



Seminar Moderne Webtechnologien

Semantic Web: Reasoners und Frameworks

Anne Augustin

Marco Kranz

Ralph Schäfermeier



Übersicht – 1. Einleitung

- **Jena Framework**
- **Demo zu Jena**
- **Reasoner**
- **Demo zu Pellet**
- **Semantic Web Client Library**



Jena Framework

Was ist Jena?

- Sammlung von Java Klassenbibliotheken zur Entwicklung von Semantic Web Anwendungen
- Open Source, aus dem HP Labs Semantic Web Programm hervorgegangen
- Seit Nov. 2001 als Projekt bei *sourceforge.net* registriert, zur Zeit 13 Entwickler





Jena Framework - Übersicht

Jena Features:

- **RDF API**
- **OWL API**
- **ARQ – Schnittstelle für SPARQL Anfragen**
- **Inference API & Engines**

RDF API - Übersicht



RDF API Grundlagen:

- `Model` als Basisklasse für RDF Graph

Standardoperationen:

- Hinzufügen von Ressourcen und Properties
- Hinzufügen von Tripeln
- Lesen / Schreiben von RDF Daten als RDF / XML oder N-Tripel



RDF API - Übersicht

Weitere Operationen:

- **Navigation innerhalb des Graphen über Iteratoren**
- **Anfragen auf dem Graphen**
 - **direkt über das Modell**
 - **über Klasse `SimpleSelector`**
- **Mengenoperationen auf Graphen (union, intersect, difference)**

Persistente Speicherung:

Jena bietet die Möglichkeit, eine Datenbank als Speichersystem zu verwenden.

- **Unterstützung aller gängigen Datenbanksysteme (Oracle, MySQL, ...)**
- **Datenbanktabellen werden automatisch angelegt**
- **gesamter Zugriff läuft über Schnittstelle**

Jena Framework - Datenbankschnittstelle



Vereinfachte Datenbankstruktur:

Statement Table

Subj.	Prop.	Obj.	GraphID
Varchar(n)	Varchar(n)	Varchar(n)	Integer
URI or pointer to Resource / Literal Table			ID

Long Resource Table

ResourceID	ResourceURI
Integer	Blob

Long Literal Table

LiteralID	LiteralURI
Integer	Blob



OWL API - Übersicht

OWL API Grundlagen:

OWL API setzt auf RDF API auf.

- `OntModel` als Basisklasse
- OWL API ist *sprachenunabhängig*, jedes Modell besitzt ein Profil, welches die Ontologiesprache angibt (RDFS, OWL full / DL / ...)
- Profil enthält zudem Informationen über Speicherart (in-memory / persistent) und verwendeten Reasoner
- bietet alle grundlegende Elemente für Darstellung von Ontologien (Klassen, Individuen, ...)



ARQ - Übersicht

ARQ Grundlagen:

Schnittstelle für Anfragen auf Model

- unterstützt verschiedene Anfragesprachen (SPARQL, RDQL, ARQ)
- unterstützt alle SPARQL-Anfragen (SELECT, CONSTRUCT, DESCRIBE, ASK)
- bietet neben standard query engine eine *remote access engine* sowie einen SQL rewriter
- die *remote access engine* ermöglicht Zugriff auf *RDF publishing server* (z.B. Joseki) via HTTP



ARQ - Übersicht

Anfrageverarbeitung:

- Anfragen auf einzelne Graphen oder *Datasets* (Menge von Graphen)
- Ergebnis einer *Anfrage* ist ein *Resultset* von Tripeln
- Daten liegen als RFD API Objekte vor, daher sind API-Aufrufe möglich
- Formatierung als *SPARQL query result XML* möglich



ARQ – Filter Functions

ARQ erweitern:

ARQ bietet verschiedene Erweiterungsmöglichkeiten

Bsp.: Filter Functions

1. Filterfunktion aus der *standard function library*

```
PREFIX jfn: <http://jena.hpl.hp.com/function#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?x
{ ?x dc:date ?date .
  FILTER (?date < jfn:now() )
}
```



ARQ - Filter Functions

Selbstdefinierte Filterfunktionen:

Filterfunktionen können je nach Anwendung selbst definiert und Dynamisch eingebunden werden.

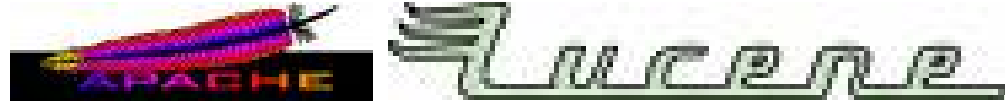
2. Selbstdefinierte Filterfunktion

```
PREFIX f: <java:app.myFunctions.>  
PREFIX dc: <http://purl.org/dc/elements/1.1/>  
SELECT ?x  
{ ?x dc:date ?date .  
  FILTER (?date < f:computeDate() )  
}
```

LARQ – ARC meets Lucene



LARQ:



LARQ ist eine Kombination aus ARQ und Apache Lucene.

Lucene:

Java-Bibliothek zur Volltextsuche inklusive Suchengine.

Verwendung:

- Indexierung von Stringliteralen
- ermöglicht dadurch Volltextsuche über Teilmengen von Ressourcen
- Bsp.: Ontologie über Bücher, Suche nach allen Büchern mit einem bestimmten Wort im Volltext



Jena Inference API

Inference API:

- **Bietet eine Schnittstelle für Reasoner**
- **Setzt auf Graph API auf (Basis für jedes *Model*), Reasoner sind dadurch unabhängig vom *Model* (RDF, Ontology, ...)**

Enthaltene Reasoner:

- **Transitive reasoner**
- **RDFS rule reasoner**
- **OWL, OWL Mini, OWL Micro Reasoners**
- **DAML micro reasoner**
- **Generic rule reasoner**



Jena Framework - Demo

- **Jena Framework**
- **Demo zu Jena**
- **Reasoner**
- **Demo zu Pellet**
- **Semantic Web Client Library**



