Lehrveranstaltung

Semantic Web Technologien

WS 2009/10

HTWG Konstanz

RDF(S) Frameworks

Programmierungs-Frameworks für RDF(S)

Seit der Einführung des Resource Description Frameworks sind bereits einige Jahre ins Land gegangen.

- Diverse Organisationen (Universitäten, Firmen...) haben praktische Forschung in Richtung Semantic Web betrieben
- Viele Programmier-Frameworks sind dabei entstanden

- Framework hier:
  - Eher Programmbibliotheken
  - Teilweise schon in Richtung Triple-Stores
  - Oft stark erweiterbare Baukästen
  - Meist auf Java basierend
    - Gibt es aber auch für alle anderen Sprachen
  - Zum größten Teil Open Source Projekte

Sesame

Programmierungs-Frameworks für RDF(S)

Sesame

- Im Februar 2000 wurde das On-To-Knowledge Forschungs-Projekts der EU ins Leben gerufen
  - Ziel: Entwicklung von Tools und Methoden für das Semantic Web
  - Beteiligung vieler Unternehmen und Organisationen
- Die Firma Aduna legt in diesem Rahmen den Grundstein von Sesame
- Als Middleware zur Speicherung und zum Abruf von RDF(S) basierter Daten und Metadaten
- Seit dem Ende von On-To-Knowledge führt Aduna Sesame als Open-Source-Projekt weiter
- Basiert auf Java-Technologien
- Großer Funktionsumfang

http://www.openrdf.org/

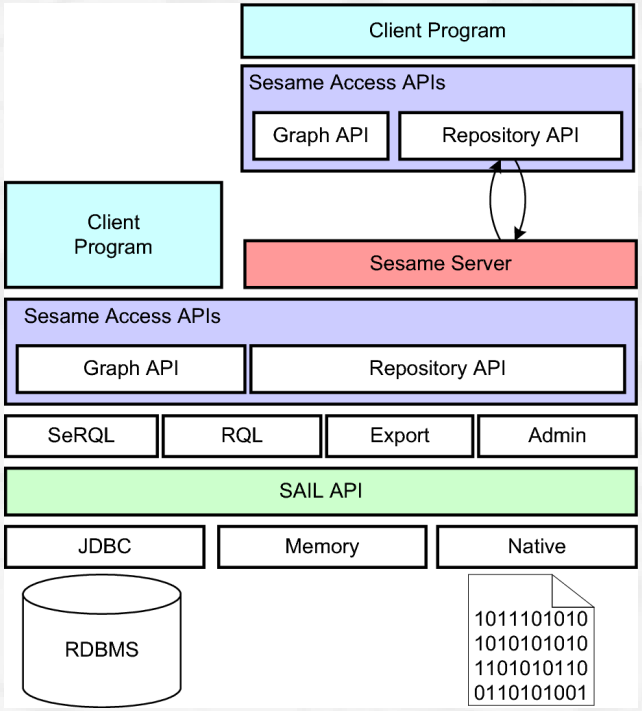
Programmierungs-Frameworks für RDF(S)

Sesame

- Aktuell zwei Versionsreihen (1.x und 2.x)
  - 1.x war noch stark auf die Verwendung als Server ausgerichtet
  - 2.x (seit Ende 2007) stärker zum Einsatz als Bibliothek
  - Rückwärtskompatibilität musste aufgegeben werden
  - Sesame 2.x nutzt viele Features von Java 5 (typsichere Collections...)
  - Sesame 2.x führt Unterstützung für Kontext und Transaktionen ein

Sesame

- Vollständig modularer Aufbau
  - (Fast) Jede Komponente kann ausgetauscht, erweitert oder den persönlichen Vorlieben angepasst werden
  - Modularität wird durch Vielzahl Interface-basierter APIs erreicht



Programmierungs-Frameworks für RDF(S)

Sesame

- Wichtigstes API: SAIL
  - Storage and Inference Layer
  - Diese Schicht sorgt für die persistierung der Daten und bietet Reasoning-Dienste an
  - Durch SAIL-API wird vollständige Abstraktion erreicht
  - Für obenliegende Schichten ist es (fast) vollkommen egal, auf welche Art die Daten letztendlich gespeichert werden

Programmierungs-Frameworks für RDF(S)

