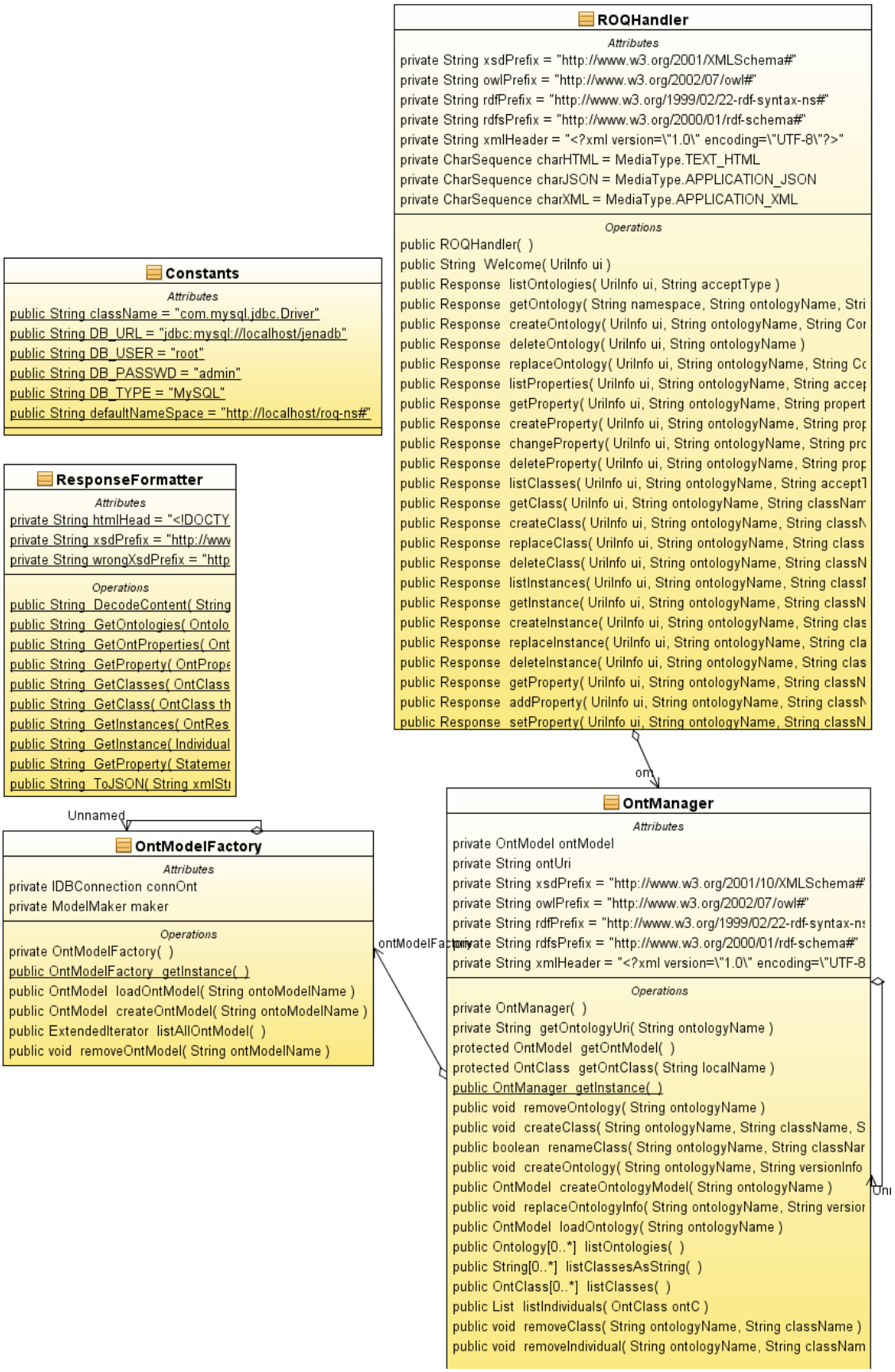
RESTful Ontology Querying Handler'ın kısaltması olan ROQHandler, http isteklerinin ilk işlendiği sınıftır. Bu sınıfı içerisinde web servisinde kullanılacak http komutları ve hangi URL adresleri ile kullanılacakları tanımlıdır. Ayrıca bu sınıf içerisinde her metodun hangi türde POST bilgisi kabul ettiği ve hangi formatta veri döndüreceği bilgisi vardır. Bu sınıf sadece URL ayrıştırması yaparak, hangi türde verinin işleneceğini belirler. Buna göre OntologyManager sınıfından gerekli metot tetiklenir.