Demo-Ontologie: Familie



veritas
iustitia
libertas

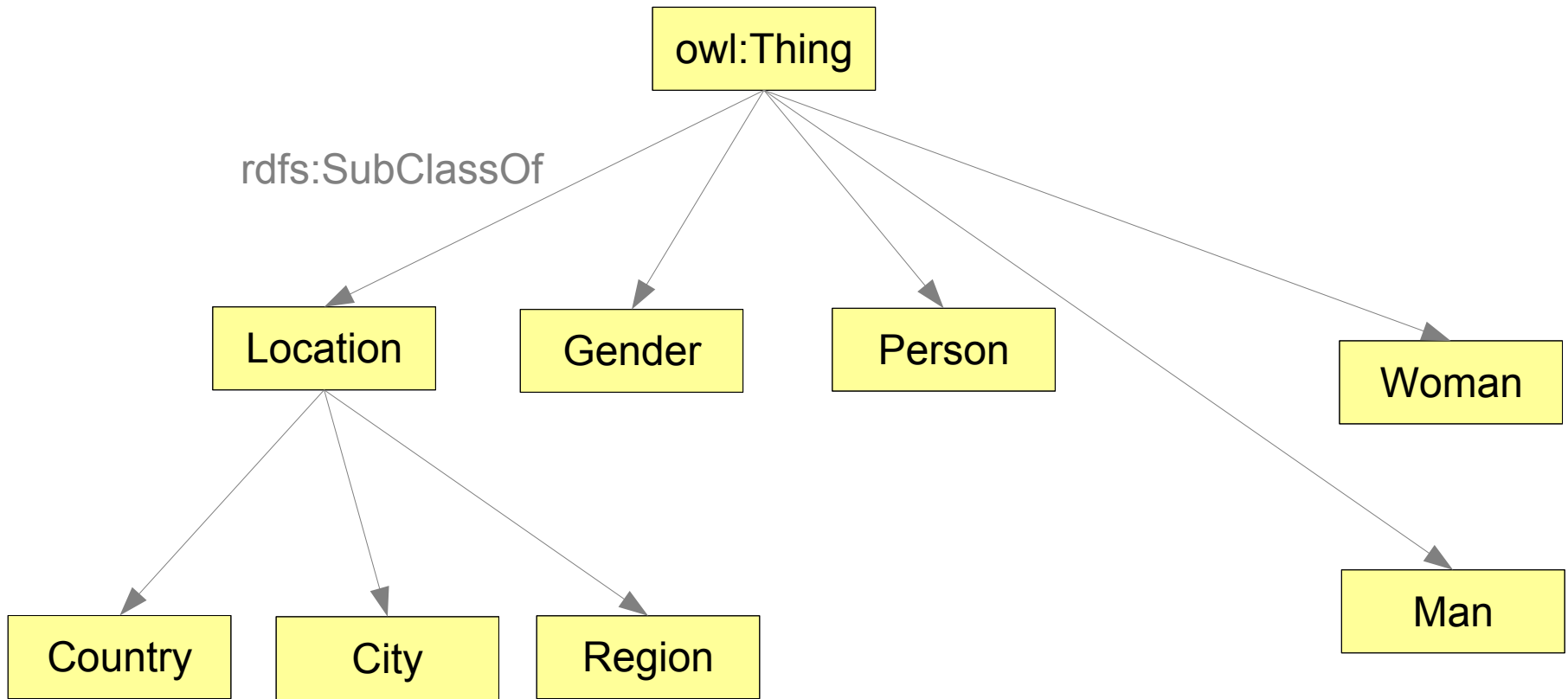


Demo-Ontologie: Wohnorte





Demo-Ontologie: OWL-Klassen



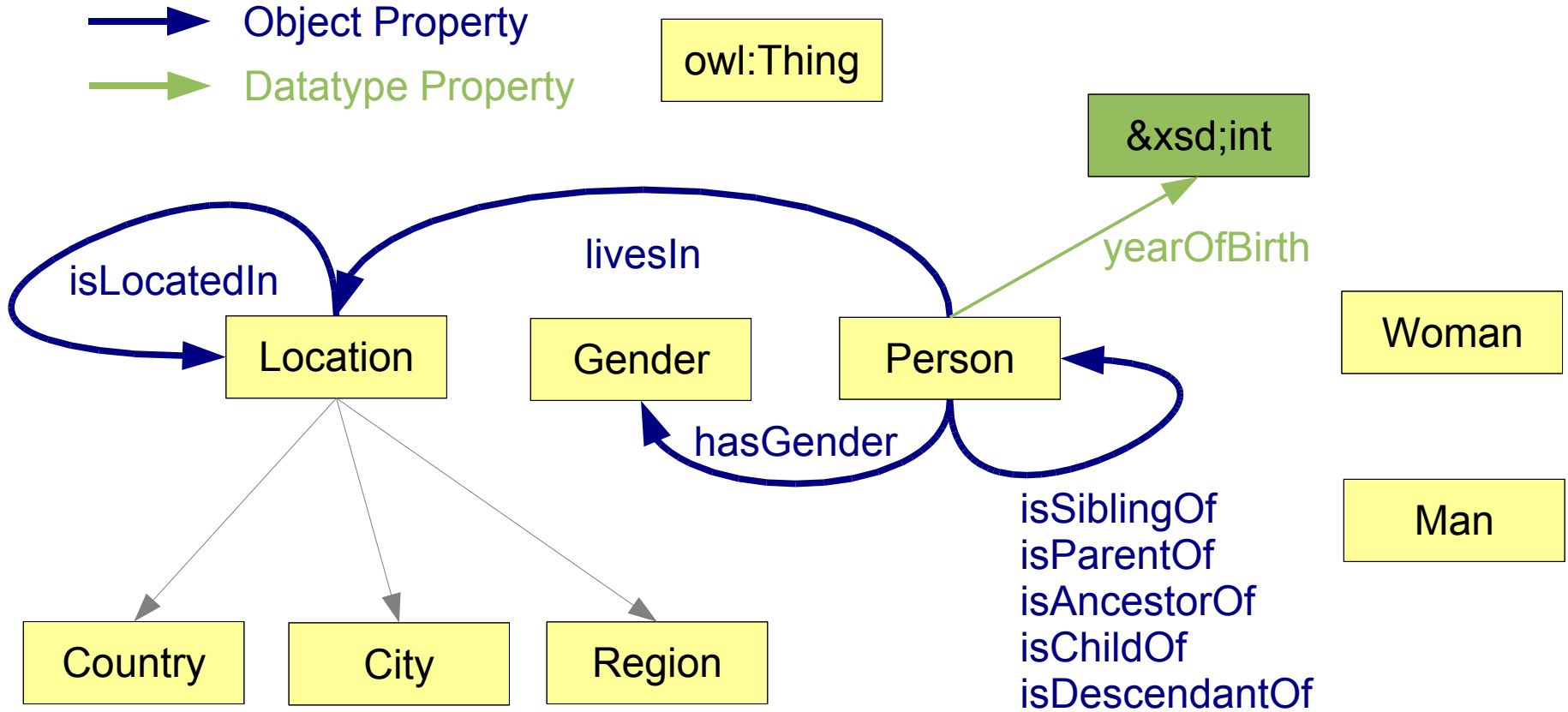


Demo-Ontologie: Eigenschaften

➡ Object Property

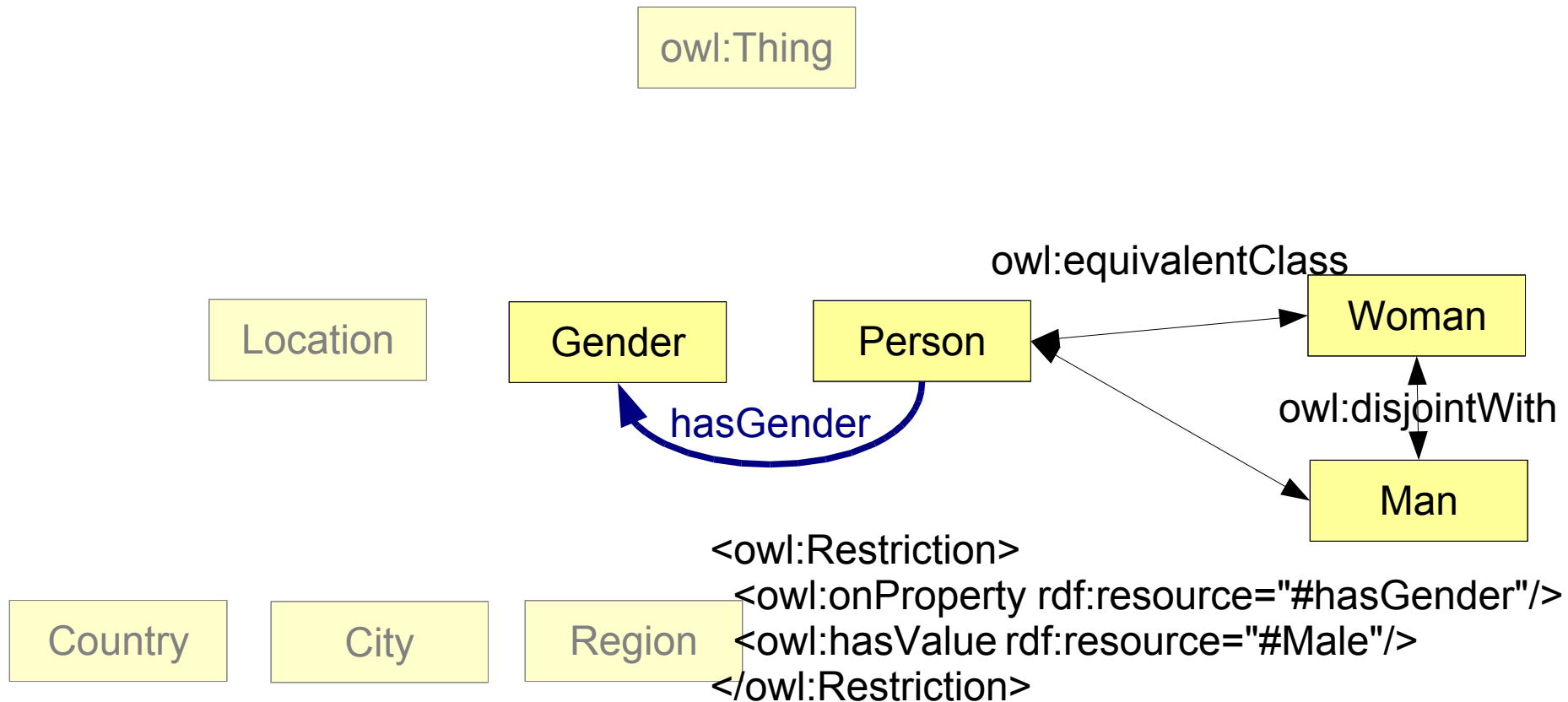
➡ Datatype Property

owl:Thing





Demo-Ontologie: Äquivalente Klassen / Einschränkungen





Demo-Applikation

- Zugriff über Java BeanShell
- Einlesen:

```
OntModelSpec modelSpec = PelletReasonerFactory.THE_SPEC;
```

```
model = ModelFactory.createOntologyModel(modelSpec);
```

```
InputStream in =  
FileManager.get().open("Ontology/family.owl");
```

```
model.read(in, "");
```



Demo-Applikation

- Namensraum – Präfixe:

```
nsfam = model.getNsPrefixURI("");
```

```
nsrdf = model.getNsPrefixURI("rdf");
```

```
nsrdfs = model.getNsPrefixURI("rdfs");
```

```
nsowl = model.getNsPrefixURI("owl");
```



Demo-Applikation

Demo als Java WebStart Applikation:

<http://page.mi.fu-berlin.de/~schaef/webtech/demo/Demo.jnlp>

```
Webtechnologien Demo
Jena API Demo Reasoner Demo
BeanShell
2.0b4 - by Pat Niemeyer (pat@pat.net)
