Lehrveranstaltung

Semantic Web Technologien

WS 2009/10

HTWG Konstanz

SPARQL

SPARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage

SPARQL – Einführung

- SPARQL Protocol and RDF Query Language
  - Offizielle W3C Recommendation vom 15. Januar 2008
  - SPARQL (engl. ausgesprochen wie sparkle)
  - Anfragen auf RDF Graphen
  - Bestandteile
    - Query Language / Anfragesprache
      - Wichtigster Bestandteil
      - Hohe Relevanz in der Praxis
    - SPROT – SPARQL Protocol for RDF
      - Protokoll zur Übertragung von SPARQL Anfragen an einen SPARQL Query Processing Service und zum Empfang von Ergebnissen durch den Anfragesteller
    - SPARQL Query Results XML Format
      - XML Dokumentformat zur Darstellung von Ergebnissen von SPARQL SELECT und ASK Queries

SPARQL Query Language

SPARQL – Einführung

- Einfache Anfragen:
  - Daten:

```
@prefix ex: <http://www.example.org/> .
ex:Volkswagen    ex:hatFirmensitzIn     “Wolfsburg”.
ex:Audi          ex:hatFirmensitzIn     “Ingolstadt”.
```

  - Anfrage:

```
PREFIX ex: <http://www.example.org>
SELECT ?firmensitz WHERE
{
    ?x ex:hatFirmensitzIn ?firmensitz .
}
```

  - Resultat:

firmensitz
“Wolfsburg”
“Ingolstadt”

SPARQL – Einführung

- Mit mehreren Variablen:
  - Daten:

```
@prefix ex: <http://www.example.org/> .
_:a    ex:hatFirmensitzIn     “Wolfsburg” .
_:a    ex:hatNamen            “Volkswagen” .
_:b    ex:hatFirmensitzIn     “Ingolstadt” .
_:b    ex:hatNamen            “Audi” .
_:c    ex:hatFirmensitzIn     “Zuffenhausen” .
```

  - Anfrage:

```
PREFIX ex: <http://www.example.org>
SELECT ?name ?firmensitz WHERE
{
   ?x ex:hatFirmensitzIn ?firmensitz .
   ?x ex:hatNamen ?name .
}
```

  - Resultat:

name	firmensitz
“Volkswagen”	“Wolfsburg”
“Audi”	“Ingolstadt”

SPARQL – Einführung

- Hauptbestandteile des SPARQL SELECT :
  - Anfragemuster (`WHERE`)
    - Verwendet TURTLE Syntax
    - Kann Variablen enthalten (`?firmensitz` …)
    - Muster muss vollständig passen, damit Ergebnisse zurückgeliefert werden können
  - URIs können mit `PREFIX` abgekürzt werden
  - Ergebnis durch “Auswahl” von Variablen (`SELECT`)

SPARQL – Einführung

- Anfragemuster:
  - Einfache Graphmuster
  - Menge von RDF Triplen in Turtle Syntax
    - Auch Turtle Abkürzungen durch ; und , erlaubt
  - Variablen werden durch ? Oder $ eingeleitet
    - **`?firmensitz`** gleichbedeutend mit **`$firmensitz`**
    - Prinzipiell auch innerhalb ein und der selben Anfrage mischbar
    - Bitte für eine Variante entscheiden, da sonst schwer bis nicht lesbar
  - Variablen können an Stelle von Subjekt, Prädikat oder Objekt stehen

SPARQL – Behandlung von Literalen

- Literale
  - Daten:

```
ex:w   ex:p     "test" .
ex:x   ex:p     "test"@de .
ex:y   ex:p     "test"^^xsd:string .
ex:z   ex:p     "42"^^xsd:integer .
```

  - Query

```
SELECT ?node WHERE { ?node ex:p “test” . }
```

Liefert nur

node
ex:w

Genaue Übereinstimmung gefordert

  - Aber Ausnahme bei numerischen Typen:

```
SELECT ?node WHERE { ?node ex:p 42 . }
```

Liefert

node
ex:z

Datentyp wird aus syntaktischer Form bestimmt

SPARQL – Behandlung von Blank Nodes

- Blank Nodes (gekennzeichnet durch _:xyz)
  - In Anfragen
    - Können an Stelle von Subjekt oder Objekt stehen
    - ID beliebig (aber nur einmal pro Anfrage verwenden)
    - Variablen, die nicht ausgewählt werden können
  - In Ergebnissen
    - Platzhalter für unbekannte Elemente
    - Ids beliebig, aber evtl. an anderen Ergebnisteil gebunden
  - Beispiel (basierend auf Daten des letzten Beispiels):

```
SELECT ?node ?name WHERE { ?node ex:hatNamen ?name . }
```

  - Ergebnis kann unterschiedliche Formen annehmen:

node	name
_:a	“Volkswagen”
_:b	“Audi”

node	name
_:b	“Volkswagen”
_:a	“Audi”

SPARQL – Gruppierung

- Gruppierung
  - Anfragemuster können durch { und } zu Gruppen zusammengefasst und verschachtelt werden
  - Beispiel:

```
SELECT ?name ?firmensitz WHERE
{
    {   ?x ex:hatFirmensitzIn ?firmensitz .
        ?x ex:hatNamen ?name . }
    {}
    ?x ex:hatGruendungsjahr .
}
```

