

Regel- und Anfragesprachen

Julius Dannert, Florian Thiemer, Alexander Waldmann

Freie Universität Berlin Fachbereich Informatik

14.5.2007

Inhalt

Regelsprachen

- Regeln

- RuleML

- SWRL

- Prolog

Anfragesprachen

- Anfragesprachen

- SPARQL

- RDQL

- SeRQL

Beispiele für Regeln

- ▶ Wenn Person P einen Bruder B und ein Kind K hat, dann ist B der Onkel von K.
- ▶ Der Discount für einen Kunden beträgt 5.0 Prozent, wenn der Kunde ein Premiumkunde und das Produkt luxuriös ist.
- ▶ Eine Euro entspricht 1,95583 DM.
- ▶ Autofahrer müssen mindestens 18 Jahre alt sein.
- ▶ Menschen sind fehlbar.
- ▶ Wenn es brennt, schnell aus dem Haus laufen.

Hierarchie der Regeln

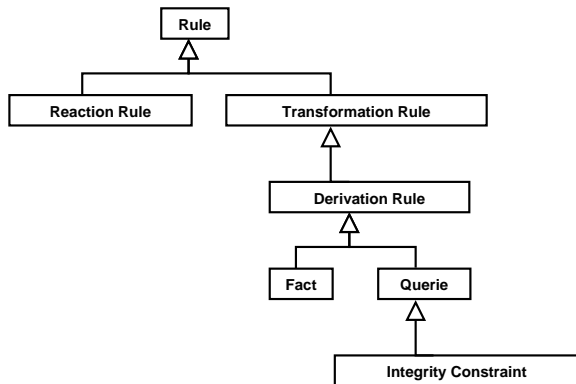


Figure: So werden Regeln eingeteilt.

Reaction rules / Reaktionsregeln

- ▶ Event-Trigger
- ▶ Auf Grund eines Ereignisses wird eine Aktion ausgeführt.
- ▶ Bsp.: SQL-Trigger, E-Mail-Filterregeln

```
CREATE RULE tipp_rechtzeitig AS ON INSERT TO tippt  
WHERE CURRENT_TIMESTAMP > (SELECT s.begin FROM spiel s  
WHERE NEW.spielId = s.spielId)  
DO INSTEAD NOTHING;
```

Transformation rules / Transformationsregeln

- ▶ Gleichheits- bzw. Äquivalenz-Regeln
- ▶ Umwandlung eines Formates in ein anderes
- ▶ Bsp.: XSLT (Zur Umwandlung von XML-Dokumenten)

```
<xsl:template match="/">
  <h1>
    <xsl:value-of select="//title"/>
  </h1>
  <p>
    <xsl:value-of select="//para"/>
  </p>
</xsl:template>
```

Derivation rules / Ableitungsregeln

- ▶ Implikations- bzw. Interferenz-Regeln
- ▶ Gewinnung neuer Informationen
- ▶ $a_1 \wedge \dots \wedge a_n \Rightarrow d$
- ▶ Bsp.: Prolog, SQL-View

```
sibling(X,Y):- father(P,X),father(P,Y).  
sibling(X,Y):- mother(P,X),mother(P,Y).
```

Facts / Fakten

- ▶ Voraussetzungslose Ableitungsregeln
- ▶ $true \Rightarrow fact$
- ▶ Bsp.: Prolog

```
father(darth_vader,luke_skywalker).
```

Queries / Abfragen

- ▶ 'Schlusslose' Ableitungsregeln
- ▶ *query* \Rightarrow *false*
- ▶ In RuleML seit Version 0.8
- ▶ Bsp.: SQL

```
SELECT sum(punkte) AS punkte  
FROM tippt t  
WHERE name='julius'
```

Integrity constraints / Integritätsregeln

- ▶ Konsistenz-Regeln
- ▶ Bsp.: OCL, SQL

```
CREATE TABLE benutzer(  
  
name varchar(35) PRIMARY KEY,  
geschlecht varchar(8) NOT NULL,  
  
CONSTRAINT geschlecht_ok  
CHECK (geschlecht IN ('M','W'))  
);
```

Regeln - Aber wofür?