bsh % print (model.listClasses());
http://www.mi.fu-berlin.de/webtechnologien/family.owl#Man
http://www.mi.fu-berlin.de/webtechnologien/family.owl#Location
http://www.w3.org/2002/07/owl#Nothing
http://www.mi.fu-berlin.de/webtechnologien/family.owl#City
http://www.w3.org/2002/07/owl#Thing
http://www.mi.fu-berlin.de/webtechnologien/family.owl#Country
http://www.mi.fu-berlin.de/webtechnologien/family.owl#Woman
http://www.mi.fu-berlin.de/webtechnologien/family.owl#Region
http://www.mi.fu-berlin.de/webtechnologien/family.owl#Person
http://www.mi.fu-berlin.de/webtechnologien/family.owl#Gender
-706c66f9:11385bc09a3:-7ff7
-706c66f9:11385bc09a3:-7ffb
-706c66f9:11385bc09a3:-7ff6
-706c66f9:11385bc09a3:-7ffa
bsh %
```



Übersicht – 3. Reasoner - Pellet

- **Jena Framework**
- **Demo zu Jena**
- **Reasoner**
- **Demo zu Pellet**
- **Semantic Web Client Library**



Reasoners und Frameworks

Reasoner

- 1. Grundlagen der Reasoner**
- 2. Tableaux Reasoning**
- 3. Pellet**
- 4. andere Reasoner**
- 5. Vergleich & Zusammenfassung**



veritas
iustitia
libertas

Grundlagen der Reasoner

Was ist überhaupt ein Reasoner?