  - Aber ohne zusätzliche Konstrukte nicht sonderlich sinnvoll

SPARQL – Optionale Muster

- OPTIONAL:
  - Angabe optionaler Teile eines Musters

```
SELECT ?x ?name ?firmensitz WHERE
{
    ?x rdf:type ex:Autohersteller .
    OPTIONAL { ?x ex:hatFirmensitzIn ?firmensitz . }
    OPTIONAL { ?x ex:hatNamen ?name . }
}
```

  - Ergebnis kann teilweise ungebunden sein:

x	name	firmensitz
ex:Volkswagen	“Volkswagen”	“Wolfsburg”
ex:Audi		“Ingolstadt”
ex:Porsche	“Porsche”	

praktisch, wenn nicht alle Daten vorhanden sind, man aber auch nicht unbedingt alle Daten benötigt.

SPARQL – Alternative Muster

- UNION
  - Angabe alternativer Teile eines Musters

```
SELECT ?x ?firmensitz WHERE
{
   ?x rdf:type ex:Autohersteller .
   { ?x ex:hatFirmensitzIn    ?firmensitz . } UNION
   { ?x ex:hatHauptstandortIn ?firmensitz . }
}
```

  - Ergebnis ist die Vereinigung (logisches ODER) der Ergebnismengen beider möglichen Varianten
  - Gelten beide Varianten (Sind für einen Autohersteller beide Triple gepflegt) werden auch beide Varianten zurückgeliefert
  - Gleiche Variablennamen in beiden Teilen von UNION beeinflussen sich nicht gegenseitig.

SPARQL – Vorrangregelung

- Kombination von UNION und OPTIONAL

```
SELECT ?x ?firmensitz ?name WHERE {
   ?x rdf:type ex:Autohersteller .
   { ?x ex:hatFirmensitzIn    ?firmensitz . } UNION
   { ?x ex:hatHauptstandortIn ?firmensitz . } OPTIONAL
   { ?x ex:hatNamen ?name . } }
```

  - OPTIONAL gilt immer genau für das Muster (die Mustergruppe), vor dem OPTIONAL steht
  - UNION und OPTIONAL sind gleichwertig und beziehen sich beide immer auf alle links von ihnen stehende Ausdrücke
  - Beispiel

```
{ { s1 p1 o1 } OPTIONAL { s2 p2 o2 } UNION { s3 p3 o3 }
              OPTIONAL { s4 p4 o4 } OPTIONAL { s5 p5 o5 } }
```

```
{ { { { { s1 p1 o1 } OPTIONAL { s2 p2 o2 }
      } UNION { s3 p3 o3 }
    } OPTIONAL { s4 p4 o4 }
  } OPTIONAL { s5 p5 o5 }
}
```

SPARQL - FILTER

- FILTER
  - Graphmuster-Vergleich führt zu einer Menge von Ergebnissen, bei der jedes Ergebnis (in unseren Beispielen jede Tabellenzeile) eine Menge an Variablen liefert, die an eine oder mehrere RDF Elemente gebunden sind
  - Mit der SPARQL Anweisung FILTER kann man die Menge der Ergebnisse weiter einschränken
  - FILTER wird eine Operation übergeben, die als Ergebnis einen boolschen Wert liefert
  - Nur Ergebnisse, für die der Wert TRUE zurückgeliefert wird, werden in die Ergebnismenge übernommen

SPARQL – FILTER – Wofür?

- Warum benötigt man FILTER?

  Manche Anfragen mit reinem Mustervergleich nicht möglich:

  - Welche Personen sind zwischen 18 und 23 Jahre alt?
  - Der Vorname welcher Personen beginnt mit einem “A”
  - Welche Attribute sind in deutscher Sprache vorhanden?

  Hierbei helfen uns FILTER weiter

SPARQL – FILTER – Beispiel

- Beispiel zu FILTER:

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price WHERE
{
   ?x ns:price ?price .
   FILTER (?price < 30.5)
    ?x dc:title ?title .
}
```

  - Schlüsselwort FILTER, gefolgt von Ausdruck in Klammern
  - Filter liefern Wahrheitswerte
  - Viele Filterfunktionen aus Xquery/XPath Standard übernommen

SPARQL – FILTER – Operatoren

- Vergleichsoperatoren: <, <=, =, >=, >, !=
  - Vergleich von Literalen gemäß “natürlicher” Reihenfolge
    - Numerisch nach Wert
    - Strings alphabetisch
    - Boolean (TRUE > FALSE)
    - …
  - Unterstützung für:
    - Numerische Datentypen (xsd:int …)
    - xsd:dateTime
    - xsd:string
    - xsd:Boolean
  - Andere Typen und RDF-Elemente unterstützen nur = und !=
  - Vergleich von Literalen inkompatibler Typen nicht möglich
    - z.B: xsd:integer und xsd:string
- Arithmetische Operatoren: +,-,*,/
  - Nur für numerische Typen
  - Verwendung zur Berechnung von Werten in Filterbedingungen

SPARQL – FILTER – Funktionen

- RDF spezifische Filterfunktionen in SPARQL

Operator	Beschreibung	Erlaubter Typ für A
BOUND(A)	Gibt an, ob die Variable A gebunden ist	Variable
isIRI(A) / isURI (A)	Gibt an, ob das Element eine IRI/URI ist	RDF-Element
isBLANK(A)	Gibt an, ob das Element ein BlankNode ist	RDF-Element
isLITERAL(A)	Gibt an, ob das Element ein Literal ist	RDF-Element
STR(A)	Darstellung von URIs und Literalen als xsd:string	URI / Literal
LANG(A)	Sprachangabe eines Literals (leer, wenn nicht vorhanden)	Literal
DATATYPE(A)	Datentyp eines Literals. Wenn Literal ungetypt und ohne Sprachangabe: xsd:string	Literal

SPARQL – FILTER – Funktionen

- Weitere RDF spezifische Filterfunktionen in SPARQL

Operator	Beschreibung
sameTERM(A, B)	Prüfen, ob A und B die gleichen RDF Terme sind
LangMATCHES(A,B)	Prüfen, ob A und B die gleichen Sprachangaben sind
REGEX(A, B)	Prüfen, ob in String A der Reguläre Ausdruck B vorkommt

- Mehrere Filter können mittels && bzw. || verbunden werden sowie mit ! negiert werden

SPARQL – Lösungen und Modifikatoren

- SELECT Anfragen generieren unsortierte Folgen von Ergebnissen
- Zum Teil sind auch unerwünschte Ergebnisse enthalten
- Manche Sätze können mehrfach vorhanden sein

- Hier kommen Modifikatoren zum Einsatz
  - Solution Sequence Modifiers
- Modifikatoren bieten uns Möglichkeiten zur
  - Sortierung: ORDER BY
  - Entfernung doppelter Ergebnisse: DISTINCT, REDUCED
  - Aufteilung durch LIMIT und OFFSET

SPARQL – ORDER BY

- `ORDER BY`
  - Sortierung von Ergebnismengen
  - Variablen, nach denen sortiert werden soll, werden nacheinander aufgeführt
  - Mit `ASC( )` und `DESC( )` kann aufsteigend bzw. absteigend sortiert werden
  - URIs werden alphabetisch als Zeichenketten sortiert
  - Reihenfolge zwischen unterschiedlichen Elementarten:

    ungebundene Variable < leerer Knoten < URIs < RDF Literale

- Beispiel:

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX ex:      <http://www.example.org/>
SELECT ?name
       WHERE { ?x foaf:name        ?name .
               ?x   ex:employeeId ?empID . }
       ORDER BY ?name DESC(?empID)
```

SPARQL – DISTINCT / REDUCED

- DISTINCT / REDUCED Beispiel:

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:x    foaf:name   "Alice" .
_:x    foaf:mbox   <mailto:alice@example.com> .
_:y    foaf:name   "Alice" .
_:y    foaf:mbox   <mailto:asmith@example.com> .
_:z    foaf:name   "Alice" .
_:z    foaf:mbox   <mailto:alice.smith@example.com> .
```

```
SELECT ?name
       WHERE { ?x foaf:name ?name }
```

name
“Alice”
“Alice”
“Alice”

```
SELECT DISTINCT ?name
       WHERE { ?x foaf:name ?name }
```

name
“Alice”

```
SELECT REDUCED ?name
       WHERE { ?x foaf:name ?name }
```

name
“Alice”

V

name
“Alice”
“Alice”

V

name
“Alice”
“Alice”
“Alice”

SPARQL – LIMIT und OFFSET

- `LIMIT`
  - Nur eine bestimmte Anzahl an Ergebniszeilen ausgeben
  - Definition durch Maximalwert
  - 0: Keine Ergebnisse, negative Werte nicht erlaubt
- `OFFSET`
  - Ergebnisausgabe erst ab angegebener Zeile beginnen
  - Offset von 0 hat keinen Effekt
- `LIMIT` und `OFFSET` machen nur Sinn in Verbindung mit `ORDER BY`, da sonst Reihenfolge nicht vorhersagbar
- Beispiel:

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT  ?name
        WHERE   { ?x foaf:name ?name }
        ORDER BY ?name
        LIMIT 5  OFFSET 10
```

  - Nur fünf Ergebniszeilen liefern, beginnend ab der elften (3. Seite)

SPARQL – Modifikatoren Reihenfolge

- Reihenfolge der Modifikatoren bei der Verarbeitung:

  1. Sortierung anhand von `ORDER BY`
  2. Entfernung nicht ausgewählter Variablen
  3. Entfernung doppelter Ergebnisse (`DISTINCT`)
  4. Entfernung der Ergebniszeilen vor dem `OFFSET`
  5. Entfernung der überschüssigen Zeilen nach `LIMIT`

- Hinweis:
  - `ORDER BY` wird **vor Entfernung** nicht ausgewählter Variablen ausgeführt
    → Sortierung auch nach nicht ausgewählten Variablen möglich

SPARQL – Anfragetypen

- Anfragetypen in SPARQL
  - Bereits kennengelernt: **SELECT**
    - Liefert als Ergebnis eine Tabelle mit den Variablenwerten
    - Einfach sequentiell zu verarbeiten
    - Struktur und Semantische Verknüpfung der Werte geht leider verloren
  - **CONSTRUCT**
    - Liefert einen RDF-Graph, der durch Ersetzung von Variablen in einer Menge von Triple-Templates konstruiert wird
  - **ASK**
    - Liefert einen Wahrheitswert, der aussagt, ob ein Anfrage-Muster in einem Graph gefunden werden kann oder nicht
  - **DESCRIBE**
    - Liefert einen RDF-Graph, der gefundene Ressourcen beschreibt