Klassisch

- ▶ Künstliche Inteligenz
- ▶ Agenten
- ▶ E-Commerce
- ▶ Dokumenten Generation aus XML
- ▶ Policies

Modern

- ▶ 'Regellose' Software ist wie datenbanklose Software
- ▶ Regeln heutzutage meist noch hart kodiert

Das ist RuleML



- ▶ XML-Basierte Markup Language
- ▶ Austausch und Speicherung von Regeln
- ▶ semiformal
- ▶ **kein** w3c-Standard, aber de facto Standard
 - ▶ bis jetzt gibt es keine w3c-Recommendation für Regelformate
 - ▶ seit Nov. 2005 Rule Interchange Format (RIF) Working Group
- ▶ www.ruleml.org
- ▶ Version 0.91

Hierarchie der Regeln

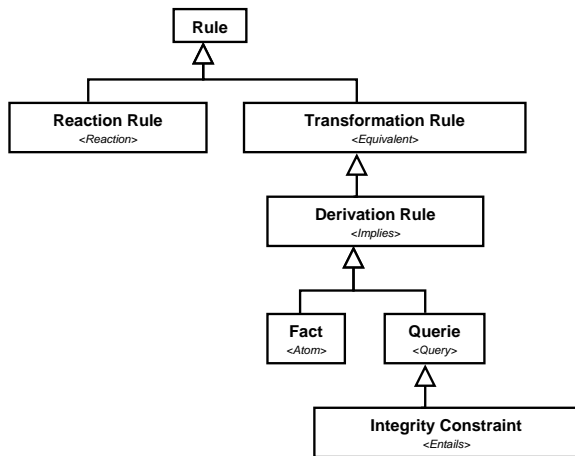


Figure: So werden in RuleML die Regeln getaggt.

Facts / Fakten

```
<Atom>  
  <Rel>luxury</Rel>  
  <Ind>Porsche</Ind>  
</Atom>
```

Derivation rules / Ableitungsregeln

```
<Implies>
  <head>
    <Atom>
      <Rel>discount</Rel>
      <Var>customer</Var>
      <Var>product</Var>
      <Ind>5.0 percent</Ind>
    </Atom>
  </head>
  <body>
    <And>
      <Atom>
        <Rel>premium</Rel>
        <Var>customer</Var>
      </Atom>
      <Atom>
        <Rel>luxury</Rel>
        <Var>product</Var>
      </Atom>
    </And>
  </body>
</Implies>
```

Transformation rules / Transformationsregeln

```
<Equivalent>
  <torso>
    <Atom>
      <Rel>own</Rel>
      <Var>person</Var>
      <Var>object</Var>
    </Atom>
  </torso>
  <torso>
    <Atom>
      <Rel>belongs</Rel>
      <Var>object</Var>
      <Var>person</Var>
    </Atom>
  </torso>
</Equivalent>
```

Queries / Abfragen

```
<Query>
  <Atom>
    <oid>
      <Ind>
        Give the discount amounts for all customers
        buying any products.
      </Ind>
    </oid>
    <Rel>discount</Rel>
    <Var>customer</Var>
    <Var>product</Var>
    <Var>amount</Var>
  </Atom>
</Query>
```

Integrity constraints / Integritätsregeln

```
<Entails>
  <Rulebase>
  ...
  </Rulebase>

  <Rulebase>
    <Forall>
      <Var>x</Var>
      <Implies>
        <Atom>
          <Rel>emp</Rel>
          <Var>x</Var>
        </Atom>
        <Exists>
          <Var>y</Var>
          <Atom>
            <Rel>ssn</Rel>
            <Var>x</Var>
            <Var>y</Var>
          </Atom>
        </Exists>
      </Implies>
    </Forall>
  </Rulebase>

</Entails>
```

Reaction rules / Reaktionsregeln

Reaction RuleML 0.1

```
<Reaction>
  <event>
    <Atom>
      <Rel>occurs</Rel>
      <Expr>
        <Fun>heartbeat</Fun><Var>Service</Var>
      </Expr>
      <Var>T</Var>
    </Atom>
  </event>
  <action>
    <Assert>
      <Atom>
        <Rel>alive</Rel>
        <Var>Service</Var>
        <Var>T</Var>
      </Atom>
    </Assert>
  </action>
</Reaction>
```

RuleML - Viele Sprachen

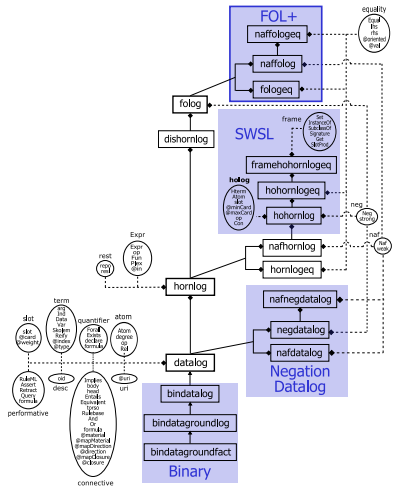


Figure: Modularisierung von RuleML

RuleML - Viele Sprachen

RuleML ist eine Familie von Untersprachen, dessen Wurzel den Zugriff auf die Sprache als ganzes erlaubt.