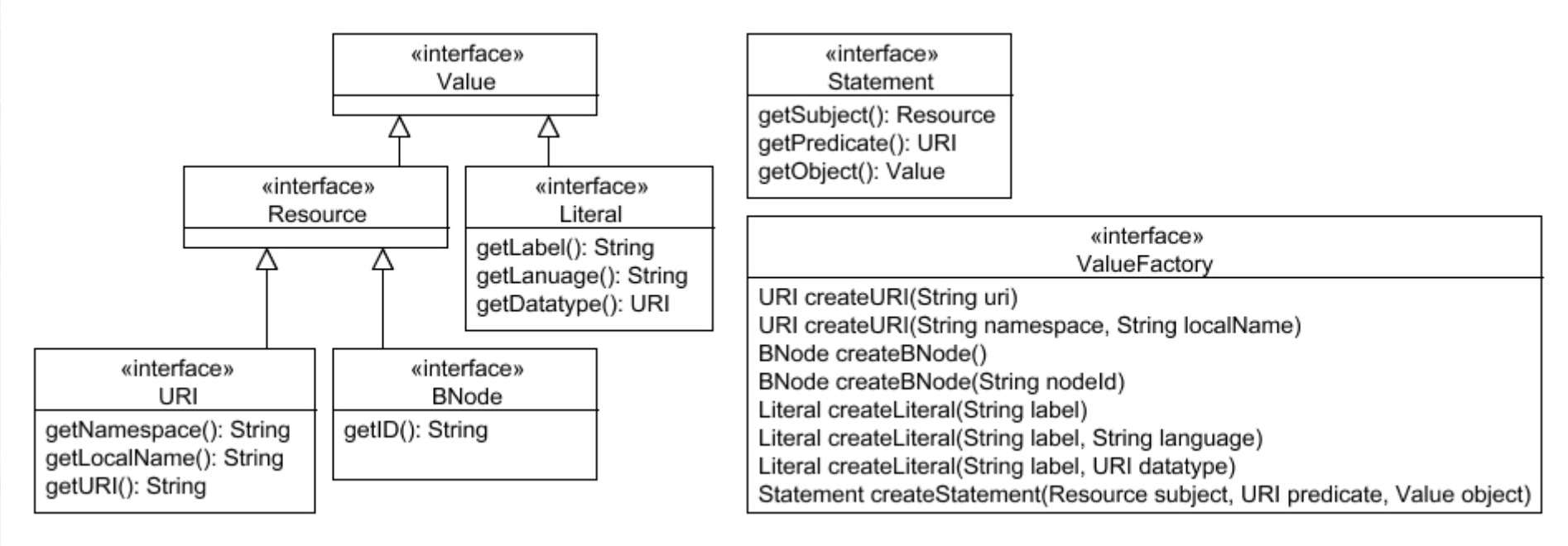
Sesame

- Weitere Layer
  - Query-Layer bietet diverse Query-Engines für verschiedene Anfragesprachen (z.B.: SPARQL)
  - Repository API
    - Zentraler Zugriffspunkt auf Sesame Repositories
    - Inhalte können abgerufen/aktualisiert werden
    - Sowohl Lokale als auch Remote Repositories ansprechbar
      - API kümmert sich um Client-Server-Kommunikation
    - In Version 2.x stark ausgebaut und um Kontextmechanismen (woher kommt dieses Triple) und Transaktionen erweitert
  - Graph-API (RDF-Modell in Sesame 2.x)
    - Programmatisches Arbeiten mit Triplen (siehe nächste Folien)

Programmierungs-Frameworks für RDF(S)

Sesame

- RDF-Model von Sesame 2.x



Programmierungs-Frameworks für RDF(S)

- Sesame – Arbeiten mit dem Graph API

```
// Erstelle einen Graph
Graph myGraph = new org.openrdf.model.impl.GraphImpl( );

// Ressourcen werden mit einer ValueFactory erstellt
ValueFactory myFactory = myGraph.getValueFactory( );
String namespace = “http://www.example.com/family#”;

// Erstellen wir zunächst benötigte Ressourcen
URI birgit = myFactory.createURI( namespace, “Birgit” );
URI hasName = myFactory.createURI( namespace, “hasName” );
Literal name = myFactory.createLiteral( “Birgit” );

// Nun können wir dieses Triple zum Graph hinzufügen
myGraph.add( birgit, hasName, name );

// Aussage können auch direkt an Ressourcen angehängt werden (1.x)
URI personClass = myFactory.createURI( namespace, “Person” );
URI rdfType = myFactory.createURI( org.openrdf.vocabulary.RDF.TYPE );
birgit.addProperty( rdfType, personClass );
```

Sesame – Weitere Eigenschaften

- Sesame liefert eine Webapplikation zur Administration des Servers in Form einer WAR-Datei
  - Deployment in einem Servlet-Container, wie zum Beispiel Tomcat
- REST-Webservice
  - Zugriff mit HTTP GET, POST, DELETE, PUT
  - Client kann bei Anfrage angeben, in welchem Format er die Antwort wünscht:
    - RDF/XML, N-Triples, Turtle, N3, TriX, …
  - Antworten auf komplexe Queries können im SPARQL Query Result Format in XML oder JSON Form oder als binäre RDF-Ergebnis-Tabellen erfolgen
- Große Community steht hinter Sesame.
  - Sesame als Backend für Ontologie-Editor Protégé
  - Sesame als Backend für Semantic Desktop Nepomuk
  - Sesame-Jena-Integrations-Schicht basierend auf SAIL
  - Wrapper / Portierungen für PHP, Python, Perl, Ruby, C# ...
  - ...