SPARQL – CONSTRUCT

- CONSTRUCT
  - Liefert einen RDF-Graph als Rückgabewert
  - Auch praktisch, um Teilgraphen zu extrahieren
  - Sequentielle Verarbeitung des Ergebnisses kompliziert
  - Ungebundene Variablen können nicht behandelt werden
  - Beispiel:

Daten

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:a    foaf:name   "Alice" .
_:a    foaf:mbox   <mailto:alice@example.org> .
```

Anfrage

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT   { <http://example.org/person#Alice> vcard:FN ?name }
WHERE       { ?x foaf:name ?name }
```

Ergebnis

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
<http://example.org/person#Alice> vcard:FN "Alice" .
```

SPARQL – ASK

- ASK
  - Prüft, ob ein Muster in einem Graph vorkommt
  - Rückgabe ist ein boolscher Wert (true/false)
  - Beispiel:

Daten

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
_:a  foaf:name       "Alice" .
_:a  foaf:homepage   <http://work.example.org/alice/> .
_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

Anfrage

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" }
```

Ergebnis

```
true
```

SPARQL – DESCRIBE

- DESCRIBE
  - Liefert einen RDF-Graph, der die in der Anfrage angegebenen Ressourcen beschreibt
  - Nicht genau spezifiziert
    - Was beschrieben wird, obliegt dem SPARQL Prozessor
  - Beispiel:

Anfrage

```
PREFIX ent:  <http://org.example.com/employees#>
DESCRIBE ?x WHERE { ?x ent:employeeId "1234" }
```

Ergebnis

```
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix vcard:  <http://www.w3.org/2001/vcard-rdf/3.0> .
@prefix exOrg:  <http://org.example.com/employees#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:    <http://www.w3.org/2002/07/owl#>

_:a     exOrg:employeeId    "1234" ;
        foaf:mbox_sha1sum   "ABCD1234" ;
        vcard:N
         [ vcard:Family       "Smith" ;
           vcard:Given        "John"  ] .
foaf:mbox_sha1sum  rdf:type  owl:InverseFunctionalProperty
```

SPARQL – Abkürzungen

- Es kann ein leerer PREFIX definiert werden
- Mit BASE kann ein Standard-Prefix festgelegt werden

```
BASE    <http://example.org/book/>
PREFIX  : <http://example.org/ontology>
SELECT  $title WHERE   { <book1>  :title  ?title }
```

- rdf:type kann mit “a” abgekürzt werden

```
SELECT  $x WHERE   { $x a <ex:Book> }
```

- RDF-Collections können mit ( ) abgekürzt werden

```
      (1 ?x 3) :p "w" .
```

Ist gleichbedeutend mit

```
_:b0  rdf:first  1 ;
      rdf:rest   _:b1 .
_:b1  rdf:first  ?x ;
      rdf:rest   _:b2 .
_:b2  rdf:first  3 ;
      rdf:rest   rdf:nil .
_:b0  :p         "w" .
```

SPARQL – RDF Dataset

- Viele RDF Data stores bieten die Möglichkeit, Triple über mehrere Graphen zu verteilen und Informationen zu jedem einzelnen Graph aufzunehmen
  - Eine Menge von Graphen nennt man RDF Dataset
- SPARQL Queries bieten die Möglichkeit, Anfragen über mehrere Graphen verteilt zu stellen
- Jedes RDF-Dataset enthält
  - **immer einen "Default" Graph** ohne Namen und
  - 0 oder mehr "benamte" Graphen, die jeweils durch eine URI gekennzeichnet werden
- Eine Anfrage muss nicht den Default-Graph ansprechen, sondern kann sich auch rein auf die benamten Graphen beziehen
- Der "aktive" Graph wird mit `GRAPH` gekennzeichnet

SPARQL – FROM (NAMED)

- Das RDF Dataset, das mit der Query abgefragt werden soll, wird mit den Stichworten `FROM` bzw. `FROM NAMED` zusammengesetzt
  - Wenn weder FROM noch FROM NAMED angegeben sind, wird das Standard RDF-Dataset der Implementierung verwendet
  - Werden mehrere FROM Graphen angegeben, entsteht der Default Graph durch einen Merge der mit FROM angegebenen Graphen
  - Werden mehrere FROM NAMED Graphen angegeben, aber kein FROM Graph, ist der Default Graph leer
- Beispiel-Anfrage: (http://example.org/dft.ttl enthält hier Infos über andere Graphen)

```
SELECT ?who ?g ?mbox
       FROM <http://example.org/dft.ttl>
       FROM NAMED <http://example.org/alice>
       FROM NAMED <http://example.org/bob>
       WHERE
       {
          ?g dc:publisher ?who .
          GRAPH ?g { ?x foaf:mbox ?mbox }
       }
```

SPARQL – GRAPH

- Anfrage auf einen Graph beschränken

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX data: <http://example.org/foaf/>

SELECT ?nick
   FROM NAMED <http://example.org/foaf/aliceFoaf>
   FROM NAMED <http://example.org/foaf/bobFoaf>
   WHERE
   {
      GRAPH data:bobFoaf
      {
         ?x foaf:mbox <mailto:bob@work.example> .
         ?x foaf:nick ?nick
      }
   }
```

SPARQL – GRAPH

- Auf Graphnamen zugreifen
  - Folgende Query matcht auf jeden der angegebenen Graphen
  - Auf den Namen des Graphen kann dann mit der src Variable zugegriffen werden