Datalog

- ▶ Schnittmenge von SQL und Prolog
- ▶ Für Informationen über relationelle Datenbanken
- ▶ fact = explizite Zeile einer Tabelle
- ▶ rule = Tabelle / View
- ▶ Kernsprache

RuleML - Viele Sprachen

Hornlog

- ▶ Hornklausel : $x_1 \wedge \dots \wedge x_n \rightarrow y$
- ▶ Prolog

Folog

- ▶ First order Logic = Prädikatenlogik der ersten Stufe
- ▶ Quantoren (\forall , \exists)
- ▶ Quantoren binden die Leerstellen von Prädikaten

So entwickelt sich RuleML

- ▶ Definition: DTD → XML-Schema (seit Version 0.85)
- ▶ XML-Syntax → RDF-Syntax
- ▶ Konkrete Syntax → abstrakte Syntax / MOF-Model

▶ Paper MOF-Model

0.7	0.8	0.87	0.88	0.91
<if>	<imp>	<Imp>	<Implies>	<Implies>
-	<_body>	<body>	<body>	<body> o. -

- ▶ JSR-94 Java Rule Engine API 1.0a Final Release (04.08.2004)

Andere Regelsprachen

- ▶ Jess (Rule Engine for Java)
 - ▶ Jess rule language (bevorzugt)
 - ▶ JessML (XML)
- ▶ Business Rules Markup Language (BRML) von IBM
- ▶ Rule Interchange Format (RIF) vom W3C
- ▶ DARPA Agent Mark Up Language (DAML)
- ▶ Knowledge Interchange Format (KIF)
- ▶ RuleSpeak
- ▶ Semantics of Business Vocabulary and Business Rules (SBVR) von OMG
- ▶ Extensible Rule Markup Language (XRML)
- ▶ Agent-Oriented Rule Markup Language (AORML)
- ▶ Extensible Rule Markup Language (XRML)
- ▶ Artificial Intelligence Markup Language (AIML)

Andere Regelsprachen: Jess

```
[findMatchesForRebeccaSmith:  
  (?p :location ?l)  
  (?l :attraction ?a)  
  (?a rdf:type :ShoppingMall)  
  
  ->( :RebeccaSmith :suggestedProperty ?p)]
```

- ▶ Rebecca sucht nach einer Wohnung nahe einer Einkaufshalle

▶ Weiterer Beispiele

SWRL

- ▶ Die **S**emantic **W**eb **R**ule **L**anguage basiert auf OWL DL/Lite und Datalog RuleML.
- ▶ W3C Member Submission
- ▶ OWL wird um Horn-Regeln erweitert.
- ▶ Man kann Aussagen treffen, die mit OWL nicht machbar sind.

Beispiel

Ein Onkel ist der Bruder eines Vaters

Zur Erinnerung...

OWL Dokumente bestehen aus

- ▶ Class axioms
- ▶ Property axioms
- ▶ Fakten über Individuen

SWRL = OWL + RuleML

SWRL Dokumente bestehen aus

- ▶ Class axioms
- ▶ Property axioms
- ▶ Fakten über Individuen
- ▶ RuleML:Implies
- ▶ RuleML:Var

Aufbau der Regeln

- ▶ Ein Axiom ist eine Regel die aus einem Head und einem Body besteht
- ▶ Head und Body bestehen aus keinem oder beliebig vielen Atomen
- ▶ Wenn die Voraussetzung (Body) gilt, muss auch die Nachbedingung (Head) gelten
- ▶ Ein leerer Body wird als immer wahr angenommen und somit als Fakt interpretiert

Head und Body

Head und Body sind Konjunktionen der Form:

- ▶ $C(x)$ oder $P(x, y)$
- ▶ *sameAs*(x, y), *differentFrom*(x, y)

Wobei C eine OWL Description oder data range ist, P eine OWL Property und x, y sind entweder Variablen, OWL Individuen oder OWL Data Values.