Mulgara Semantic Store

Mulgara Semantic Store

- Fork des Kowari-Projektes
- Ziel: Implementierung hochskalierbarer transaktionssicherer Triple-Store-Engine nicht basierend auf relationalen Datenbanken (siehe Sesame, Jena), sondern mit eigenem rein Java-basiertem Storage-Layer
- Open-Source-Projekt
- Sehr aktiv (Releases etwa alle 2 Monate)
- Basiert auf Java-Technologien
- Großer Funktionsumfang

http://www.mulgara.org

Programmierungs-Frameworks für RDF(S)

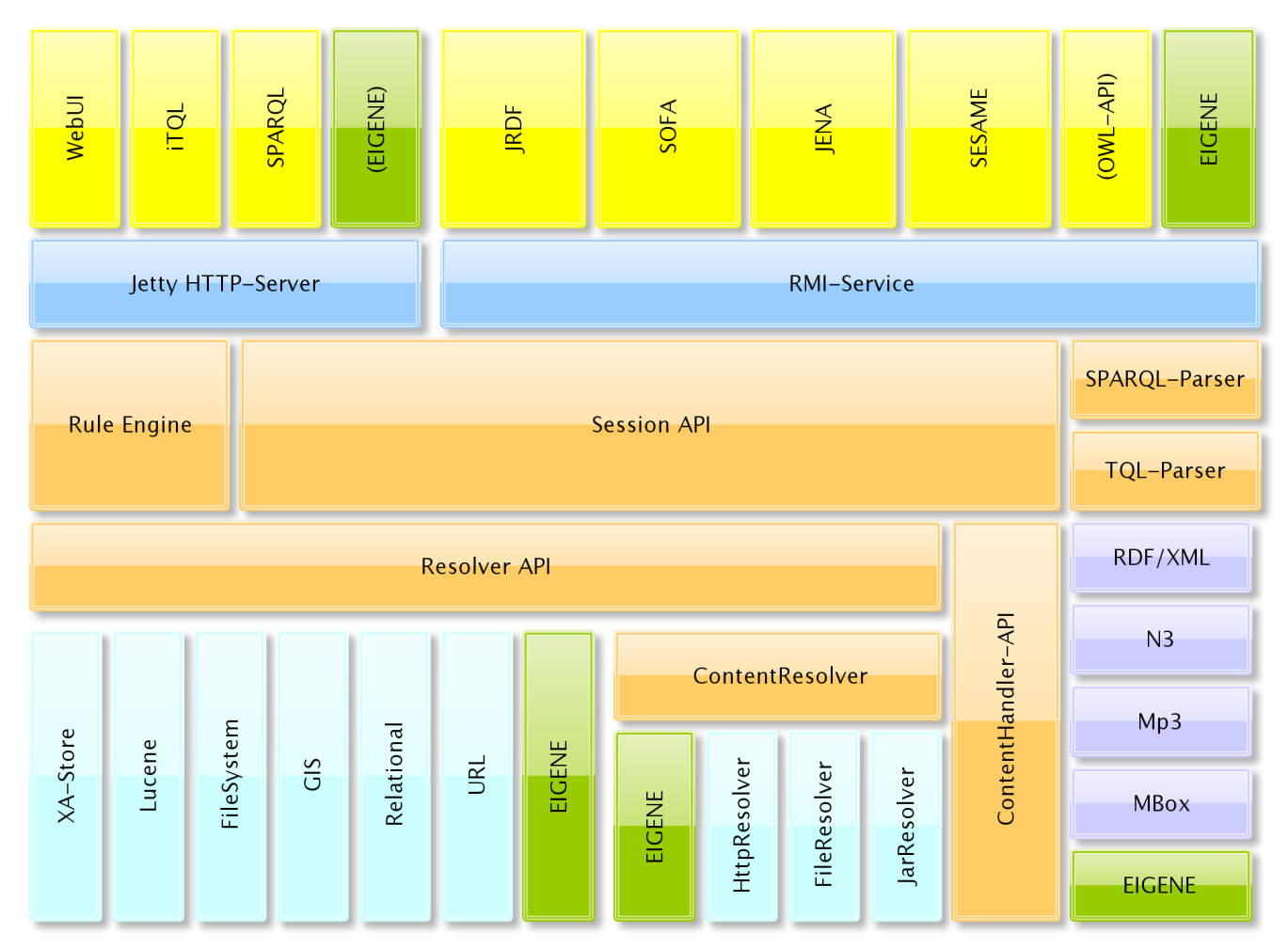
Mulgara Semantic Store

- Mulgara ist mehr als ein Triple-Store:
  - Mulgara kann per RMI angesprochen werden
  - Per SAIL als Storage für Sesame verwendbar
  - Integrierter HTTP-Server
    - SOAP Webservice
    - SPARQL-Protokoll-Endpunkt
    - Oberfläche zum einfachen Ausführen von Anfragen
  - Parser für diverse Query-Sprachen und RDF-Serialisierungen
  - Mächtige Integrationsschnittstelle Resolver-API
  - Eigene Anfragesprache iTQL
  - Eigenes RDF-API: JRDF (stark an Sesames Graph-API angelehnt)

