SPARQL – Syntax und Intuition

Pascal Hitzler Markus Krötzsch Sebastian Rudolph

Institut AIFB · Universität Karlsruhe

Semantic Web Technologies 1 (WS07/08)

9. Januar 2008

http://semantic-web-grundlagen.de

AIFB

Outline

AIFB

Semantic Web Technologies 1

1. Einleitung und Ausblick
2. XML und URIs
3. Einführung in RDF
4. RDF Schema
5. Logik – Grundlagen
6. Semantik von RDF(S)
7. OWL – Syntax und Intuition
8. OWL – Semantik und Reasoning
9. SPARQL – Syntax und Intuition (→ Webseite dieser Vorlesung)
10. Semantik von SPARQL
11. Konjunktive Anfragen und Regelsprachen
12. OWL 1.1 – Syntax und Semantik
13. Bericht aus der Praxis
14. Semantic Web – Anwendungen

Literatur zu dieser Vorlesung online siehe
→ Semantic Web – Grundlagen, Kapitel 7

AIFB

Anfragesprachen für das Semantic Web?

Wie kann auf in RDF oder OWL spezifizierte Informationen zugegriffen werden?

AIFB

Anfragesprachen für das Semantic Web?

Wie kann auf in RDF oder OWL spezifizierte Informationen zugegriffen werden?

Abfrage von Informationen in RDF(S)

- Einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung

„Folgt ein bestimmter RDF-Graph aus einem gegebenen?“

AIFB

Anfragesprachen für das Semantic Web?

Wie kann auf in RDF oder OWL spezifizierte Informationen zugegriffen werden?

Abfrage von Informationen in RDF(S)

- Einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung

„Folgt ein bestimmter RDF-Graph aus einem gegebenen?“

Abfrage von Informationen in OWL

- Logisches Schließen

„Folgt eine Subklassen-Beziehung aus einer OWL-Ontologie?“
„Welches sind die Instanzen einer Klasse einer OWL-Ontologie?“

AIFB

Genügen OWL und RDF nicht?

Selbst OWL ist als Anfragesprache oft zu schwach

- „Welche Zeichenketten in deutscher Sprache sind in der Ontologie angegeben?“
- „Welche Propertys verbinden zwei bestimmte Indiviuen?“
- „Welche Paare von Personen haben eine gemeinsames Elternteil?“

$\rightsquigarrow$ weder in RDF noch in OWL ausdrückbar.

Genügen OWL und RDF nicht?

Selbst OWL ist als Anfragesprache oft zu schwach

- „Welche Zeichenketten in deutscher Sprache sind in der Ontologie angegeben?“
- „Welche Propertys verbinden zwei bestimmte Indiviuen?“
- „Welche Paare von Personen haben eine gemeinsames Elternteil?“

$\rightsquigarrow$ weder in RDF noch in OWL ausdrückbar.

Anforderungen:

- Große Ausdruckstärke zur Beschreibung der gefragten Information
- Möglichkeiten zur Formatierung, Einschränkung und Manipulation der Ergebnisse

AIFB

Übersicht

Agenda für diese und die folgende Vorlesung:

Vorlesung 9:

- Grundlagen der RDF-Anfragesprache SPARQL

Vorlesung 10:

- Semantik der RDF-Anfragesprache SPARQL
- Konjunktive Anfragen für OWL

AIFB

Outline

AIFB

SPARQL

SPARQL (sprich engl. *sparkle*) steht für
SPARQL Protocol And RDF Query Language

- W3C-Spezifikation kurz vor Standardisierung
- Anfragsprache zur *Abfrage von Instanzen aus RDF-Dokumenten*
- schon heute große praktische Bedeutung

Teile der SPARQL-Spezifikation

- Anfragesprache: Thema dieser Vorlesung
- Ergebnisformat: Darstellung von Ergebnissen in XML
- Anfrageprotokoll: Übermittlung von Anfragen und Ergebnissen