OWL Description

In OWL gibt es 6 verschiedene Arten von Class descriptions

- ▶ Einen Klassenidentifizierer (URI reference)
- ▶ Aufzählung von Individuen
- ▶ Property restriction
- ▶ Schnitt von 2 oder mehreren Class descriptions
- ▶ Vereinigung von 2 oder mehreren Class descriptions
- ▶ Komplement von einer Class descriptions

OWL Property

In OWL gibt es 2 Arten von Properties

- ▶ Object Property
 - ▶ Verlinkt ein Individuum zu einem Individuum
 - ▶ equivalentProperty
 - ▶ inverseOf
- ▶ Datatype Property
 - ▶ verlinkt ein Individuum zu einem Datenwert
 - ▶ TransitivProperty
 - ▶ Symmetric Property

OWL Individuals

- ▶ `sameAs`
 - ▶ Zwei Individuen werden als gleich gekennzeichnet
- ▶ `differentFrom`
 - ▶ Zwei Individuen werden als verschieden gekennzeichnet

Beispiel:

```
<owl:Class rdf:ID="FootballTeam">  
  <owl:sameAs rdf:resource="http://sports.org/US#SoccerTeam"/>  
</owl:Class>
```

Funktionsweise von SWRL

- ▶ Man fügt Bindungen von Variablen als Erweiterung der OWL Interpretation hinzu.
- ▶ Eine Regel gilt dann als erfüllt, wenn jede Bindung die die Vorbedingung erfüllt, auch die Nachbedingung erfüllt.
- ▶ Die semantischen Bedeutungen von Axiomen und Ontologien werden nicht verändert.

SWRL - Syntax

Die XML Syntax von SWRL ist eine Kombination aus der OWL und RuleML Syntax

Vorteile

- ▶ Beliebige OWL Klassen können als Prädikate in Regeln verwendet werden
- ▶ Regeln und Ontologie Axiome können beliebig gemischt werden
- ▶ Mapping zu einem RDF Graphen ist möglich
- ▶ Interoperabilität zwischen RuleML und OWL wird erleichtert

SWRL - Konkrete Syntax

```
<swrlx:classAtom>  
  <owlx:Class owlx:name="Person" />  
  <ruleml:var>x1</ruleml:var>  
</swrlx:classAtom>
```

```
<swrlx:classAtom>  
  <owlx:IntersectionOf>  
    <owlx:Class owlx:name="Person" />  
    <owlx:ObjectRestriction owlx:property="hasParent">  
      <owlx:someValuesFrom owlx:class="Physician" />  
    </owlx:ObjectRestriction>  
  </owlx:IntersectionOf>  
  <ruleml:var>x2</ruleml:var>  
</swrlx:classAtom>
```

SWRL - Konkrete Syntax

```
<swrlx:individualPropertyAtom swrlx:property="hasParent">  
  <ruleml:var>x1</ruleml:var>  
  <owlx:Individual owlx:name="John" />  
</swrlx:individualPropertyAtom>
```

```
<swrlx:datavaluedPropertyAtom swrlx:property="grade">  
  <ruleml:var>x1</ruleml:var>  
  <owlx:DataValue owlx:datatype="xsd:int">4</owlx:DataValue>  
</swrlx:datavaluedPropertyAtom>
```

SWRL - Konkrete Syntax

```
<ruleml:imp>
  <ruleml:_rlabel ruleml:href="#example1"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom swrlx:property="hasParent">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="hasBrother">
      <ruleml:var>x2</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom swrlx:property="hasUncle">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

Was ist Prolog

- ▶ Prolog steht für Programming in logic
- ▶ Prolog gehört zu den logischen Programmiersprachen
- ▶ Erste Entwicklungen seit den 70ern
- ▶ ISO Standard seit 1995
- ▶ Sehr viele Implementierungen von Prolog (SWIProlog, GNUProlog, ...)

Prolog

- ▶ Ein Prologprogramm besteht aus
 - ▶ Fakten
 - ▶ Regeln
 - ▶ Anfragen
- ▶ Prolog probiert Anfragen durch die gegebenen Fakten und Regeln zu erfüllen
- ▶ Closed-World-Assumption

Prolog - Fakten

- ▶ Fakten bestehen aus einem Funktor und einem oder mehreren Literalen
- ▶ Sie werden mit einem Punkt abgeschlossen
- ▶ Fakten bilden die Wissensbasis der geschlossenen Welt in Prolog