```
PPREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?src ?bobNick
  FROM NAMED <http://example.org/foaf/aliceFoaf>
  FROM NAMED <http://example.org/foaf/bobFoaf>
  WHERE
  {
    GRAPH ?src
    {
      ?x foaf:mbox <mailto:bob@work.example> .
      ?x foaf:nick ?bobNick
    }
  }
```

SPARQL - GRAPH

- Mögliche Graph URIs beschränken
  - Auf eine Variable, die in einem GRAPH-Abschnitt verwendet wurde, kann auch in einem anderen GRAPH-Abschnitt oder in einem Muster für den Default-Graph zugegriffen werden

```
PREFIX  data:  <http://example.org/foaf/>
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX  rdfs:  <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?mbox ?nick ?ppd
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
{
  GRAPH data:aliceFoaf
  {
    ?alice foaf:mbox <mailto:alice@work.example> ;
           foaf:knows ?whom .
    ?whom  foaf:mbox ?mbox ;
           rdfs:seeAlso ?ppd .
    ?ppd  a foaf:PersonalProfileDocument .
  } .
  GRAPH ?ppd
  {
      ?w foaf:mbox ?mbox ;
         foaf:nick ?nick
  }
}
```

SPARQL Result XML Format

SPARQL

- Wie sind Antworten auf SPARQL Queries formatiert?
  - Antwort auf CONSTRUCT und DESCRIBE ist ein RDF Graph
    → Antwort erfolgt z.B.: als RDF/XML
  - Wie werden Antworten auf SELECT codiert?
    - Antwort ist eine Tabelle von Variablenwerten
  - Wie werden Antworten auf ASK Queries codiert?
    - Antwort ist nur ein boolscher Wert
  - Hierfür gibt es das SPARQL Query Result Format
  - MIME-Type: application/sparql-results+xml
  - Empfohlene Dateiendung: .srx

- Format definiert SPARQL Result Document

  - XML Grundgerüst:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
   <head>
     … Hier stehen Header-Daten …
   </head>

   … Hier stehen die Resultate …

</sparql>
```

  - Head kann z.B. Referenz auf Metadaten enthalten:

```
<link href="metadata.rdf"/>
```

- Antwort-Format für SPARQL SELECT Queries
  - Head enthält Elemente für alle angefragten Variablen
  - Die Folge der Resultate (Element `result`) werden im Element `results` zusammengefasst
  - Jedes Resultat enthält ein oder mehrere `binding` Elemente
    - Gebundene Variable wird mit Attribut `name` angegeben
  - Binding-Typen:
    - RDF URI Reference U
      ```
      <binding name="x"><uri>U</uri></binding>
      ```
    - RDF Literal S
      ```
      <binding name="y"><literal>S</literal></binding>
      ```
    - RDF Literal S with language L
      ```
      <binding name="z"><literal xml:lang="L">S</literal></binding>
      ```
    - RDF Typed Literal S with datatype URI D
      ```
      <binding name="a"><literal datatype="D">S</literal></binding>
      ```
    - Blank Node label I
      ```
      <binding name="b"><bnode>I</bnode></binding>
      ```

- Antwort-Format für SPARQL SELECT Queries
  - Beispiel:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="x"/>
    <variable name="hpage"/>
    <variable name="name"/>
    ...
  </head>
  <results>
    <result>
      <binding name="x">
         <bnode>r2</bnode>
      </binding>
      <binding name="hpage">
         <uri>http://work.example.org/bob/</uri>
      </binding>
      <binding name="name">
         <literal xml:lang="en">Bob</literal>
      </binding>
      …
     </result>
     <result> … </result> …
  </results>
</sparql>
```

- Antwort-Format für SPARQL ASK Queries
  - Head kann (Referenz auf) Metadaten enthalten
  - Element `variable` darf im Head nicht vorkommen
  - Resultat wird im Element "`boolean`" zurückgeliefert

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
     ...  head …
  </head>
  <boolean>true</boolean>
</sparql>
```

SPARQL Protokoll

SPARQL

- Protokoll zum Transport von SPARQL Queries zwischen Query Client und Query Prozessor
- Bindings für HTTP und SOAP
- Definiert als einziges Interface `SparqlQuery`, welches wiederum die Methode `query` enthält
- Query ist definiert nach dem In-Out message exchange pattern, welches folgende 2 Nachrichten definiert
    1. Nachricht
        - Ausgezeichnet durch Label “In” mit Richtung “in”
        - Empfangen von einem Knoten N
    2. Nachricht
        - Ausgezeichnet durch Label “Out” mit Richtung “out”
        - Gesendet an Knoten N
- Bei Fehlern wird Out durch eine “Fault” Message ersetzt