AIFB

Einfache Anfragen

Eine einfache Beispielanfrage:

```
PREFIX ex:  <http://example.org/>
SELECT ?titel ?autor
WHERE
  { ?buch    ex:VerlegtBei  <http://springer.com/Verlag> .
    ?buch    ex:Titel       ?titel .
    ?buch    ex:Autor       ?autor . }
```

Einfache Anfragen

Eine einfache Beispielanfrage:

```
PREFIX ex:  <http://example.org/>
SELECT ?titel ?autor
WHERE
  { ?buch    ex:VerlegtBei  <http://springer.com/Verlag> .
    ?buch    ex:Titel       ?titel .
    ?buch    ex:Autor       ?autor . }
```

- Hauptbestandteil ist ein Anfragemuster (`WHERE`)
  - ⇝ Anfragemuster verwenden die Turtle-Syntax für RDF
  - ⇝ Muster dürfen Variablen enthalten (`?variable`)
- Kurzschreibweisen für URIs möglich (`PREFIX`)
- Anfrageergebnis durch Auswahl von Variablen (`SELECT`)

AIFB

Beispielergebnis

Beispiel RDF-Dokument:

```
@prefix  ex: <http://example.org/> .
ex:SemanticWeb ex:VerlegtBei <http://springer.com/Verlag> ;
               ex:Titel      "Semantic Web - Grundlagen" ;
               ex:Autor      ex:Hitzler, ex:Krötzsch,
                             ex:Rudolph, ex:Sure .
```

Ergebnis der Anfrage: Tabelle mit einer Zeile je Ergebnis

titel	autor
"Semantic Web - Grundlagen"	http://example.org/Hitzler
"Semantic Web - Grundlagen"	http://example.org/Krötzsch
"Semantic Web - Grundlagen"	http://example.org/Rudolph
"Semantic Web - Grundlagen"	http://example.org/Sure

AIFB

Einfache Graph-Muster

Die grundlegenden Anfragemuster sind einfache Graph-Muster

- Menge von RDF-Tripeln in Turtle-Syntax
- Turtle-Abkürzungen (mittels , und ;) zulässig
- Variablen werden durch ? oder $ gekennzeichnet (`?variable` hat gleiche Bedeutung wie `$variable`)
- Variablen zulässig als Subjekt, Prädikat oder Objekt

AIFB

Einfache Graph-Muster

Die grundlegenden Anfragemuster sind einfache Graph-Muster

- Menge von RDF-Tripeln in Turtle-Syntax
- Turtle-Abkürzungen (mittels , und ;) zulässig
- Variablen werden durch ? oder $ gekennzeichnet (`?variable` hat gleiche Bedeutung wie `$variable`)
- Variablen zulässig als Subjekt, Prädikat oder Objekt

Zulässig $\neq$ lesbar:

```
PREFIX ex:  <http://example.org/>
SELECT $rf456df ?_AIFB WHERE { ?ef3a_3 ex:VerlegtBei
<http://springer.com/Verlag> . ?ef3a_3 ex:Titel
       ?rf456df . $ef3a_3    ex:Autor ?_AIFB . }
```

(semantisch äquivalent zur vorherigen Anfrage)

AIFB

Leere Knoten

Was bedeuten leere Knoten in SPARQL?

Leere Knoten in Anfragemustern:

- Zulässig als Subjekt oder Objekt
- ID beliebig, aber niemals gleiche ID mehrfach pro Anfrage
- Verhalten sich wie Variablen, die nicht ausgewählt werden können

AIFB

Leere Knoten

Was bedeuten leere Knoten in SPARQL?