```
male(tom).
```

```
parent(sam,bob).
```

```
brother(sam,tom).
```

Prolog - Regeln

- ▶ Regeln werden als Implikationen aufgeschrieben
- ▶ Links steht der Head, der aus einem Funktor und beliebig vielen Literalen oder Variablen besteht
- ▶ Die Klauseln werden mit einem , für eine UND-Verknüpfung und mit ; für eine ODER-Verknüpfung zusammengesetzt.

`uncle(X,Y) :- parent(Z,Y), brother(Z,X) .`

Zu lesen:

- ▶ Wenn Z ein Elternteil von Y ist und X der Bruder von Z, dann ist X der Onkel von Y

Prolog - Anfragen

```
?- uncle(X,bob).
```

Prolog sucht in seiner Datenbank und ersetzt bei matching

```
parent(Z,bob),brother(Z,X).
```

```
parent(sam,bob),brother(sam,X)
```

```
parent(sam,bob),brother(sam,tom)
```

⇒

```
X=tom
```

Prolog und RDF/OWL

- ▶ SWI-Prolog Open-Source Implementation des ISO Standards
- ▶ Lizenz unter der LGPL \Rightarrow SWI-Prolog kann in Anwendungen benutzt werden, die nicht unter der GPL stehen
- ▶ SWI-Prolog stellt libraries bereit um RDF und OWL Dokumente parsen, ändern und erstellen zu können

Im folgenden betrachten wir die rdf/semweb library von SWI-Prolog

Prolog und RDF

- ▶ Zentrales Modul ist die `rdf_db.pl` Datei
 - ▶ kann RDF-Tripel speichern
 - ▶ bietet grundlegende Query Funktionen
- ▶ `rdfs.pl` und `owl.pl` bieten Query-Unterstützung für RDFS und OWL
- ▶ `rdf_edit.pl` erlaubt Änderungen an RDF-Dokumenten
- ▶ `rdf_write.pl` und `rdf_write_xml.pl` ermöglicht Speicherung als RDF Graph

Das RDF Datenmodell

- ▶ Das RDF Modell entspricht einem gerichteten Graphen
- ▶ Subjekt, Prädikat und Objekt bilden ein sogenanntes RDF-Tripel
- ▶ Man nennt dies auch ein Statement innerhalb einer Domäne
- ▶ Subjekt bezeichnet eine Ressource
- ▶ Prädikat sagt etwas über das Subjekt aus
- ▶ Objekt beschreibt den Wert des Prädikat

RDF - Beispiel

- ▶ Der Autor von <http://www.beispielseite.de> ist Max Mustermann
- ▶ Der Titel von <http://www.beispielseite.de> ist Hallo Welt
- ▶ Subjekt in beiden Fällen <http://www.beispielseite.de>
- ▶ Prädikate sind hatAutor und hatTitel
- ▶ Objekte sind Max Mustermann und Hallo Welt

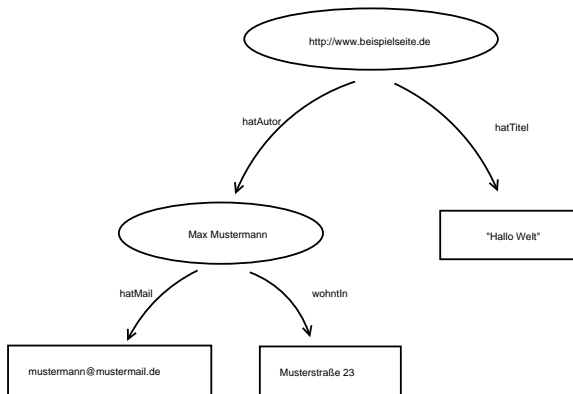
RDF - Beispiel

- ▶ Ein Objekt kann auch wieder eine Ressource bezeichnen
- ▶ Somit kann ein Objekt eines Statements ein Subjekt eines anderen Statements sein

Beispiel

- ▶ Die E-Mail von Max Mustermann ist `mustermann@mustermail.de`
- ▶ Max Mustermann wohn in der Musterstraße 23

RDF - Beispiel



Das ganze in XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.beispielseite.de">
    <hatAutor rdf:ressource="http://www.beispielseite.de/MaxMustermann" />
    <hatTitel>Hallo Welt</hatTitel>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.beispielseite.de/MaxMustermann">
    <hatMail>mustermann@mustermail.de</hatMail>
    <wohntIn>Musterstrasse 23</wohntIn>
  </rdf:Description>
</rdf:RDF>
```