Motivation

Semantic Web: Ansammlung von *semantischen* Informationen

Ziel: Automatische Inferenz von und Suche nach Wissen

Voraussetzung: Formulierung des Wissens in formaler Sprache, die Inferenz erlaubt (Description Logic)



Grundlagen der Reasoner

Reasoner = (Theorem) Prover = Logical Inference Engine

Kann aus logischen Axiomen und Aussagen Schlüsse ziehen

Überprüft die Konsistenz von Ontologien

Wertet logische Ableitungsregeln aus und erzeugt so Informationen, die nicht explizit in der Ontologie vorhanden sind

Verschiedene Implementierungen verfügbar -> Teil 3+4 des Vortrags

veritas
iustitia
libertas



Grundlagen der Reasoner

Kleine Einführung in Description Logic(DL)



Grundlagen der Reasoner

DL ist eine Prädikatenlogik erster Stufe.

DL benutzt als Sprachmittel Konzepte, Rollen und Individuen, um Domänen zu beschreiben.

Die Syntax: Konzepte/Klassen

Rollen

Konstruktoren/Operatoren: aus Konstruktoren lassen sich neue Konzepte oder Rollen aus schon bestehenden Rollen und Konzepten definieren

Grundlagen der Reasoner



Concepts		
ALC	Atomic	A, B
	Not	$\neg C$
	And	$C \sqcap D$
	Or	$C \sqcup D$
	Exists	$\exists R.C$
	For all	$\forall R.C$
Q(N)	At least	$\geq n R.C$ ($\geq n R$)
	At most	$\leq n R.C$ ($\leq n R$)
O	Nominal	$\{i_1, \dots, i_n\}$

Roles	
Atomic	R
Inverse	R^-

Ontology (=Knowledge Base)

Concept Axioms (TBox)

Subclass	$C \sqsubseteq D$
Equivalent	$C \equiv D$

Role Axioms (RBox)

\sqsubseteq Subrole	$R \sqsubseteq S$
\mathcal{S} Transitivity	$\text{Trans}(S)$

Assertional Axioms (ABox)

Instance	$C(a)$
Role	$R(a, b)$
Same	$a = b$
Different	$a \neq b$

Entnommen aus:
Vorlesung „Web Ontology
Language Semantik“,
Hitzler, Sure, Ankolekar
TH Karlsruhe



Grundlagen der Reasoner

Kleine Einführung in Description Logic(DL)

Beispiel:

Konzepte/Klassen: Person, Kind

Rollen/Property: hatKind(Person,Kind)

Konstruktoren/Operatoren: Elternteil

eine Person, die ein Kind hat

Elternteil \equiv Person \sqcap \exists hatKind.Kind

Knowledge Base

Definition: Knowledge Base(KB) = T-Box + A-Box (+ R-Box)

Tbox: beinhaltet allgemeine Aussagen und Axiome
über Konzepte(Klassen) und Rollen

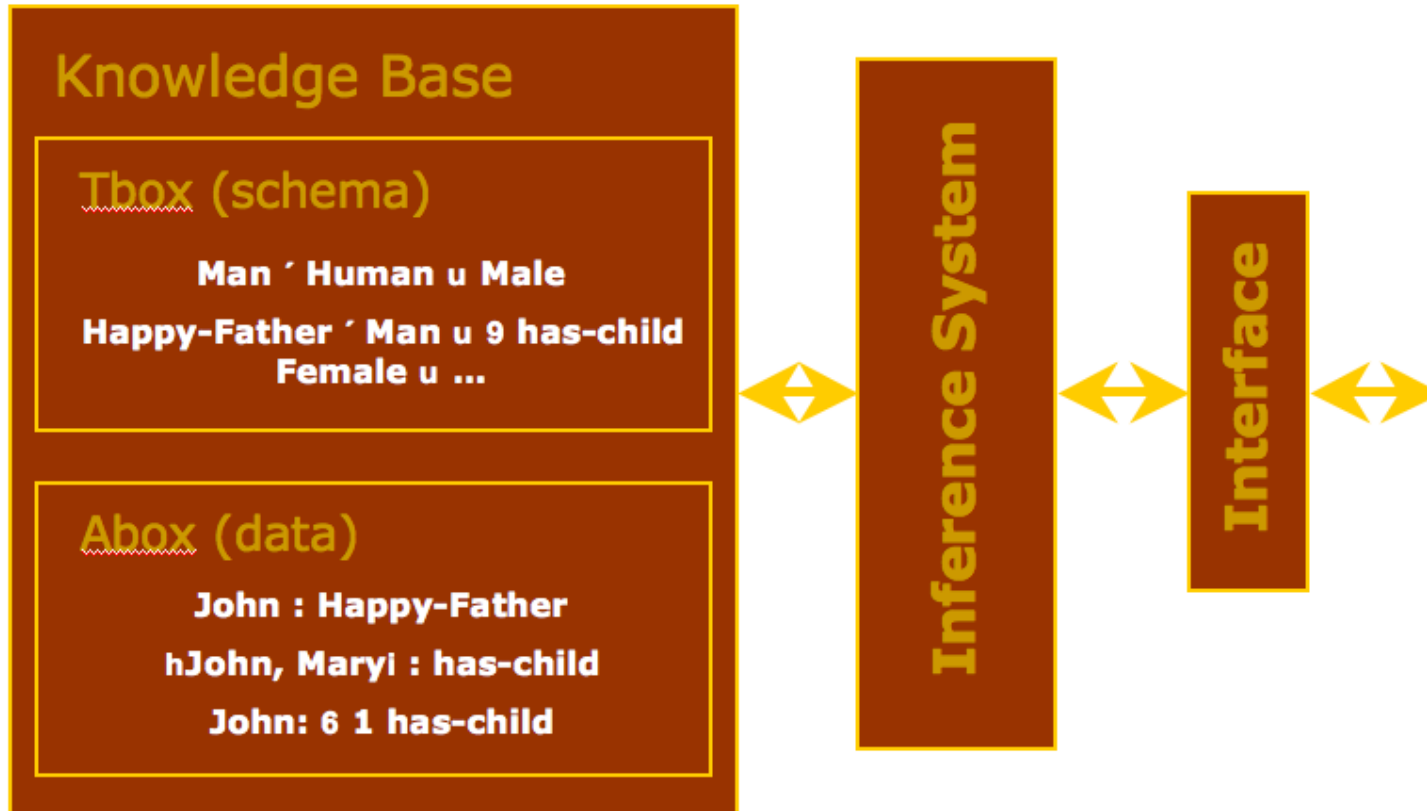
Abox: beinhaltet Aussagen über Individuen von Klassen/Konzepten

Ein Reasoner kann nun aus einer Knowledge Base logische Schlüsse ziehen oder sie auf Konsistenz überprüfen.