Leere Knoten in Anfragemustern:

- Zulässig als Subjekt oder Objekt
- ID beliebig, aber niemals gleiche ID mehrfach pro Anfrage
- Verhalten sich wie Variablen, die nicht ausgewählt werden können

Leere Knoten in Ergebnissen:

- Platzhalter für unbekannte Elemente
- IDs beliebig, aber eventuell an andere Ergebnisteile gebunden:

subj	wert
_:a	"zum"
_:b	"Beispiel"

subj	wert
_:y	"zum"
_:g	"Beispiel"

subj	wert
_:z	"zum"
_:z	"Beispiel"

Datentypen

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:  <http://example.org/> .
ex:bsp1  ex:p  "test" .
ex:bsp2  ex:p  "test"^^xsd:string .
ex:bsp3  ex:p  "test"@de .
ex:bsp4  ex:p  "42"^^xsd:integer .
```

Was liefert eine Anfrage mit folgendem Muster?

```
{    ?subject  <http://example.org/p>  "test" . }
```

Datentypen

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:  <http://example.org/> .
ex:bsp1  ex:p  "test" .
ex:bsp2  ex:p  "test"^^xsd:string .
ex:bsp3  ex:p  "test"@de .
ex:bsp4  ex:p  "42"^^xsd:integer .
```

Was liefert eine Anfrage mit folgendem Muster?

```
{   ?subject  <http://example.org/p>  "test" . }
```

⇝ `ex:bsp1` als einziges Ergebnis
⇝ genaue Übereinstimmung der Datentypen gefordert

Aber: Abkürzung für Zahlenwerte möglich

```
{   ?subject  <http://example.org/p>  42 . }
```

⇝ Datentyp wird aus syntaktischer Form bestimmt
(`xsd:integer`, `xsd:decimal`, oder `xsd:double`)

AIFB

Outline

AIFB

Gruppierende Graph-Muster

Einfache Graph-Muster können durch {...} gruppiert werden.

Beispiel:

```
PREFIX ex:  <http://example.org/>
SELECT ?titel ?autor
WHERE
  { { ?buch    ex:VerlegtBei  <http://springer.com/Verlag>
      ?buch    ex:Titel       ?titel . }
    { }
    ?buch    ex:Autor       ?autor .
  }
```

$\rightsquigarrow$ Sinnvoll erst bei Verwendung zusätzlicher Konstruktoren

Optional Muster

Das Schlüsselwort `OPTIONAL` erlaubt die Angabe optionaler Teile eines Musters.

Beispiel:

```
{ ?buch    ex:VerlegtBei  <http://springer.com/Verlag> .
  OPTIONAL { ?buch    ex:Titel       ?titel . }
  OPTIONAL { ?buch    ex:Autor       ?autor . }
}
```

Optional Muster

Das Schlüsselwort `OPTIONAL` erlaubt die Angabe optionaler Teile eines Musters.

Beispiel:

```
{ ?buch    ex:VerlegtBei  <http://springer.com/Verlag> .
  OPTIONAL { ?buch    ex:Titel       ?titel . }
  OPTIONAL { ?buch    ex:Autor       ?autor . }
}
```

⇝ Teile eines Anfrageergebnisses können ungebunden sein:

buch	titel	autor
http://example.org/buch1	"Titel1"	http://example.org/autor1
http://example.org/buch2	"Titel2"	
http://example.org/buch3	"Titel3"	_:a
http://example.org/buch4		_:a
http://example.org/buch5		

Alternative Muster

Das Schlüsselwort `UNION` erlaubt die Angabe alternativer Teile eines Musters.

Beispiel:

```
{ ?buch    ex:VerlegtBei  <http://springer.com/Verlag> .
  { ?buch    ex:Autor       ?autor . } UNION
  { ?buch    ex:Verfasser   ?autor . }
}
```

$\rightsquigarrow$ Ergebnis entspricht Vereinigung der Ergebnisse mit einer der beiden Bedingungen

Anm.: Gleiche Variablennamen in beiden Teilen von `UNION` beeinflussen sich nicht

Kombination von Optionen und Alternativen (1)

Wie sind Kombinationen von `OPTIONAL` und `UNION` zu verstehen?

```
{ ?buch    ex:VerlegtBei  <http://springer.com/Verlag> .
  { ?buch    ex:Autor       ?autor . } UNION
  { ?buch    ex:Verfasser   ?autor . } OPTIONAL
  { ?autor   ex:Nachname    ?name . }
}
```

- Vereinigung zweier Muster mit angefügtem optionalem Muster
  oder
- Vereinigung zweier Muster, wobei das zweite einen optionalen Teil hat?