- Auszug aus der WSDL 2.0 Definition

```
<!-- Abstract SparqlQuery Interface →
<interface name="SparqlQuery" styleDefault="
http://www.w3.org/2006/01/wsdl/style/iri">

   <!-- the Interface Faults -->
   <fault name="MalformedQuery" element="st:malformed-query"/>
   <fault name="QueryRequestRefused" element="st:query-request-refused"/>

   <!-- the Interface Operation -->
   <operation name="query"
              pattern="http://www.w3.org/2006/01/wsdl/in-out">
      <documentation>
         The operation is used to convey queries and their results from
         clients to services and back again.
      </documentation>

      <input messageLabel="In" element="st:query-request"/>
      <output messageLabel="Out" element="st:query-result"/>

      <!-- the interface faults are out faults -->
      <outfault ref="tns:MalformedQuery" messageLabel="Out"/>
      <outfault ref="tns:QueryRequestRefused" messageLabel="Out"/>
   </operation>
</interface>
```

SPARQL

- Anfrage wird in einem Element vom Typ `st:query-request` verpackt
  - Enthält einen Query String (Element `query`) und
  - 0 oder mehr RDF Dataset Beschreibungen
    - Elemente `default-graph-uri` und `named-graph-uri`
    - Dataset kann auch mit FROM (NAMED) im Query angegeben werden
    - Bei Abweichungen müssen die Angaben im Protokoll verwendet werden
- Antwort erfolgt in einem Element vom Typ `st:query-result`
  - Als SPARQL Result Document bei SELECT und ASK queries
    - Element `vbr:sparql`
  - Als RDF/XML für CONSTRUCT und DESCRIBE
    - Element `rdf:RDF`
  - Fehler als MalformedQuery und QueryRequestRefused

- HTTP Bindings
  - queryHttpGet und queryHttpPost für HTTP GET und POST
  - Faults werden an HTTP Status codes gebunden:
    - MalformedQuery → HTTP 400: Bad request
    - QueryRequestRefused → HTTP 500: Internal Server Error
  - queryHttpGet **sollte** immer verwendet werden außer in Fällen, in denen die URL-codierte Query an praktische Grenzen stößt
  - Query wird im HTTP-Form-Parameter query übergeben
    - Query String muss URL-encodiert sein
    - Gilt gleichfalls für GET und POST
  - Dataset kann mit Anfrageparametern default-graph-uri und named-graph-uri angegeben werden
  - Über HTTP content negotiation kann erwünschtes Rückgabeformat für CONSTRUCT und DESCRIBE angegeben werden
    - z.B.: text/turtle

SPARQL – Protokoll – HTTP Beispiel

Query

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?book ?who WHERE { ?book dc:creator ?who }
```

GET Request URL

```
http://example.org/sparql/?query=PREFIX%20dc%3A%20%3Chttp%3A%2F%2Fpurl.org%2Fdc
%2Felements%2F1.1%2F%3E%20%0ASELECT%20%3Fbook%20%3Fwho%20%0AWHERE%20%7B%20%3Fbook
%20dc%3Acreator%20%3Fwho%20%7D&default-graph-uri=http://www.example.org/graph1
```

HTTP Response

```
HTTP/1.1 200 OK
Date: Fri, 06 May 2005 20:55:12 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4 DAV/1.0.3
Connection: close
Content-Type: application/sparql-results+xml

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head>
   <variable name="book"/>
   <variable name="who"/>
 </head>
 <results distinct="false" ordered="false">
   <result>
     <binding name="book"><uri>http://www.example/book/book5</uri></binding>
     <binding name="who"><bnode>r29392923r2922</bnode></binding>
   </result>
   ...
 </results>
</sparql>
```

- SOAP Bindings
  - Binding querySoap
  - Verwendet HTTP POST zum Transport von Anfragen
  - WSDL Binding Definition:

```
<binding name="querySoap" interface="SparqlQuery"
         type="http://www.w3.org/2006/01/wsdl/soap"
         wsoap:version="1.2"#
         wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
    <fault ref="tns:MalformedQuery" wsoap:code="soap:Sender" />
    <fault ref="tns:QueryRequestRefused" wsoap:code="soap:Sender" />
    <operation ref="tns:query"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-
response" />
</binding>
```

SPARQL – Protokoll – SOAP Beispiel

SOAP Request

```
POST /services/sparql-query HTTP/1.1
Content-Type: application/soap+xml
Accept: application/soap+xml, multipart/related, text/*
User-Agent: Axis/1.2.1
Host: www.example
SOAPAction: ""
Content-Length: 438

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body><query-request xmlns="http://www.w3.org/2005/09/sparql-protocol-types/#">
    <query>SELECT ?z {?x ?y ?z . FILTER regex(?z, 'Harry')}</query>
  </query-request></soapenv:Body></soapenv:Envelope>
```

SOAP Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body><query-result xmlns="http://www.w3.org/2005/09/sparql-protocol-types/#">
    <ns1:sparql xmlns:ns1="http://www.w3.org/2005/sparql-results#">
      <ns1:head><ns1:variable name="z"/></ns1:head>
      <ns1:results distinct="false" ordered="false">
        <ns1:result>
          <ns1:binding name="z">
            <ns1:literal>Harry Potter and the Chamber of Secrets</ns1:literal>
          </ns1:binding>
        </ns1:result>
          …
      </ns1:results></ns1:sparql></query-result></soapenv:Body></soapenv:Envelope>
```