Programmierungs-Frameworks für RDF(S)

Mulgara Semantic Store

- Mulgara Architektur:



Programmierungs-Frameworks für RDF(S)

- Mulgara – Arbeiten mit JRDF

```
// Erstelle einen Graph (speicherbasiert)
Graph myGraph = new GraphImpl( );

// Elemente werden über eine ElementFactory erstellt
GraphElementFactory myFactory = myGraph.getElementFactory( );
String namespace = “http://www.example.com/family#”;

// Erstellen wir zunächst benötigte Ressourcen
URIReference birgit = myFactory.createResource( new URI( namespace,
                                                         “Birgit” ) );
URIReference hasName = myFactory.createResource( new URI( namespace,
                                                         “hasName” );
Literal name = myFactory.createLiteral( “Birgit” );

// Das fügen wir nun zu einem Triple zusammen
Triple statement = myFactory.createTriple( birgit, hasName, name );

// Das Statement kann nun zum Graph hinzugefügt werden
myGraph.add( statement );
```

Mulgara Semantic Store – Modelltypen in Mulgara

- Werden bei Erzeugung des Models übergeben
- Standard mulgara:Model
  - Normales Model auf eigenem (XA) Store
  - XA Store performante Implementierung basierend auf B-Bäumen
  - dreiteilig aufgebaut:
    - Node Pool: Für jede Ressource eine eindeutige numerische ID
    - String Pool: Bildet Ids des NP auf eigentliche Daten ab
    - Statement Store: Verknüpfungen zwischen Ids des NP
- mulgara:ViewModel
  - Temporärer Graph der kombinierte Sicht auf mehrere Graphen gleichzeitig bietet
  - Kombination von Graphen als Vereinigung / Schnittmenge
  - Schreiben nicht möglich
- mulgara:Lucene
  - Volltext-Index-Modell basierend auf Suchmaschine Lucene
  - Literale als Objekt werden direkt indiziert
  - URIs als O werden als URL behandelt und abgerufen/Inhalt indiziert
  - Anfragen hierbei mit Unschärfe möglich (Groß/Klein/Wortteile...)

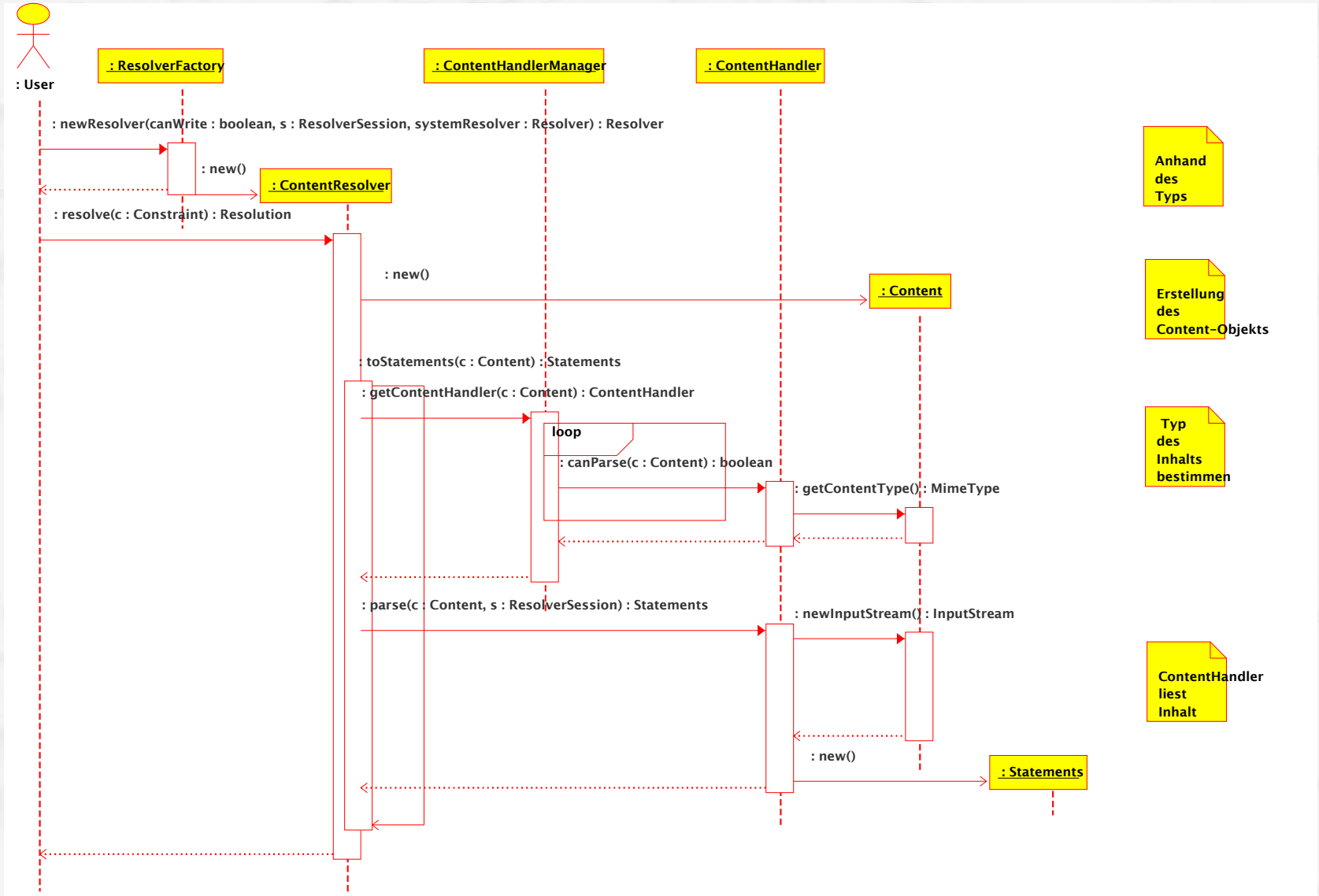
Mulgara Semantic Store – Modelltypen in Mulgara