Grundlagen der Reasoner

DL Architecture





Grundlagen der Reasoner

Beispiel

Tbox-Aussage: Ein Hund ist ein Tier.

Abox-Aussage: Fiffi ist ein Hund.

Hund \sqsubseteq Tier

Fiffi: Hund

Der Reasoner kann jetzt schlussfolgern:

Fiffi ist ein Tier.

Fiffi: Tier



Grundlagen der Reasoner

Beispiel 2

Tbox: Ein Hund ist ein Tier.
Tier \neq Mensch

Hund \sqsubseteq Tier
Tier $\sqsubseteq \neg$ Mensch

Abox: Fiffi ist ein Hund.
Fiffi ist ein Mensch.

Fiffi: Hund
Fiffi: Tier

Der Reasoner kann nun feststellen, dass Widersprüche in KB vorhanden

veritas
iustitia
libertas



Tableaux Reasoning

Tableaux Reasoning - Einfache logische Inferenz



Tableaux Reasoning

Tableaux Reasoning

Viele Reasoning-Aufgaben kann man auf die Erfüllbarkeit der Knowledge Base zurückführen.

Konsistenz = Erfüllbarkeit

Tableaux-Algorithmen prüfen die (Un)erfüllbarkeit einer KB

Ist ϕ erfüllbar?

-> Teste ob nicht ϕ unerfüllbar ist.



Tableaux Reasoning

Arten von Reasoning

Klassenkonsistenz

- $KB \cup \{C(a)\}$ unerfüllbar (a neu)

$$C \equiv \perp \text{ gdw}$$

Klasseninklusion (Subsumption)

- $KB \cup \{C \sqcap \neg D(a)\}$ unerfüllbar (a neu)

$$C \sqsubseteq D \text{ gdw}$$

Klassenäquivalenz

- $C \sqsubseteq D$ und $D \sqsubseteq C$

$$C \equiv D \text{ gdw}$$

Klassendisjunktheit

- $KB \cup \{(C \sqcap D)(a)\}$ unerfüllbar (a neu)

$$C \sqcap D = \perp \text{ gdw}$$

Klassenzugehörigkeit

- $KB \cup \{\neg C(a)\}$ unerfüllbar (a neu)

$$C(a) \text{ gdw}$$

Instanzgenerierung (Retrieval)

- Prüfe Klassenzugehörigkeit für alle Individuen.
- Schwerer, dies gut zu implementieren!

alle $C(X)$ finden

veritas
iustitia
libertas



Tableaux Reasoning

Konsistenzcheck online

The screenshot shows a Safari browser window with the address bar displaying `http://www.mindswap.org/2003/pellet/demo.shtml`. The page title is "Pellet OWL Reasoner - Online Demo". The main content area features a blue header with the text "Pellet has been moved to <http://pellet.owl.com>". Below this, there is a navigation menu with links for Overview, Project Page, Download, Support, FAQ, Online demo, Performance, and Pellet Widget. The main heading is "OWL Consistency Checker".

The instructions state: "Enter a URI or paste an OWL document into the following text field to check the consistency of an OWL ontology. The level of the input file (Lite, DL or Full) will also be shown. Additionally, you can specify a URI of another file to check if all the triples in this conclusions file is entailed from the triples in input file. The classification of all the concepts in the input file can also be printed in a tree or table format. Please send your comments, questions and problems to [Pellet users list](#). Before posting messages, you need to first [subscribe](#) to the mailing list."

There are several input fields and buttons:

- Examples:** A dropdown menu is open, showing a list of 9 examples. The first option, "Please choose an example and click submit to see different ways to use the demo", is selected.
- Input:** A text field for entering a URI or OWL document.
- URI:** A text field for specifying a URI of another file.
- or Text:** A text field for pasting an OWL document.
- Options:** A list of checkboxes and a dropdown menu. The checked options are "Enable species validation" and "Check ontology consistency". The "Display class hierarchy" dropdown is set to "NO".
- New SPARQL Query:** A text field for entering a SPARQL or RDQL query.

Buttons for "Clear All", "Submit", and "SPARQL" are visible. The browser's status bar at the bottom shows the time as 01:26:58 and the date as So 22:46.



Tableaux Reasoning

Tableaux-Algorithmen

Ein Modell ist eine konkrete Belegung, die alle T-Box Aussagen zu wahren Aussagen macht.

Tableaux Algorithmen prüfen die Konsistenz (Erfüllbarkeit).

Tableaux Algorithmen bauen zunächst ein Modell einer KB, das konsistent ist zu allen KB-Aussagen:

- Starte mit gegebenen Basisfakten: ABox-Aussagen in der KB
- Vervollständige Modell sukzessive durch Regeln aus TBox



Tableaux Reasoning

Beispiel: Tableaux-Reasoning

Tbox:

Ein HappyParent ist eine Person, deren Kinder alle Doctor sind oder mindestens ein Kind haben, das Doctor ist.

$\text{HappyParent} \equiv \text{Person} \sqcap \forall \text{hasChild} . (\text{Doctor} \sqcup \exists \text{hasChild} . \text{Doctor})$,