Kombination von Optionen und Alternativen (1)

Wie sind Kombinationen von `OPTIONAL` und `UNION` zu verstehen?

```
{ ?buch    ex:VerlegtBei  <http://springer.com/Verlag> .
  { ?buch    ex:Autor       ?autor . } UNION
  { ?buch    ex:Verfasser   ?autor . } OPTIONAL
  { ?autor   ex:Nachname    ?name . }
}
```

- Vereinigung zweier Muster mit angefügtem optionalem Muster
  oder
- Vereinigung zweier Muster, wobei das zweite einen optionalen Teil hat?

⇝ Erste Interpretation korrekt:

```
{ ?buch    ex:VerlegtBei  <http://springer.com/Verlag> .
  { { ?buch    ex:Autor       ?autor . } UNION
    { ?buch    ex:Verfasser   ?autor . }
  } OPTIONAL { ?autor   ex:Nachname    ?name . }
}
```

AIFB

Kombination von Optionen und Alternativen (2)

Allgemeine Regeln

- `OPTIONAL` bezieht sich immer auf genau ein gruppierendes Muster rechts davon.
- `OPTIONAL` und `UNION` sind gleichwertig und beziehen sich auf jeweils alle links davon stehenden Ausdrücke (*linksassoziativ*)

Beispiel:

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

Kombination von Optionen und Alternativen (2)

Allgemeine Regeln

- `OPTIONAL` bezieht sich immer auf genau ein gruppierendes Muster rechts davon.
- `OPTIONAL` und `UNION` sind gleichwertig und beziehen sich auf jeweils alle links davon stehenden Ausdrücke (*linksassoziativ*)

Beispiel:

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

bedeutet

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

AIFB

Outline

AIFB

Wozu Filter?

Viele Anfragen sind auch mit komplexen Graph-Mustern nicht möglich:

- „Welche Personen sind zwischen 18 und 23 Jahre alt?“
- „Der Nachname welcher Personen enthält einen Bindestrich?“
- „Welche Texte in deutscher Sprache sind in der Ontologie angegeben?“

⇝ Filter als allgemeiner Mechanismus für solche Ausdrucksmittel

Filter in SPARQL

Beispiel:

```
PREFIX ex: <http://example.org/>
SELECT ?buch WHERE
  { ?buch ex:VerlegtBei <http://springer.com/Verlag> .
    ?buch ex:Preis      ?preis
    FILTER (?preis < 35)
  }
```

- Schlüsselwort `FILTER`, gefolgt von Filterausdruck in Klammern
- Filterbedingungen liefern Wahrheitswerte (und ev. auch Fehler)
- Viele Filterfunktionen nicht durch RDF spezifiziert
  $\rightsquigarrow$ Funktionen teils aus XQuery/XPath-Standard für XML übernommen

AIFB

Filterfunktionen: Vergleiche und Arithmetik

Vergleichoperatoren: `<`, `=`, `>`, `<=`, `>=`, `!=`

- Vergleich von Datenliteralen gemäß der jeweils *natürlichen* Reihenfolge
- Unterstützung für numerische Datentypen, `xsd:dateTime`, `xsd:string` (alphabetische Ordnung), `xsd:Boolean` (1 > 0)
- für andere Typen und sonstige RDF-Elemente nur `=` und `!=` verfügbar
- kein Vergleich von Literalen inkompatibler Typen (z.B. `xsd:string` und `xsd:integer`)