- mulgara:XMLSchemaModel
  - Datentyp-Graph
  - Enthalten konzeptuelle jede wahre Aussage für einen Datentyp
  - Beispielsweise 1<2, Heiligabend ist vor Sylvester …
  - Prädikate mulgara:lt, mulgara:gt, mulgara:after, mulgara:before
- mulgara:TypeModel
  - Zur Typisierung von RDF-Knoten (Resource,Literal,B-Node)
  - Ermöglicht in Anfragen, das etwa nur Literale geliefert werden
- mulgara:FileSystemModel
  - Informationen über das Dateisystem
    - Lese-/Schreibberechtigungen
    - Dateigrößen
    - Besitzer
    - Funktionieren ähnlich wie mulgara:View

Mulgara Semantic Store – Resolver SPI

- Schnittstelle zur Integration externer Inhalte
- Plugin-fähig
- Protokollbasiert wird ein passender Resolver gewählt
- Resolver kümmert sich um das Empfangen von Inhalten
- Verarbeitung von Inhalten durch ContentHandler
- ContentHandler wird anhand des Mime-Type der Daten ermittelt
- Passender ContentHandler liefert dann die Triple
    - Contenthandler bereits vorhanden für
        - RDF/XML
        - Mailbox
        - MP3
        - ...

Mulgara Semantic Store – Resolver SPI



Mulgara Semantic Store – Topaz Project

- http://www.topazproject.org
- Java-Bibliothek zur RDF-basierten Speicherung und Arbeit mit Objekten
- Orientiert sich sehr stark an ORM-Technologien (Object-Relational-Mapping)
- Vorbild: Hibernate Framework
- Idee:
  - Persistente Daten auf dem Triple-Store sollen auf klassenbasierte Objekte abgebildet werden
- Objektorientierte Konzepte wie Vererbung, Komposition und Assoziation werden berücksichtigt
- Zusätzlich: OQL Object Query Language – Anfragen basierend auf Objekten anstatt auf Triplen

Programmierungs-Frameworks für RDF(S)

- Topaz – Object-Triple-Mapping
  Bezug zu SW-Elementen wird per Annotation hergestellt

```
@Entity(graph="family", types={"ex:Person"})
public class Person
{
    private URI id;
    private String name;
    …

    @Id
    public void setId( URI id ){ this.id = id; }
    public URI getId( ) { return this.id; }

    @Predicate(uri="ex:hasName")
    public void setName( String name ){ this.name = name; }
    public String getName( ){ return this.name; }

    ...
}
```

Programmierungs-Frameworks für RDF(S)

- Arbeiten mit Topaz Objekten

```
Session session = sessFactory.openSession( );
Transaction txn = session.beginTransaction( );

URI birgitUri = URI.create( family + “Birgit” );

// Objekt erstellen
Person birgit = new Person( birgitUri );
birgit.setName( “Birgit” );
birgit.setWohnort( “Allensbach” );
…

// Objekt in den TripleStore schreiben
session.saveOrUpdate( birgit );
…

// Objekt aus dem Store abrufen und auf dessen Eigenschaften zugreifen
Person person2 = session.get( Person.class, familiy+”Stephan” );
System.out.println( person2.getWohnort( ) );
```