SPARQL Upcoming Features

SPARQL – Upcoming Features

- Version 1.0 von SPARQL deckt nicht alles Erwünschte ab
- SPARQL soll schnell weiterentwickelt werden
- Unter

  http://www.w3.org/TR/sparql-features/

  werden die zukünftigen Features von der SPARQL WG bekannt gegeben (Aktueller Entwurf vom 2. Juli 2009)
- Noch keine endgültige Spezifikation, aber trotzdem oft relevant
- Unterteilung in
  - Required Features – Werden auf die ein oder andere Art mit hoher Wahrscheinlichkeit in der nächsten Version vorhanden sein
  - Time permitting Features – Wenn genügend Zeit vorhanden ist, gehen diese Features in den Standard ein

SPARQL – Upcoming Features

- Required features:
  - Aggregatfunktionen
  - Subqueries
  - Negation
  - Project Expressions
  - SPARQL Update
  - Dienstbeschreibung
- Time-permitting features
  - BGP Erweiterungen for entailment regimes (OWL, RDFS, RIF)
  - Eigenschaftspfade
  - Verbreitete SPARQL Funktionen (e.g. String maninpulation)
  - Einfache verknüpfte Anfragen
  - Verbesserung der Syntax

SPARQL – Upcoming Features

- Aggregatfunktionen
  - Erlauben Operationen wie Zählen, Bestimmung des numerischen Maximums/Minimums/Durchschnitts durch das Zusammenfassen von Spaltenwerten
  - Anwendungsfälle z.B. in der Datenanalyse:
    - Bestimmung der Anzahl von einander unterschiedlicher Ressourcen, die einem bestimmten Kriterium entsprechen
    - Berechnung der Durchschnittsnote von Studenten gruppiert nach dem Regierungsbezirk
    - Summierung der Wahlkampfspenden, gruppiert nach PLZ und Partei
  - Muss im momentanen Standard durch externe Skripte realisiert werden
  - Werden bereits zu unterschiedlichen Teilen von SPARQL Engines unterstützt:
    - Redland → COUNT()
    - Jena ARQ → COUNT(), SUM()
    - Virtuoso → AVG(), COUNT(), SUM(), MIN(), MAX(), +custom
    - ARC → COUNT(), MAX(), MIN(), AVG(), SUM()

SPARQL – Upcoming Features

- Beispiele zu Aggregatfunktionen:

```
SELECT COUNT(?person) AS ?alices
WHERE {
   ?person :name "Alice" .
}
```

```
SELECT AVG(?value) AS ?average
WHERE {
  ?good a       :Widget ;
        :value ?value .
}
```

SPARQL – Upcoming Features

- Subqueries
  - Manchmal ist es notwendig, die Resultate einer Anfrage an eine weitere Anfrage weiterzugeben
  - Momentan muss man dazu
    - Resultate der ersten Anfrage empfangen
    - Mit speziellen Skripten verarbeiten
    - Und sie dann für die zweite Anfrage verwenden
  - Anwendungsfälle:
    - Die 10 letzten Blogeinträge herausfinden, die von einem Autor verfasst worden sind
    - Eine Liste von Leuten mit deren Freunden abfragen, mit jeweils nur einem Namen pro Person
    - Die Anzahl eindeutiger Ergebnisse beschränken, anhand der Anzahl der Ressourcen anstatt anhand der Zahl der Lösungen
  - Die genaue Syntax ist momentan noch nicht bestimmt

SPARQL – Upcoming Features

- Beispiel zu Subqueries (wie in ARQ):

```
SELECT ?person ?name WHERE {
  :Alice foaf:knows ?person .
  {
    SELECT ?name WHERE {
      ?person foaf:name ?name
    } LIMIT 1
  }
}
```

- Vorhandene Implementierungen:
- Virtuoso unterstützt sowohl skalare Subqueries (an allen Stellen, an denen eine Variable auftreten kann) als auch Subqueries als abgeleitete Tabellen
  - ARQ unterstützt eingebettete SELECTS (siehe oben)

SPARQL – Upcoming Features

- Negation:
  - Viele Aufgaben erfordern es zu überprüfen, ob bestimmte Triple in einem Graph vorkommen oder nicht.
  - Die Abwesenheit von Triplen zu überprüfen ist “Negation by failure”
  - Auf Umwegen auch in SPARQL 1.0 möglich:

```
SELECT ?name
WHERE { ?x foaf:givenName  ?name .
        OPTIONAL { ?x foaf:knows ?who } .
        FILTER (!BOUND(?who)) }
```

  - Anwendungsfälle:
    - Alle Leute identifizieren, die niemanden kennen oder eine bestimmte Eigenschaft nicht besitzen
    - Jeden Inhalt identifizieren, der in einem bestimmten Review Prozess noch keinem Lektor zugeordnet worden ist
    - Kunden identifizieren, die ein bestimmtes Objekt nicht erstanden haben
  - Realisierung entweder mit neuem Schlüsselwort oder mit Filterfunktion

SPARQL – Upcoming Features

- Beispiele: (1. mit SeRQL MINUS und 2. mit UNSAID)

```
SELECT x FROM {x} foaf:givenName {name}
MINUS SELECT x FROM {x} foaf:givenName {name} ;
                        foaf:knows {who}
```

```
SELECT ?x
  WHERE { ?x foaf:givenName ?name
          UNSAID { ?x foaf:knows ?who } }
```

- Existierende Implementierungen:
  - RDF::Query → UNSAID Syntax
  - SeRQL → MINUS Operator
  - Jena ARQ → NOT EXISTS (mit UNSAID als Alias)