Filterfunktionen: Vergleiche und Arithmetik

Vergleichoperatoren: `<`, `=`, `>`, `<=`, `>=`, `!=`

- Vergleich von Datenliteralen gemäß der jeweils *natürlichen* Reihenfolge
- Unterstützung für numerische Datentypen, `xsd:dateTime`, `xsd:string` (alphabetische Ordnung), `xsd:Boolean` (1 > 0)
- für andere Typen und sonstige RDF-Elemente nur = und `!=` verfügbar
- kein Vergleich von Literalen inkompatibler Typen (z.B. `xsd:string` und `xsd:integer`)

Arithmetische Operatoren: `+`, `-`, `*`, `/`

- Unterstützung für numerische Datentypen
- Verwendung zur Kombination von Werten in Filterbedingungen

Bsp.: `FILTER( ?gewicht/(?groesse * ?groesse) >= 25 )`

AIFB

Filterfunktionen: Spezialfunktionen für RDF (1)

SPARQL unterstützt auch RDF-spezifische Filterfunktionen:

`BOUND(A)`	`true` falls `A` eine gebundene Variable ist
`isURI(A)`	`true` falls `A` eine URI ist
`isBLANK(A)`	`true` falls `A` ein leerer Knoten ist
`isLITERAL(A)`	`true` falls `A` ein RDF-Literal ist
`STR(A)`	lexikalische Darstellung (`xsd:string`) von RDF-Literalen oder URIs
`LANG(A)`	Sprachcode eines RDF-Literals (`xsd:string`) oder leerer String falls kein Sprachcode
`DATATYPE(A)`	Datentyp-URI eines RDF-Literals (`xsd:string` bei ungetypten Literalen ohne Sprachangabe)

AIFB

Filterfunktionen: Spezialfunktionen für RDF (2)

Weitere RDF-spezifische Filterfunktionen:

`sameTERM(A,B)`	`true`, falls `A` und `B` dieselben RDF-Terme sind.
`langMATCHES(A,B)`	`true`, falls die Sprachangabe `A` auf das Muster `B` passt
`REGEX(A,B)`	`true`, falls in der Zeichenkette `A` der reguläre Ausdruck `B` gefunden werden kann

Beispiel:

```
PREFIX ex: <http://example.org/>
SELECT ?buch WHERE
  { ?buch    ex:Rezension  ?text .
    FILTER ( langMATCHES( LANG(?text), "de") )
  }
```

Filterfunktionen: Boolesche Operatoren

Filterbedingungen können mit Booleschen Operatoren verknüpft werden: &&, ||, !

Teilweise auch durch Graph-Muster ausdrückbar:

- Konjunktion entspricht Angaben mehrerer Filter
- Disjunktion entspricht Anwendung von Filtern in alternativen Mustern

Outline

AIFB

Ausgabeformatierung mit SELECT

Bisher waren alle Ergebnisse Tabellen: Ausgabeformat `SELECT`

Syntax: `SELECT <Variablenliste>` oder `SELECT *`

Vorteil

Einfache sequentielle Abarbeitung von Ergebnissen

Nachteil

Struktur/Beziehungen der Objekte im Ergebnis nicht offensichtlich

AIFB

Ausgabeformatierung mit CONSTRUCT

Kodierung von Ergebnissen in RDF-Graphen: Ausgabeformat `CONSTRUCT`

Syntax: `CONSTRUCT <RDF-Schablone in Turtle>`

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person  ex:mailbox  ?email .
            ?person  ex:telefon  ?telefon . }
WHERE { ?person  ex:email   ?email .
        ?person  ex:tel     ?telefon . }
```

Ausgabeformatierung mit CONSTRUCT

Kodierung von Ergebnissen in RDF-Graphen: Ausgabeformat `CONSTRUCT`

Syntax: `CONSTRUCT <RDF-Schablone in Turtle>`

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person  ex:mailbox  ?email .
            ?person  ex:telefon  ?telefon . }
WHERE { ?person  ex:email   ?email .
        ?person  ex:tel     ?telefon . }
```

Vorteil

Stukturiertes Ergebnis mit Beziehungen zwischen Elementen

Nachteile

- Sequentielle Abarbeitung von Ergebnissen erschwert
- Keine Behandlung von ungebundenen Variablen

Weitere Formate: ASK und DESCRIBE

SPARQL unterstützt zwei weitere Ausgabeformate:

- `ASK` prüft nur, ob es Ergebnisse gibt, keine Parameter
- `DESCRIBE` (informativ) liefert zu jeder gefundenen URI eine RDF-Beschreibung (anwendungsabhängig)

Outline

AIFB

Wozu Modifikatoren?