Zurück zu Prolog

```
24 ?- load_rdf('C:/.../Beispiel1.rdf',List).
```

```
List = [rdf('__Description2', 'http://www.w3.org/1999/02/22-rdf-syntax-ns#resource',  
literal('http://www.beispielseite.de/MaxMustermann')), rdf('http://www.beispielseite.de',  
hatAutor, '__Description2'), rdf('http://www.beispielseite.de', hatTitel,  
literal('Hallo Welt')), rdf('http://www.beispielseite.de/MaxMustermann', hatMail,  
literal('mustermann@mustermail.de')), rdf('http://www.beispielseite.de/MaxMustermann',  
wohntIn, literal('Musterstrasse 23'))
```

- ▶ Mit *checklist(assert, List)* lassen sich die RDF-Tripel als Fakt abspeichern

Prolog und RDF

```
23 ?- rdf(X,Y,Z).
```

```
X = '__Description1',  
Y = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#resource',  
Z = literal('http://www.beispielseite.de/MaxMustermann') ;
```

```
X = 'http://www.beispielseite.de',  
Y = hatAutor,  
Z = '__Description1' ;
```

```
X = 'http://www.beispielseite.de',  
Y = hatTitel,  
Z = literal('Hallo Welt') ;
```

```
X = 'http://www.beispielseite.de/MaxMustermann',  
Y = hatMail,  
Z = literal('mustermann@mustermail.de') ;
```

```
X = 'http://www.beispielseite.de/MaxMustermann',  
Y = wohntIn,  
Z = literal('Musterstrasse 23') ;
```

No

▶ Weiterer Infos

RDF Query Language Designs

Unterscheidung

- ▶ SQL-like: RDQL/Squish, SeRQL, RDFDB QL, RQL, ...
- ▶ XPath-like: Versa, RDFPath
- ▶ Rules-like: N3QL, Triple, DQL, OWL-QL, ...
- ▶ Using XML: XSLT, XPath, XQuery
- ▶ SQL-ähnliche Sprachen mit Abstand die verbreitetsten

Wozu Anfragesprachen

- ▶ Kennt RDF-Graphen und Tripel
- ▶ Kann auf RDF-Graphen operieren
- ▶ Erlaubt Softwareentwicklung auf einem höheren Niveau als Tripel
- ▶ Erlaubt "cross-language, cross-platform" Entwicklung

SPARQL

- ▶ RDF data access query language
- ▶ Anfragesprache und Daten-Zugriffs-Protokoll für das Semantic Web
- ▶ "SPARQL Protocol and RDF Query Language"
- ▶ Spezifikation entwickelt unter der RDF Data Access Working Group (DAWG)
- ▶ <http://www.w3.org/2001/sw/DataAccess/>
- ▶ W3C Working Draft

SPARQL

- ▶ Reine Anfragesprache (SELECT)
- ▶ Keine Datenmanipulation (UPDATE,DELETE)
- ▶ SQL-Ähnliche Syntax
- ▶ Viele unabhängige Implementierungen / Anwendungen

SPARQL: Simple Anfrage

```
PREFIX abc: <http://example.com/exampleOntologie#>
SELECT ?capital ?country
WHERE {
    ?x abc:cityname ?capital.
    ?y abc:countryname ?country.
    ?x abc:isCapitalOf ?y.
    ?y abc:isInContinent abc:africa.
}
```

- ▶ Variablen mit führendem ?
- ▶ Triple Muster stellen Anfrage-Bedingungen dar
- ▶ XML-Namespaces ähnliches PREFIX

SPARQL: Komplexe Anfrage

```
PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?name ?symbol ?number
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE
{
  ?element table:name ?name.
  ?element table:symbol ?symbol.
  ?element table:atomicNumber ?number.
}
```

- ▶ Graphen Muster: Mehrere Triple Muster die die Anfragebedingungen darstellen
- ▶ Variablen haben immer gleichen Wert
- ▶ WHERE ist optional

Verkürzte Anfrage

```
PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT *
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE
{
  ?element table:name      ?name;
           table:symbol    ?symbol;
           table:atomicNumber ?number.
}
```