Jena

Programmierungs-Frameworks für RDF(S)

Jena (Seit Herbst 2009: OpenJena)

- Zunächst von Hewlett Packard im Rahmen des Semantic Web Research Forschungsprogramm (etwa 2000) entwickelt
- Als Open Source Software freigegeben worden
- Mittlerweile marktführendes Framework für die Implementierung Semantik-gestützter Applikationen
- Riesiger Funktionsumfang
- Einfach zu erlernendes API
- Hervorragend dokumentiert (für ein Open Source Projekt)
- Große Community mit Erweiterungen

http://www.openjena.org
http://jena.sourceforge.net

Jena – Data Abstraction Layer

- DAL basiert auf JDBC (Java DB Connectivity)
- Einheitlicher Zugriff auf relationale Datenbanken zur persistenten Speicherung von Triplen
- Gleiche Verwendung von Schnittstellen bei persistentem und speicherbasiertem Modell
- Direkt von Jena unterstützte DBMS:
  - MySQL, HSQLDB, PostgreSQL, Oracle, MS SQL
- Weitere Implementierung (z.B: IBM DB2) von Drittanbietern

Jena – Import / Export von Graphen

- Große Menge unterstützter Formate:
  - RDF/XML (ausführlich) und RDF/XML-ABBREV (abgekürzt)
  - N3 / N-TRIPLE / TURTLE
  - …
- RDF Parser ARP ist eigenständiges Modul
  - Offene Schnittstelle
  - Keine Abhängigkeiten zum Rest von Jena
    => auch für andere Frameworks verwendbar

Jena – Inferenz API

- Integration einer Vielzahl von Reasonern
- Einige Reasoner bereits in Jena vorhanden:
  - z.B. Zur einfachen aber effizienten Schlussfolgerung über die Properties rdfs:subPropertyOf und rdfs:subClassOf
  - Aber auch andere die schon den Umgang mit einfachen OWL-Konstrukten ermöglichen
- Über das API wird eine mächtige Reasoning-Engine für benutzerdefinierte Regeln zur Verfügung gestellt

Jena – Weitere interessante Features

- Verfolgen von Änderungen mit ModelChangedListener
  - Interface
  - Kann einfach implementiert und beim Model angemeldet werden
  - Listener-Pattern wie z.B: beim AWT / Swing
  - Methoden werden dann bei Veränderungen des Models aufgerufen
- SPARQL-Implementierung ARQ
  - Unterstützt Standard-Anfragen
    - SELECT, CONSTRUCT, DESCRIBE und ASK
  - Erweitert SPARQL um Modifikations-Statements (SPARQL/Update):
    - INSERT, DELETE …
- Diverse Kommandozeilen-Tools
  - Queries von der Kommandozeile aus
  - Schemagen: Ontology-Files → Java-Source (Konstanteninterfaces)
- Und noch vieles mehr ...

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Modell erstellen und füllen

```
// Erstellen eines leeren RDF – Modells
Model model = ModelFactory.createDefaultModel( );

// Festlegen einiger URIs
String familyUri = "http://example.com/family/";
String ontology = "http://example.com/ontology#";

// Erstellen einiger Eigenschaftstypen
Property hatVorname = model.createProperty( ontology + "hasFirstName" );
Property hatWohnort = model.createProperty( ontology + "livesInCity" );
Property istVaterVon = model.createProperty( ontology + "isFatherOf" );
Property istSchwesterVon =model.createProperty( ontology+"isSisterOf" );

// Erstellen der Resource "Birgit" (http://example.com/family/Birgit)
Resource birgit = model.getResource( familyUri + "Birgit" );
birgit.addProperty( hatVorname , "Birgit" );
birgit.addProperty( hatWohnort , "Allensbach" );

// Erstellen der Resource "Stephan" (http://example.com/family/Stephan)
Resource stephan = model.getResource( familyUri + "Stephan" );
stephan.addProperty( hatVorname , "Stephan" );
stephan.addProperty( hatWohnort , "Syrgenstein" );
```

Programmierungs-Frameworks für RDF(S)

```
// Erstellen der Resource "Hans" (http://example.com/family/Hans)
Resource hans = model.createResource( familyUri + "Hans" );
hans.addProperty( hatVorname , "Hans" );
hans.addProperty( hatWohnort , "Syrgenstein" );


// Aussagen, dass Birgit die Schwester von Stephan ist
birgit.addProperty( istSchwesterVon, stephan );

// Aussagen, dass Hans Vater von Birgit ist
hans.addProperty( istVaterVon, birgit );

// Aussagen kann man auch direkt so erstellen:
Statement statement = model.createStatement( hans,istVaterVon,stephan );
// Man muss das Statement dann aber separat zum Model hinzufügen
model.add( statement );

// Zur Massenverarbeitung auch mit Arrays...
Statement[] statements = new Statement[1];
statements[0] = statement;
model.add( statements );

// … sowie mit Listen
List list = new ArrayList( );
list.add( statement );
model.add( list );
```