Bisher nur grundsätzliche Formatierungseinstellungen für Ergebnisse:

- Wie kann man definierte Teile der Ergebnismenge abfragen?
- Wie werden Ergebnisse geordnet?
- Können wiederholte Ergebniszeilen sofort entfernt werden?

$\rightsquigarrow$ Modifikatoren der Lösungssequenz (*solution sequence modifiers*)

Ergebnisse sortieren

Sortierung von Ergebnissen mit Schlüsselwort `ORDER BY`

```
SELECT ?buch, ?preis
WHERE { ?buch  <http://example.org/Preis>  ?preis . }
ORDER BY ?preis
```

Ergebnisse sortieren

Sortierung von Ergebnissen mit Schlüsselwort `ORDER BY`

```
SELECT ?buch, ?preis
WHERE { ?buch  <http://example.org/Preis>  ?preis . }
ORDER BY ?preis
```

- Sortierung wie bei Filter-Vergleichoperatoren,
- Sortierung von URIs alphabetisch als Zeichenketten
- Reihenfolge zwischen unterschiedlichen Arten von Elementen:
  Ungebundene Variable < leere Knoten < URIs < RDF-Literale
- nicht jede Möglichkeit durch Spezifikation definitert

Ergebnisse sortieren

Sortierung von Ergebnissen mit Schlüsselwort `ORDER BY`

```
SELECT ?buch, ?preis
WHERE { ?buch  <http://example.org/Preis>  ?preis . }
ORDER BY ?preis
```

- Sortierung wie bei Filter-Vergleichoperatoren,
- Sortierung von URIs alphabetisch als Zeichenketten
- Reihenfolge zwischen unterschiedlichen Arten von Elementen: Ungebundene Variable < leere Knoten < URIs < RDF-Literale
- nicht jede Möglichkeit durch Spezifikation definitert

Weitere mögliche Angaben:

- `ORDER BY DESC(?preis)`: absteigend
- `ORDER BY ASC(?preis)`: aufsteigend, Voreinstellung
- `ORDER BY DESC(?preis), ?titel`: hierarchische Ordnungskriterien

AIFB

LIMIT, OFFSET und DISTINCT

Einschränkung der Ergebnismenge:

- `LIMIT`: maximale Anzahl von Ergebnissen (Tabellenzeilen)
- `OFFSET`: Position des ersten gelieferten Ergebnisses
- `SELECT DISTINCT`: Entfernung von doppelten Tabellenzeilen

```
SELECT DISTINCT ?buch, ?preis
WHERE { ?buch  <http://example.org/Preis>  ?preis . }
ORDER BY ?preis  LIMIT 5  OFFSET 25
```

⇝ `LIMIT` und `OFFSET` nur mit `ORDER BY` sinnvoll!

Kombination von Modifikatoren

Reihenfolge bei Abarbeitung von Modifikatoren:

1. Sortierung gemäß `ORDER BY`
2. Entfernung der nicht ausgewählten Variablen
3. Entfernung doppelter Ergebnisse (`DISTINCT`)
4. Enfernung der ersten `OFFSET` Ergebnisse
5. Entfernung aller Ergebnisse bis auf `LIMIT`

⇝ Sortierung auch nach nicht ausgewählten Variablen möglich

⇝ `ORDER BY` nicht nur für `SELECT` relevant

Outline

AIFB

Vorgestellte SPARQL-Merkmale im Überblick

Grundstruktur
PREFIX
WHERE

Ausgabeformate
SELECT
CONSTRUCT
ASK
DESCRIBE

Graph-Muster
Einfache Graph-Muster
{...}
OPTIONAL
UNION

Filter
BOUND
isURI
isBLANK
isLITERAL
STR
LANG
DATATYPE
sameTERM
langMATCHES
REGEX

Modifikatoren
ORDER BY
LIMIT
OFFSET
DISTINCT

AIFB

Ausblick

Offene Fragen

- Wie genau ist die Semantik von SPARQL definiert?
- Wie schwer ist die vollständige Umsetzung von SPARQL? Implementationen?
- Wie kann man Anfragen an RDF Schema oder OWL stellen?