- ▶ * gibt alle Variablen aus Graphen Muster aus
- ▶ bequemer, aber der Prozessor sortiert Spalten
- ▶ TURTLE RDF Syntax
- ▶ Bedingungen können OPTIONAL sein

Datentypen

Datentypen für Literale

```
@prefix dt: <http://example.org/datatype#> .  
@prefix ns: <http://example.org/datatype#> .  
@prefix : <http://example.org/datatype#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
  
:x ns:p "42"^^xsd:integer .  
:y ns:p "abc"^^dt:specialDatatype .  
:z ns:p "cat"@en .
```

XSD Datatypes: boolean, integer, decimal, dateTime

Union

```
PREFIX table: <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ....
FROM .....
WHERE
{
  {
    <Tripel Muster>
  }
  UNION
  {
    <Tripel Muster>
  }
}
```

Union

- ▶ Mehrere Graphen Muster vereint
- ▶ Vereinigung der Lösungsmengen beider Anfragebedingungen
- ▶ Sortierung möglich via ORDER BY
- ▶ Lösungsmengenbegrenzung via LIMIT xx
- ▶ 'Paging' via OFFSET xx

Anfrageergebnis

```
BASE <http://www.daml.org/2003/01/periodictable/>
PREFIX table: <PeriodicTable#>

SELECT ?name
FROM <PeriodicTable.owl>
{ ?element table:name ?name. }
```

row	name
1	sodium
2	neon
3	iron

- ▶ Lösungen der Anfrage in Tabellenform
- ▶ Eine Zeile entspricht einer Lösung
- ▶ Spalten sind Variablen in Graphen Muster

Ergebnisstruktur

Ergebnisstruktur

- ▶ Passt zu bestehender SQL-Unterstützung
- ▶ Protokolle: ODBC, JDBC
- ▶ SQL Database access APIs: Perl DBI
- ▶ SPARQL wurde in Oracle integriert

SPARQL Query Results XML Format

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="name"/>
  </head>
  <results ordered="false" distinct="false">
    <result>
      <binding name="name">
        <literal datatype="...XMLSchema#string">sodium</literal>
      </binding>
    </result>
    <!-- more results -->
  </results>
</sparql>
```

SPARQL Query Results XML Format

- ▶ SPARQL Query Results XML Format
- ▶ <http://www.w3.org/TR/rdf-sparql-XMLres/>

Tools und APIs

- ▶ ARQ, a SPARQL processor for Jena
- ▶ <http://jena.sourceforge.net/ARQ/>
- ▶ Rasqal, the RDF query library included in Dave Beckett's comprehensive Redland framework
- ▶ <http://librdf.org/rasqal/>
- ▶ RDF::Query
- ▶ <http://kasei.us/code/rdf-query/>

RDQL

- ▶ RDQL - eine Anfragesprache für RDF
- ▶ Ähnlich SPARQL, mit Syntaxunterschieden
- ▶ W3C submission

RDQL

```
SELECT { Liste von Variablen }  
FROM { Liste der RDF-Quellen }  
WHERE { Vergleichsmuster zum Graphen }  
AND { Filter }  
USING { Namespace-Mappings }
```

RDQL

```
SELECT ?family , ?given
WHERE  (?vcard vcard:FN "John Smith")
       (?vcard vcard:N ?name)
       (?name vcard:Family ?family)
       (?name vcard:Given ?given)
USING vcard FOR <http://www.w3.org/2001/vcard-rdf/3.0#>
```

- ▶ Triple Muster geklammert
- ▶ SPARQL-PREFIX entspricht hier USING

RDQL

```
SELECT ?resource
WHERE (?resource info:age ?age)
AND ?age >= 24
USING info FOR <http://example.org/peopleInfo#>
```

- ▶ Anfragebedingungen mit Literalen werden mit AND angehängt
- ▶ Anfragen ohne Quellenangabe bekommen RDF Graph von ausführender Umgebung zugewiesen

RDQL

- ▶ RDQL-Anfragen in Java: Jena
- ▶ <http://jena.sourceforge.net/index.html>
- ▶ unterstützt RDF, RDFS und OWL, SPARQL

Sesame

- ▶ Entwickelt von Aduna (<http://www.openrdf.org/>)
- ▶ Sesame: Open Source RDF Framework
- ▶ Speichern, Querying und Reasoning von RDF/RDFS
- ▶ In Entwicklungsphase, wenig Werkzeuge verfügbar

SeRQL

- ▶ Sesame RDF Query Language, basiert auf RQL, RDQL
- ▶ Ausgesprochen 'circle'
- ▶ Vor allem nützlich dank Path expressions
- ▶ Andere RDF Anfragesprachen haben Pfad Ausdrücke mit maximaler Länge 1
- ▶ und finden Tripel eines RDF Graphen
- ▶ SeRQL findet Pfad Ausdrücke beliebiger Länge

SeRQL Anfragen