**ResponseFormatter**

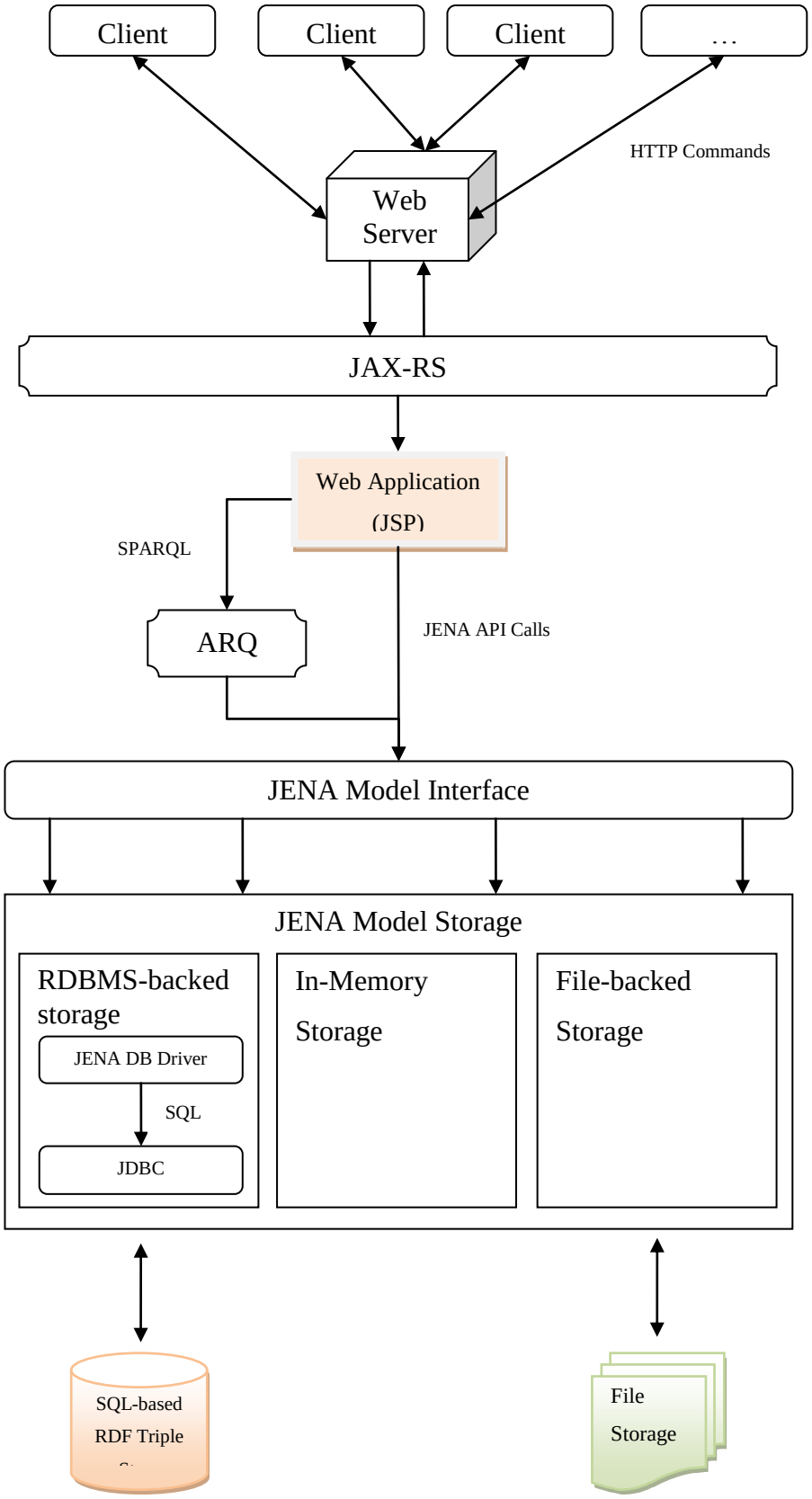
İstemciye gönderilecek veri formatı bu sınıf içerisinde oluşturulur. Bir ontolojinin tamamı istendiğinde RDF/XML gibi bir yapı ortaya çıkartılabilirken, bir sınıf listesi veya belirli bir instance için bilgi istendiğinde bunun istemci tarafından kabul edilen formata dönüştürülmesi gerekir. Bu işi yapmakla görevli olan ResponseFormatter sınıfı HTML ve XML oluşturmak için basit string (dizgi) metotlarını kullanır. JSON formatı olarak çıktı istendiğinde ise, üretilmiş XML içeriği JSON serializer kütüphanesi kullanılır.