$\rightsquigarrow$ Vorlesung 10 und Vorlesung 11. . .

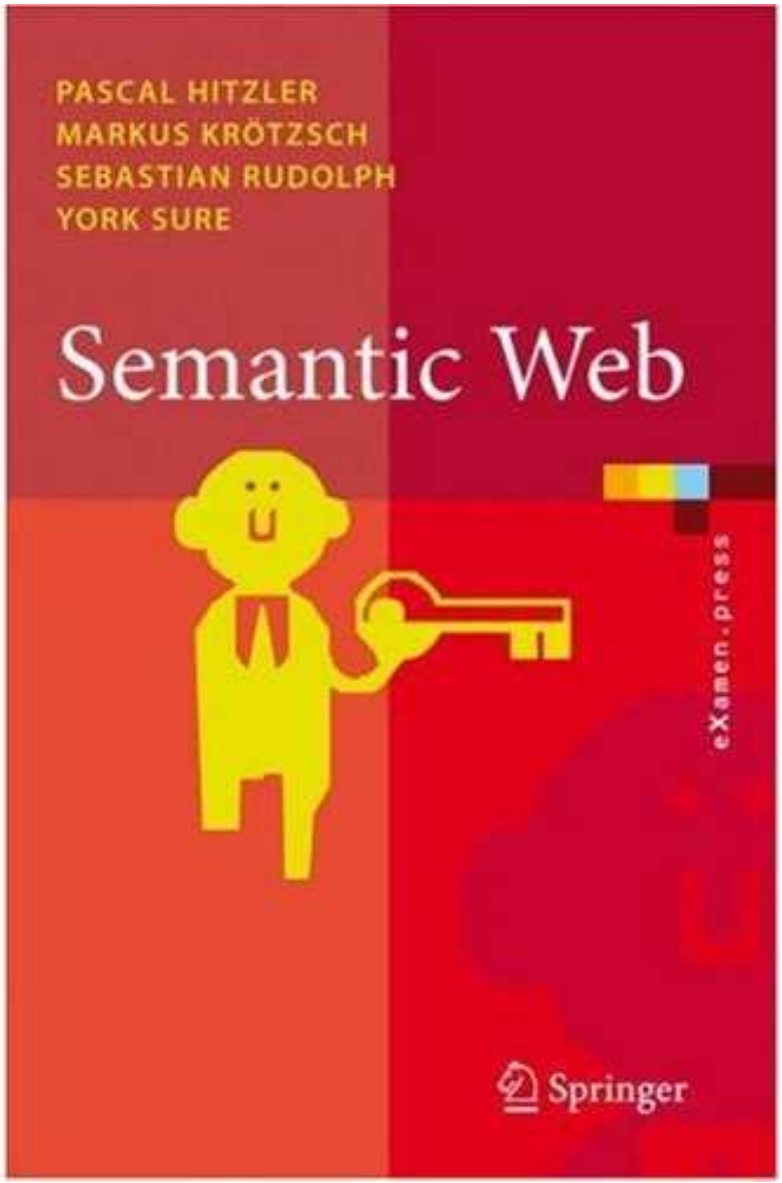
AIFB

Literatur

Pascal Hitzler
Markus Krötzsch
Sebastian Rudolph
York Sure

Semantic Web
Grundlagen

Springer 2008, 277 S., Softcover
ISBN: 978-3-540-33993-9
Aktuelle Literaturhinweise online



AIFB