```
SELECT DISTINCT *  
FROM {Country1} ex:borders {} ex:borders {Country2}  
USING NAMESPACE  
    ex = <http://example.org/things#>
```

- ▶ Anfragen bestehen aus SELECT, FROM, WHERE, LIMIT, OFFSET und USING NAMESPACE Klauseln
- ▶ bei CONSTRUCT-Anfragen: CONSTRUCT statt SELECT, sonst identisch
- ▶ SQL-ähnliche Syntax wie SPARQL

SeRQL Anfragen

```
CONSTRUCT {Parent} ex:hasChild {Child}  
FROM {Child} ex:hasParent {Parent}  
USING NAMESPACE  
    ex = <http://example.org/things#>
```

- ▶ CONSTRUCT Anfrage liefert RDF Graphen als Menge von Tripeln
- ▶ Diese Anfrage definiert das inverse zu 'foo:hasParent' als 'foo:hasChild'
- ▶ Erzeugt Informationen die aus bestehenden Informationen abgeleitet wurde

SeRQL Anfragen

```
SELECT Country
FROM {Country} ex:population {Population}
WHERE Population < "1000000"^^xsd:positiveInteger
USING NAMESPACE
    ex = <http://example.org/things#>
```

- ▶ SELECT Anfrage - Vergleich SPARQL. Operatoren für die WHERE-Klausel: =, <, >, !=
- ▶ Funktionen isLiteral(), isURI(), uvm.
- ▶ Verknüpft mit AND, OR, NOT
- ▶ XML Schema Datentyp Unterstützung für Literale

SeRQL



```
{Person} ex:worksFor {Company} rdf:type {ex:ITCompany}
```

```
{Person} ex:worksFor {Company}.  
{Company} rdf:type {ex:ITCompany}
```

- ▶ Geschweifte Klammern umgeben Knoten im Graphen
- ▶ Dazwischen die Kanten
- ▶ Pfad-Ausrichtung von Links nach Rechts

```
{Person} ex:worksFor {} rdf:type {ex:ITCompany}  
{Painting} ex:painted_by {} ex:name {"Picasso"}  
{comic:RoadRunner} SomeRelation {foo:WillyECoyote}
```

- ▶ Knoten können leer bleiben wenn Wert uninteressant
- ▶ Knoten und Kanten können Variablen, Literale und URIs sein

SeRQL

```
SELECT title  
FROM {book} dc10:title {title}
```

UNION

```
SELECT title  
FROM {book} dc11:title {title}
```

USING NAMESPACE

```
dc10 = <http://purl.org/dc/elements/1.0/>,  
dc11 = <http://purl.org/dc/elements/1.1/>
```

- ▶ UNION anders aufgebaut als bei SPARQL
- ▶ INTERSECT
- ▶ MINUS

SeRQL

Weitere tolle Features

- ▶ Geschachtelte Anfragen
- ▶ IN Operator
- ▶ ANY, ALL, EXISTS

SeRQL

- ▶ Verschiedene Short-Cuts möglich wie in SPARQL:
- ▶ Multi-Value-nodes
- ▶ Optionale Pfad-Ausdrücke geklammert mit []
- ▶ (vergleich OPTIONAL bei SPARQL)

Referenzen

- ▶ Redland Rasqal RDF Query Demonstration:
<http://librdf.org/query>
- ▶ => RDQL und SPARQL Unterstützung
- ▶ Umfangreicher Vergleich von RDF-Anfragesprachen:
- ▶ <http://www.aifb.uni-karlsruhe.de/WBS/pha/rdf-query/rdfquery.pdf>

Inhalt

Regelsprachen

- Regeln

- RuleML

- SWRL

- Prolog

Anfragesprachen

- Anfragesprachen

- SPARQL

- RDQL

- SeRQL

Fragen?

Danke!