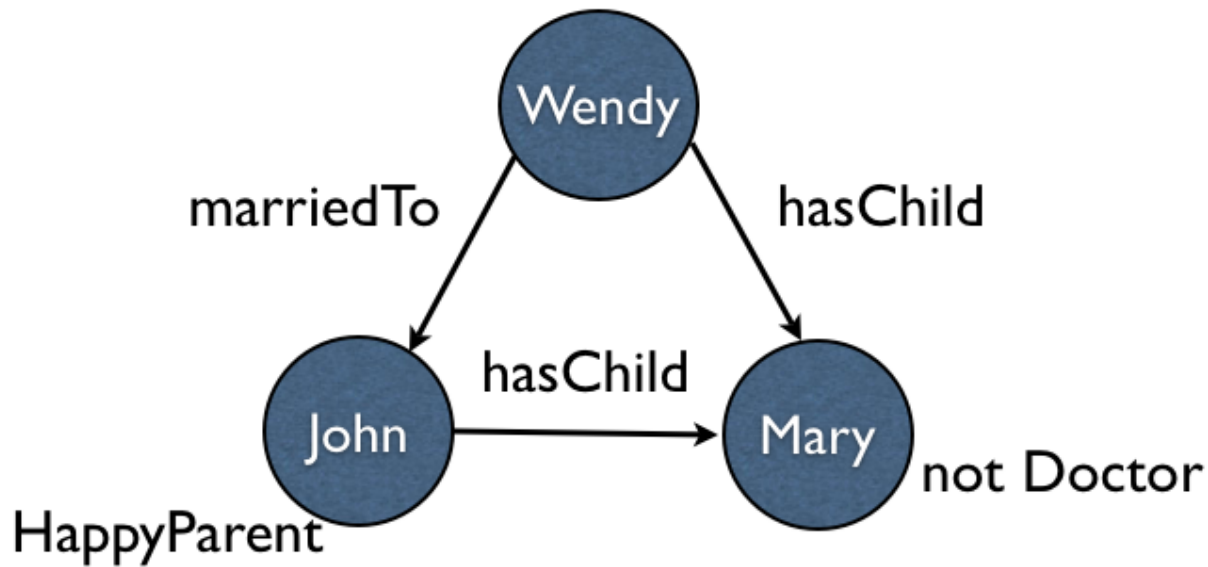
Abox: John: HappyParent
John hasChild Mary
Mary:: Doctor
Wendy hasChild Mary
Wendy marriedTo John

John ist ein HappyParent
John hat Kind Mary
Mary ist kein Doctor
Wendy hat Kind Mary
Wendy ist mit John verheiratet



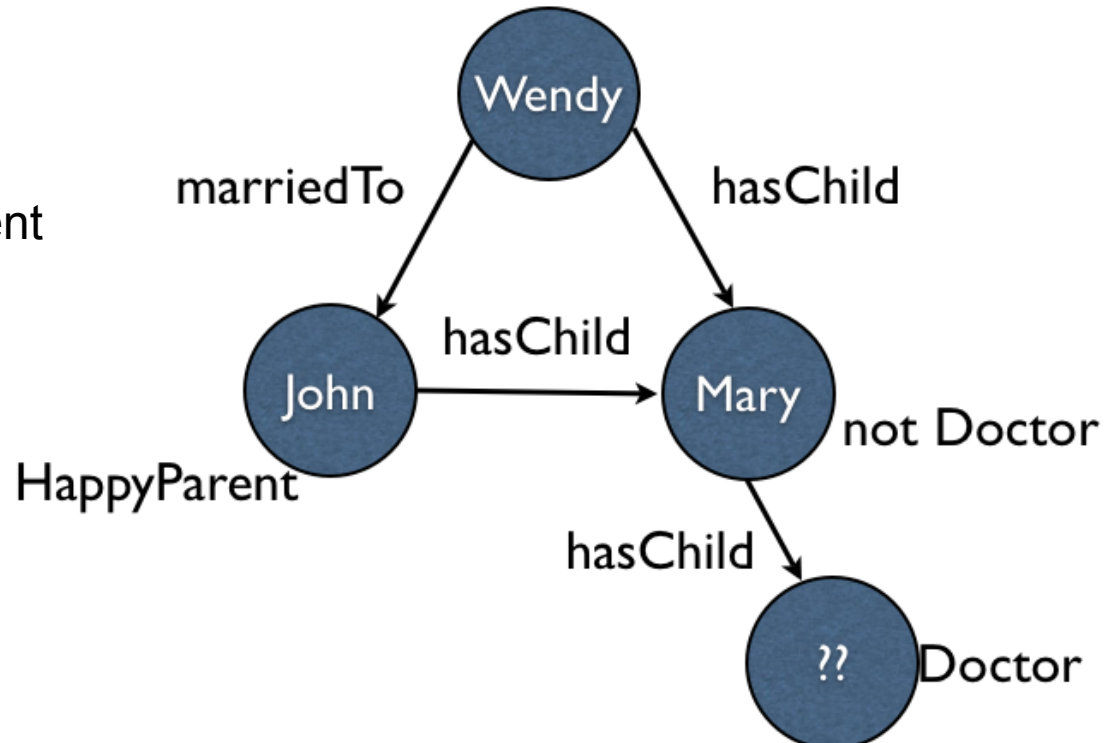
Tableaux Reasoning

Schritt1: Abox- Aussagen modellieren



Schritt2: Tbox-Axiome anwenden

Finde erfüllbares
Modell:
Tbox-Aussage HappyParent
muss für John
erfüllt sein



veritas
iustitia
libertas



Pellet: Ein Reasoner



Wie arbeitet Pellet?

veritas
iustitia
libertas



Pellet: Ein Reasoner



Pellet

Entwickelt von Evren Sirin und Bijan Parsia, U MD, USA

In Java implementiert

Versteht OWL Lite und OWL DL Syntax

Beide Sprachen basieren auf Description Logic.