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Modell untersuchen

```
// Gib mit alle Elemente die Vater von jemandem sind
// da Subjekte immer Ressourcen sind, bekommen wir einen ResIterator
ResIterator parents = model.listSubjectsWithProperty( istVaterVon );
while (parents.hasNext()) {
  Resource person = parents.nextResource();
  // Gib die URI der Ressource aus
  System.out.println(person.getURI());
}

// Gib mir alle Elemente, von denen jemand Vater ist
// Da Objekte Literale oder Ressourcen sein können, bekommen wir einen
// Node Iterator zurück
NodeIterator children = model.listObjectsOfProperty( istVaterVon );

// Gib mir alle Elemente von denen Birgit Schwester ist
NodeIterator siblings = model.listObjectsOfProperty( birgit,
                                                     istSchwesterVon );

// Oder frag Birgit selbst (In dem Fall bekommen wir einen StmtIterator)
StmtIterator moreSiblings = birgit.listProperties( istSchwesterVon );
```

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Modell untersuchen 2

```
// Finden eines bestimmten Statements (Ist eine Aussage vorhanden)
model.listStatements( birgit, istSchwesterVon, stephan );

// Gib mir alle Statements mit birgit als Subjekt, Stephan als Objekt
model.listStatements( birgit, null, stephan );

// Gib mir alle Statements über Stephan
model.listStatements( birgit, null, stephan );

// Gib mir alle Statements mit der istVaterVon Eigenschaft
model.listStatements( null, istVaterVon, null );
```

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Modelle persistieren mit Model.write()

```
// Einfache Ausgabe des Models nach System.out in RDF/XML-Form
model.write( System.out );


// Anstelle von System.out können auch andere Output-Streams stehen
File file = new File( filename );
FileOutputStream fos = new FileOutputStream( file );
model.write( fos );


// Ausgabe mit abgekürzter RDF/XML Syntax
model.write( fos, "RDF/XML-ABBREV" );


// Ausgabe im N-Triple-Format mit Basis-URL für relative URIs
model.write( fos, "N-TRIPLE", "http://www.example.org/" );

// Vordefinierte Formate: RDF/XML, RDF/XML-ABBREV, N-TRIPLE, TURTLE, N3
```

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Modelle lesen mit Model.read( )

```
// Zunächst müssen wir ein Model erstellen
Model model = ModelFactory.createDefaultModel( );

// FileManager verwenden um eine Eingabedatei zu finden
InputStream is = FileManager.get().open( inputFileName );
// es sollte überprüft werden ob  is != null ist

// Nun können wir problemlos lesen (ohne Behandlung relativer URIs)
model.read( is, null );

// Optional kann wieder ein RDF-Format angegeben werden
model.read( is, null, "N3" );

// Vordefinierte Formate: RDF/XML, RDF/XML-ABBREV, N-TRIPLE, TURTLE, N3
// RDF/XML-ABBREV ist hier gleichbedeutend mit RDF/XML
// null bedeuted wieder RDF/XML
```

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Modelle verschmelzen

```
// Zunächst erstellen wir uns 3 Modelle
Model model1 = ModelFactory.createDefaultModel( );
Model model2 = ModelFactory.createDefaultModel( );
Model model3 = ModelFactory.createDefaultModel( );

// Lesen wir die Informationen aus 2 Dateien
model1.read( new InputStreamReader( fileName1 ), "" );
model2.read( new InputStreamReader( fileName2 ), "" );

// Nun können wir die zwei Modelle einfach vereinen
model3 = model1.union( model2 );

// Und das verschmolzene Modell wieder ausgeben
model.write( System.out, "RDF/XML-ABBREV" );
```

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Arbeiten mit RDF-Containern

```
// Erstellung eines Bag
Bag steiners = model.createBag( );

// Iterator für hatWohnort-Statements, bei denen Ort mit “stein” endet
StmtIterator iter = model.listStatements(
    new SimpleSelector( null, hatWohnort, (RDFNode) null ){
        public boolean selects( Statement s )
        {
            return s.getObject().toString().endsWith( “stein” );
        } } );

// holen wir uns alle Subjekte des Iterators und fügen sie der Bag hinzu
while( iter.hasNext( ) ){
    steiners.add( iter.nextStatement( ).getSubject( ) );
}

// Die Ausgabe sieht in etwa folgendermaßen aus:
// …
// <rdf:Description rdf:nodeID=”A3”>
//    <rdf:type rdf:resource=”http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag” />
//    <rdf:_1 rdf:resource=”http://www.example.com/family/Stephan” />
//    <rdf:_2 rdf:resource=”http://www.example.com/family/Hans” />
//    …
// </rdf:Description>
```