- Vermutlich wird es UNSAID
  - Bereits für Version 1.0 vorgeschlagen, aber nicht realisiert worden

SPARQL – Upcoming Features

- Project Expressions
  - Möglichkeit, die Resultate von Expressions an Variablen zu binden anstatt von lediglich RDF Termen
  - Eine PE kann sein: Eine Variable, eine konstante URI, ein konstantes Literal, ein beliebiger Ausdruck (inkl. Funktionsaufrufe) über Variablen oder Konstanten
  - Bei Funktionen sind auch benutzerdefinierte denkbar
  - Anwendungsfälle
    - Die Gesamtkosten einer Position in einer Bestellung als Produkt von zwei Variablen: `?unit_cost * ?quantity`
    - Verwendung von SPARQL Accessoren, um die in einem Dataset verwendeten Sprachen herauszufinden: `LANG(?o)`
    - Rückliefern berechneter Werte, wie etwa der aktuelle Wochentag: `ex:dayOfTheWeek(ex:Today())`
    - Stringverarbeitung: `ex:substring(?url, 8, ex:length(?url))`
  - In Verbindung mit Subqueries ein mächtiges Instrument für alle möglichen Anwendungsfälle

SPARQL – Upcoming Features

- Beispiele:

```
SELECT ?name (?age > 18) AS over18
 WHERE {   ?person :name ?name ;
                   :age  ?age . }

SELECT fn:string-join(?givenName, ' ', ?surname) AS ?
fullName
 WHERE {  ?person foaf:givenname ?givenName ;
                  foaf:surname   ?surname ;
                  foaf:interest  :trees . }

CONSTRUCT { ?x foaf:name ?fullName }
 WHERE {
   { SELECT fn:string-join(?gn, " ", ?sn) AS ?fullName
     WHERE { foaf:givenname ?gn ; foaf:surname ?sn . }
} }
```

- Existierende Implementierungen
  - Redland, ARQ (mit etwas anderer Syntax), Virtuoso

SPARQL – Upcoming Features

- SPARQL/Update
  - Spracherweiterung, um Veränderungen an einem Graphen auszudrücken
  - Lehnt sich an SPARQL Syntax an
    - Wer SPARQL bereits kennt, soll sich SPARQL/UPDATE schnell aneignen können
  - Folgende Operationen sollen definiert werden:
    - Einfügen neuer Triple in einen Graph
    - Löschen vorhandener Triple
    - Ausführen mehrerer UPDATE-Operationen mit einer Anfrage
    - Im Store einen neuen Graph erstellen
    - Einen im Store vorhandenen Graph löschen
  - Wurde als Member Submission ins W3C eingebracht
    - Stammt ursprünglich vom Jena/ARQ Projekt
  - Jedoch einige Bedenken vorhanden:
    - Sicherheitsprobleme, Transaktionssicherheit, Wunsch auch nach Operationen für das Modifizieren von Daten ...

SPARQL – Upcoming Features

- Beispiele:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA
{ <http://example/book3> dc:title     "A new book" ;
                         dc:creator   "A.N.Other" .}
```

```
DELETE { ?book ?p ?v }
WHERE
  {   ?book dc:date ?date .
       FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
      ?book ?p       ?v       }   }
```

- Existierende Implementierungen:
  - Jena/ARQ
  - Virtuoso

SPARQL – Upcoming Features

- Time-permitting features:
  - Basic Graph Pattern Erweiterungen für Vererbungsordnungen
    - Durch erweiterte Semantik in z.B. RDFS und OWL werden beim Graphvergleich weitere Lösungen als nur die direkten möglich
  - Eigenschaftspfade
    - Viele Anfragen an RDF-Graphen erfordern das Traversieren hierarchischer Datenstrukturen mit undefinierter Tiefe
      - Alle Elemente einer RDF-Collection zurückgeben
      - Alle Namen meiner Vorfahren (alle ex:mother und ex:father)
      - Alle direkte und inderekte Oberklassen einer rdfs:Class
  - Häufig verwendete SPARQL Funktionen
    - Viele Implementierungen bieten neben den von SPARQL vorgeschriebenen noch weitere Funktionen. Ein Teil davon soll in den Standard übernommen werden
  - Basic Federated Queries
    - Gleichzeitiges Abfragen mehrerer SPARQL Endpoints
  - Dienstbeschreibung
    - SPARQL Endpunkte sollen die von ihnen unterstützten Fähigkeiten an Anfragende melden können
  - Überarbeitung der SPARQL Syntax
    - Vereinfachungen wie Kommas zwischen Variablen und Ausdrücken in SELECTS

SPARQL – Upcoming Features

Noch Fragen ?

SPARQL

- Literatur:
  - Buch “Semantic Web Grundlagen”, Springer Verlag 2008
    Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, York Sure
    ISBN: 978-3-540-33993-9
  - W3C: SPARQL Query Language for RDF
    http://www.w3.org/TR/rdf-sparql-query/
  - W3C: SPARQL Protocol for RDF
    http://www.w3.org/TR/rdf-sparql-protocol/
  - W3C: SPARQL Query Results XML Format
    http://www.w3.org/TR/rdf-sparql-XMLres/
  - W3C: SPARQL New Features and Rational
    http://www.w3.org/TR/sparql-features
  -