Şekil 17. Bir İstemin Gerçekleşmesi




Şekil 18. Uygulama Sınıf Diyagramı




Şekil 19. Sistem Mimarisi

# 5. SONUÇ VE GELECEK ÇALIŞMALAR

Bu tez çalışmasında, semantik web için şema görevi görecek ontolojilerin ve oluşturulmasını ve yönetilmesini sağlayacak RESTful bir web servisi geliştirilmiştir. Bu web servisinin özelliği jenerik olması, herhangi bir ontoloji yapısı ile çalışabilmesidir. Geliştirilen web servis protokolü istenen her türlü ontoloji yapısı değişikliğini yapmayı, örneğin yeni sınıflar (class) ya da özellikler (property) eklemek veya silmek gibi, sağlamakta, ayrıca bu ontolojilere veri ekleme, çıkarma, değiştirme ve güncelleme (CRUD) de yapılabilmektedir. Bu altyapı semantik web tabanlı uygulama geliştirmek isteyenler için uygun bir yapıdır. Uygulamalar bu arayüz ile hızlı bir şekilde geliştirilebilir, uygulama programcıları semantik web altyapısını bilmeden yalnızca bu arayüzle çalışarak veri yönetimini gerçekleştirebilirler. Bu arayüzle geliştirilen sistemler açık sistem özelliği de sağlayabilirler, verileri dış sistemlere aynı arayüzle açarak verilerin diğer uygulamalar tarafından paylaşılmasını da sağlayabilirler.

Bu web servisi altyapısı geliştirilirken bilinen en yaygın kütüphane olan Jena Framework kullanılmıştır. Bir veri tabanı ile desteklenen bu yapı, kalıcı olarak ontolojilerin saklanması ve gerektiğinde hafıza üzerinde hızlı bir şekilde işlem yapılmasına olanak sağlamıştır.

Gelecek çalışmalarda, ontoloji ve içerdiği kavramlar çağırılırken namespace kullanımının desteklenmesini sağlayacak bir yapı ortaya çıkarılması hedeflenmektedir. Ayrıca alt property ve alt sınıf yapısının kullanımı da desteklenecektir. Ayrıca performans çalışmaları da yapılacak ve "linked data"[7] türü veriler üzerinde hızlı sorgulama ve güncelleme teknikleri geliştirilecektir.

[7] www.linkeddata.org

## KAYNAKLAR


[1] "Tim Berners Lee", erişim adresi: http://tr.wikipedia.org/wiki/Tim_Berners-Lee, erişim tarihi: 30 Ağustos 2010.

[2] Antoniou Gregoris, Van Harmelen Frank, A Semantic Web Premier,MIT Press, 2004.

[3] "Extensible Markup Language (XML) 1.0 (Fifth Edition)" erişim adresi: http://www.w3.org/TR/REC-xml/, erişim tarihi:10 Ağustos 2010.

[4] Battal, Abdullah, 2009, Semantik Web İle Geliştirilen Bir Televizyon Programı Öneri Sistemi, Yüksek Lisans Tezi, TOBB ETÜ Fen Bilimleri Enstitüsü, Ankara

[5] "RDF Primer" erişim adresi: http://www.w3.org/TR/rdf-primer/, erişim tarihi:30 Ağustos 2010.