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Arbeiten mit Persistenz

```
// DB Parameter einstellen
String M_DB_URL  = "jdbc:mysql://localhost/jenatest";
String M_DB_USER = "testuser";
String M_DB_PASS = "testpass";
String M_DB      = "MySQL";
String M_DBDRIVER_CLASS = "com.mysql.jdbc.Driver";
// DB Treiber Klasse laden
Class.forName( M_DBDRIVER_CLASS );
// Connection zur Datenbank erstellen
IDBConnection connection = new DBConnection( M_DB_URL, M_DB_USER,
                                             M_DB_PASS, M_DB );

// ModelMaker für die Connection erstellen
ModelMaker maker = ModelFactory.createModelRDBMaker( connection );

// Standard Model erstellen ...
Model defModel = maker.createDefaultModel();

// ... oder existierendes Modell öffnen
Model existingModel = maker.openModel();

// alles weitere wie bisher – Jena kümmert sich um die Persistenz
```

Programmierungs-Frameworks für RDF(S)

- Jena Beispiel – Arbeiten mit Persistenz
- Benannte Modelle:

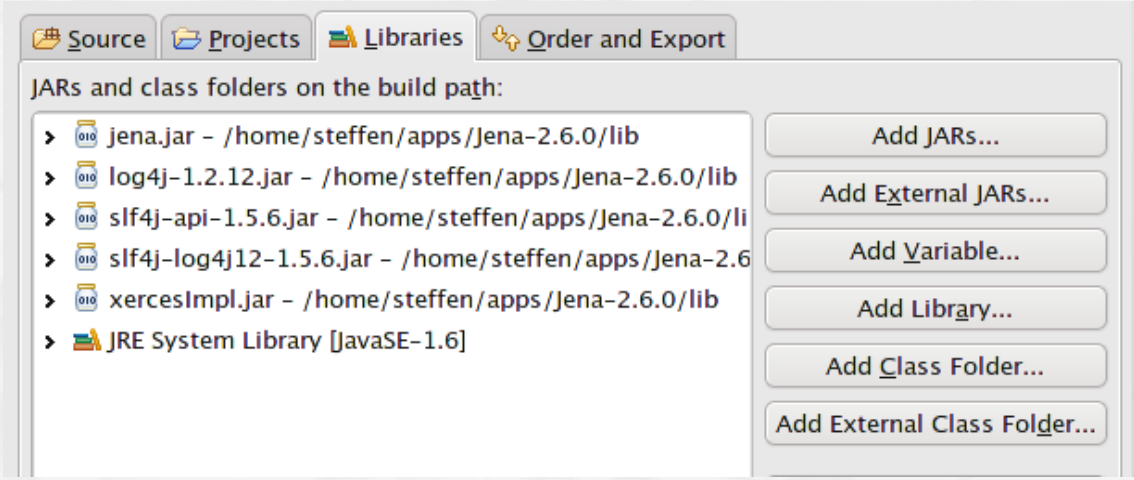
```
// benanntes Modell erstellen
Model namedModel = maker.createModel( “MyNamedModel” );

// existierendes benanntes Modell öffnen
Model previousNamedModel = maker.openModel( “MyStoredNamedModel” );
```

```
// Prüfen ob Model auf Datenbank existiert
ModelRDB model;
if( !connection.containsModel( modelName ) )
    model = ModelRDB.createModel( connection, modelName );
else
    model = ModelRDB.open( connection, modelName );
```

Programmierungs-Frameworks für RDF(S)

Jena – Ist auf den Rechnern im Pool F033 installiert

- Zum Programmieren müssen einfach die nötigen JAR-Files aus dem lib Unterverzeichnis von Jena eingebunden werden:
  - jena.jar
  - log4j-*.jar
  - slf4j-*.jar
  - slf4j-log4j*-*.jar
  - xercesImpl.jar
- Unter Eclipse:
  - Project Properties
  - Java Build-Path
  - Libraries
  - Add External JARs
- Happy Hacking !



Programmierungs-Frameworks für RDF(S)

Noch Fragen ?

Programmierungs-Frameworks für RDF(S)

- Literatur:
  - Buch “Semantic Web Grundlagen”, Springer Verlag 2008
    Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, York Sure
    ISBN: 978-3-540-33993-9
  - Sesame Dokumentation
    http://www.openrdf.org/documentation.jsp
  - Mulgara Tutorial / Dokumentation
    http://docs.mulgara.org/
    http://www.mulgara.org/trac/wiki
  - Jena RDF-API Tutorial
    http://openjena.org/tutorial/RDF_API/index.html
  - Umfangreiches Tutorial von IBM zu Jena:
    http://www.ibm.com/developerworks/xml/library/j-jena/
  - JENA API-Docs
    http://jena.sourceforge.net/javadoc/index.html