arbeitet mit Tableaux-Algorithmen für A-Box- und T-Box-Reasoning

Korrekt und vollständig

Bedienung über Kommandozeile

APIs für OWL und Jena

Bestandteil des offiziellen SWOOP Editors für OWL

<http://www.mindswap.org/2003/pellet/>

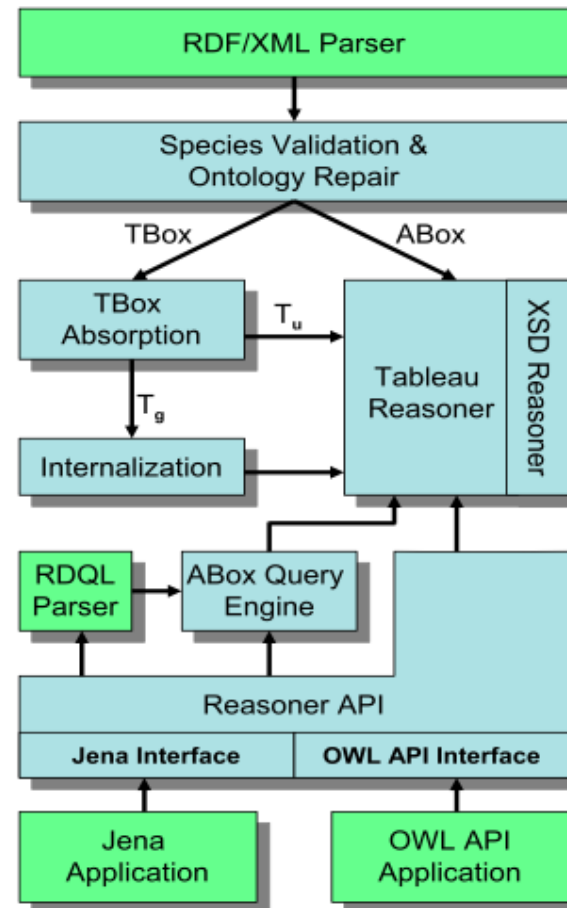
Pellet: Ein Reasoner

Architektur von Pellet

1. OWL Ontologie wird über RDF-Tripel in KB Aussagen konvertiert
2. Dabei werden Axiome über Klassen in der Tbox-Komponente gespeichert, Aussagen über Individuen in der Abox Komponente.

Handelt es sich bei der Ontologie um ein OWL Full-Dokument, versucht Pellet vorerst es in OWL DL umzuwandeln.

3. Der Tableaux-Reasoner verwendet nun standardmäßiges Tableaux-Reasoning mit zusätzlichen Standard-Optimierungen.



Grafik entnommen aus:

Pellet: An OWL DL Reasoner
Bijan Parsia and Evren Sirin
MINDSWAP Research Group

University of Maryland, College Park, MD

veritas
iustitia
libertas



Andere Reasoner

Andere Reasoner



FaCT++

FaCT++

Entwickelt von:

Dmitry Tsarkov an der University of Manchester

Reimplementierung von etablierten FaCT Reasoning Algorithmen in C++

Traditionelle, optimierte Reasoner Architektur --> wenig Benutzerinteraktivität

Standard T-Box Reasoning

Unterstützt OWL DL mit ABoxes

FaCT++ ist ein korrekter, vollständiger Reasoner

Unterstützt keine Anfragesprachen!

<http://owl.man.ac.uk/factplusplus/>



RacerPro



RacerPro

Volker Haarslev, Concordia University, Canada und
Ralf Möller, TU Hamburg-Harburg

Implementierung von Tableaux- Algorithmen
Unterstützt mehrere A-Boxes und T-Boxes

Benutzerinteraktion über Client-Server oder CLI (command line interface)
OWL DL + OWL Lite KB Unterstützung
Kompatibel zu anderen Sprachstandards (z.B. FaCT-XML, DAML+OIL)

Unterstützt Anfragesprache nRQL (new Racer Query Language)

A-Box „publish-subscribe retrieval“: Antwortet nach jedem gefundenen Treffer

<http://www.sts.tu-harburg.de/~r.f.moeller/racer/>



KAON2



KAON2

Infrastruktur für OWL-DL, SWRL, F-Logik Ontologien.

Völlig neuer Ansatz:

Implementiert durch neue Algorithmen, die eine KB auf ein disjunktives Datalog Programm reduzieren

Implementiert in Java, kommerzielle und freie Distribution verfügbar

Beweisbar vollständig und korrektes Reasoning in KAON2

<http://kaon2.semanticweb.org/>



KAON2



KAON2

Optimierte Rechenzeit Effizienz für große A-Boxes

Weniger umfangreiche T-Box Dienste

Extraktion von Wissen aus relationalen Datenbanken möglich

Unterstützung für Anfragesprache: SPARQL



Hoolet

Hoolet

Entwickelt von: Dmitry Tsarkov, University of Manchester

Standard T-Box und A-Box Anfragen (ohne Schachtelung)

Benutzt als einziger einen First Order Beweiser/Prover: Vampire

Vampire ist ein korrekter, aber unvollständiger FOL Beweiser/Prover

Übersetzt Ontologie in First Order Logic Axiome

--> Vampire: Konsistenzprüfung


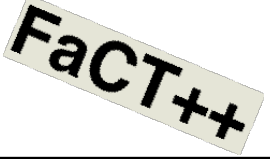
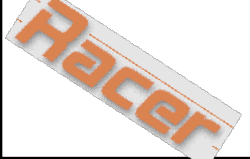


Benutzt die WonderWeb OWL API zum Analysieren und Verarbeiten von OWL-Dokumenten

Java GUI für Übersetzung von und Anfragen an OWL Ontologien

Vollständige OWL DL Unterstützung

<http://owl.man.ac.uk/hoolet/>

Vergleich & Zusammenfassung

					
Korrekt / Vollständig	✓	✓	✓	✓	Nur korrekt
Benutzer Schnittstelle	CLI, Jena API	Servlet, DIG reasoner	Client-Server, CLI	Umfangreich + relationale DB	Java GUI, CLI
A-/T-Box	✓	✓	✓	✓	✓
Kompatibilität	OWL DL	FaCT ++ Syntax	OWL DL, FaCT, DAML + OIL, KRSS	OWL DL, SWRL, F-Logik	OWL DL
Anfragesprachen	RDQL	✗	nRQL	SPARQL	✗



Literatur

Literatur

Thorsten Liebig: „Reasoning mit OWL - System Support and Insights“,
Technical Report, Universität Ulm, September 2006

Bijan Parsia und Evren Sirin: „Pellet: An OWL DL Reasoner“,
Description Logics 2004

V. Haarslev und R. Moeller: "Racer: A core inference engine for the
Semantic Web", In 2nd International Workshop on Evaluation of
Ontology-based Tools 2003

Taowei David Wang, Bijan Parsia, James A. Hendler: „A survey of the web
Ontology Landscape“, International Semantic Web Conference 2006



Übersicht – 4. Demo zu Pellet

- **Jena Framework**
- **Demo zu Jena**
- **Reasoner**
- **Demo zu Pellet**
- **Semantic Web Client Library**

veritas
iustitia
libertas



Demo zu Pellet

Webtechnologien Demo

Jena API Demo Reasoner Demo

Städte Städte Save


```
SELECT ?person ?city
WHERE {
  {
    ?person family:livesIn ?location.
    ?location family:isLocatedIn ?city.
    ?city rdf:type family:City
  } UNION {
    ?person family:livesIn ?city.
    ?city rdf:type family:City
  }
}
```

```
( ?person = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#Agneta> ) (
?city = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#Stockholm> )
( ?person = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#Demeter> ) (
?city = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#Sydney> )
( ?person = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#Bree> ) (
?city = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#London> )
( ?person = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#Chuck> ) (
?city = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#SanDiego> )
( ?person = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#Bettina> ) (
?city = <http://www.mi.fu-berlin.de/webtechnologien/family.owl#London> )
```