[6] Powers, Shelly, Practical RDF,O'Reilly Media,Temmuz 2003.

[7] "RDF Schema" erişim adresi: http://www.w3.org/TR/rdf-schema/, erişim tarihi: 30 Ağustos 2010.

[8] "OWL Web Ontology Language Overview" erişim adresi: http://www.w3.org/TR/owl-features/, erişim tarihi: 30 Ağustos 2010.

[9] "SPARQL Protocol For RDF" erişim adresi: http://www.w3.org/TR/rdf-sparql-protocol/, erişim tarihi: 30 Ağustos 2010.

[10] Hebeler, John, Fisher, Matthew Blace, Ryan, Perez-Lopez, Andrew, Semantic Web Programming, Wiley Publishing, 2009.

[11] "SPARQL Update" erişim adresi: http://www.w3.org/Submission/SPARQL-Update/, erişim tarihi 30 Ağustos 2010.

[12] "Representational State Transfer (REST)" erişim adresi: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, erişim tarihi: 27 Ağustos 2010.

[13] Sandoval, Jose, RESTful Java Web Services, Apress, Kasım 2009.

[14] "Freebase" erişim adresi: http://www.freebase.com, erişim tarihi: 27 Ağustos 2010.

[15] "JSON" erişim adresi: http://www.json.org, erişim tarihi: 27 Ağustos 2010.

[16] "The Protégé Ontology Editor and Knowledge Acquisition System" erişim adresi: http://protege.stanford.edu, erişim tarihi: 28 Ağustos 2010.

[17] Lathem, Jon, Gomadam,Jon Karthik, Sheth, Amit P. SA-REST and (S)mashups : Adding Semantics to RESTful Services, International Conference on Semantic Computing, 469-476, Eylül 2007.

[18] Alowisheq, Areeb, Millard, David E., EXPRESS: EXPressing REstful Semantic Services, 2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology – Workshops, 453-456, Eylül 2009.

[19] Alarcon, Rosa, Wilde, Erik, From RESTful Services to RDF: Connecting the Web and the Semantic Web, UC Berkeley School of Information Report 2010-041, Haziran 2010.

[20] DeLeon, Alexander, Dumontier, Michel, Publishing OWL ontologies with Presto, 2008.

[21] Burke, Bill, RESTful Java with JAX-RS, O'Reilly, Kasım 2009.

[22] Breitman,Karin Koogan, Casanova, Marco Aantonio,Truszkowski, Walter, Semantic Web: Concepts, Technologies and Applications, Springer Science+Business Media, 2004.

[23] "JSON-Lib" erişim adresi: http://json-lib.sourceforge.net/, erişim tarihi 29 Ağustos 2010.

[24] "Jena Semantic Web Framework" erişim adresi: http://jena.sourceforge.net, erişim tarihi: 31 Ağustos 2010.

[25] Chakrabarti, Sujit Kumar, Kumar, Prashant, Test-the-REST: An Approach to Testing RESTfulWeb-Services, 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009.

# ÖZGEÇMİŞ

**Kişisel Bilgiler**


| | |
|---|---|
| Soyadı, Adı: | MABOÇOĞLU,Abdulhamit |
| Uyruğu: | T.C |
| Doğum tarihi ve yeri: | 20.10.1982, Rize |
| Medeni Hali: | Bekar |
| Telefon: | 0 (506) 815 53 53 |
| Email: | mabocoglu@hotmail.com |


**Eğitim**

| Derece | Eğitim Birimi | Mezuniyet Tarihi |
|---|---|---|
| Lisans | Çankaya Üniversitesi, Bilgisayar Müh. | 2007 |

**İş Deneyimi**

| Yıl | Yer | Görev |
|---|---|---|
| 2006-2007 | BMBSoft | Software Developer |
| 2007-2008 | TOBB Ekonomi ve Teknoloji Üniversitesi | Araştırma Görevlisi |
| 2008- | Türk Telekom Genel Müdürlüğü | Uzman Yardımcısı |

**Yabancı Dil**

İngilizce