Übersicht – 5. Semantic Web Client Library


- **Jena Framework**
- **Demo zu Jena**
- **Reasoner**
- **Demo zu Pellet**
- **Semantic Web Client Library**



Semantic Web Client Library

Semantic Web Client Library - Projektübersicht:

- **entwickelt am FB WiWiss der FU Berlin von Dr. Chris Bizer, Tobias Gauß und Richard Cyganiak**
- **Teil des Projekts *NG4J – Named Graph API for Jena***
- **seit Sep. 2004 als Projekt bei *sourceforge.net* registriert**
- **basiert auf Jena Framework**




Semantic Web Client Library

Semantic Web Client Library:

Eine Java-Bibliothek, die SPARQL und *find*-Anfragen auf dem Semantic Web ermöglicht.

- **Grundgedanke: Alle als RDF / OWL veröffentlichten Daten können als ein großer Graph betrachtet werden**
- **dynamischer Aufbau des RDF-Graphen während der Anfragebearbeitung**
- **Graph entsteht durch Dereferenzierung von HTTP URI's**



Semantic Web Client Library

Kommandozeilentool:

Beispielapplikation, welche SPARQL- und *find*-Anfragen über die Kommandozeile ermöglicht

Beispielanfrage: *finde Namen und Homepages der Freunde von Richard Cyganiak und Namen und Homepages ihrer Freunde*

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

```
SELECT DISTINCT ?friendsname ?friendshomepage ?foafsname ?foafshomepage
```

```
WHERE {
```

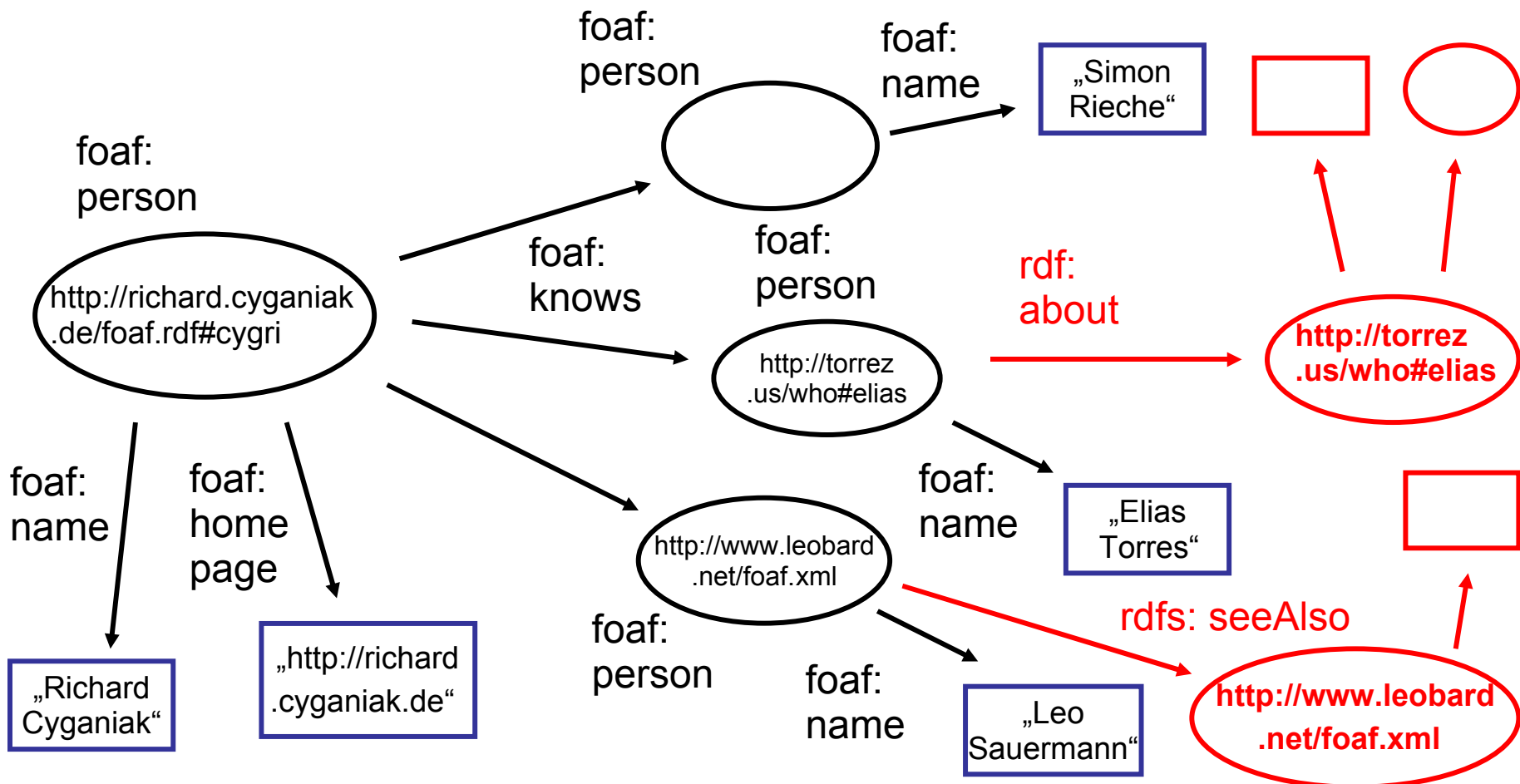
```
  { <http://richard.cyganiak.de/foaf.rdf#cygri> foaf:knows ?friend .  
    ?friend foaf:mbox_shalsum ?mbox .  
    ?friendsURI foaf:mbox_shalsum ?mbox .  
    ?friendsURI foaf:name ?friendsname .  
    ?friendsURI foaf:homepage ?friendshomepage . }  
  OPTIONAL { ?friendsURI foaf:knows ?foaf .  
             ?foaf foaf:name ?foafsname .  
             ?foaf foaf:homepage ?foafshomepage .
```

```
  }
```

```
}
```

Semantic Web Client Library

Beispiel für dynamischen Graphenaufbau:





Quellen

Jena Framework: <http://jena.sourceforge.net/>

Apache Lucene: <http://lucene.apache.org/>

Semantic Web Client Library:
<http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/semwebclient/>

veritas
iustitia
libertas



Vielen Dank!