Programming the Semantic Web

Steffen Staab & Team@WeST
This presentation contains program code, engineering, speculation and worse. It may be viewed as offensive by some viewers.
Semantic Web: Great History, Still Long Way to Go!

Big Systems

Practice
Semantic Web: Great History, Still Long Way to Go!

Big Systems

Practice

Research influence
Wanted: mainstream adoption!

Big Systems

Practice

Research influence
WeST

Steffen Staab
staab@uni-koblenz.de

Semantic Web: Great History, Still Long Way to Go!

Target: Programming

Wanted: mainstream adoption!

Big Systems

Practice

Research influence
Target: Programming

Wanted: mainstream adoption!

Big Systems

Practice

Research influence
SEMANTIC WEB PROGRAMMING: BIRD’S EYES VIEW
Semantic Web Programming

Linked Data

RDF file

Ontology

SPARQL endpoint

Steffen Staab
staab@uni-koblenz.de
Hypothesis:
Semantic Web data is wonderful, but programming with Semantic Web has changed too little since 2000 and still is a mess.
Hypothesis:
Semantic Web data is wonderful, but programming with Semantic Web has changed too little since 2000 and still is a mess.

We promise flexibility, but code still hard to maintain.
Semantic Web Programming

Linked Data

RDF file

Ontology

SPARQL endpoint

Data Mgmt

Eclipse

Visual Studio
Semantic Web Programming

Linked Data

 Ontology

RDF file

„Inside“ Data Mgmt

SPARQL endpoint

Eclipse

Visual Studio
Semantic Web Impedance Mismatch

„Inside“
✓ http get
✓ Dereferencing
✓ Query execution
✓ Updating
✓ Indexing
✓ Triplification
✓ Integration
✓ Streaming

„Inside“
Data Mgmt

Eclipse

Visual Studio

...
Semantic Web Impedance Mismatch

„Inside“
✓ http get
✓ Dereferencing
✓ Query execution
✓ Updating
✓ Indexing
✓ Triplification
✓ Integration
✓ Streaming

Mind the Gap!

Eclipse

Visual Studio
Semantic Web

Linked Data

RDF file

"Inside" Data Mgmt

"Outside" Data Mgmt

Ontology

SPARQL endpoint

Eclipse

Visual Studio
Semantic Web

„Outside“

„Inside“ Data Mgmt

„Outside“ Data Mgmt

Eclipse

Visual Studio

...
Understanding by the developer

Semantic Web

„Outside“

„Inside“ Data Mgmt

„Outside“ Data Mgmt

Eclipse

Visual Studio

....

"Outside"
"Outside"

Understanding by the developer

"Search + Code"

Eclipse

Visual Studio
“Outside“

- Understanding by the developer
  - „Search + Code“
  - „Browse + Code“

- ....

- Eclipse

- Visual Studio
“Outside“
- Understanding by the developer
  - “Search + Code“
  - “Browse + Code“
- Triple-object mapping

“Inside“ Data Mgmt  "Outside“ Data Mgmt

Eclipse

Visual Studio
“Outside“

- Understanding by the developer
  - “Search + Code“
  - “Browse + Code“
- Triple-object mapping
- Code generation

- “Inside“
  - Data Mgmt

- “Outside“
  - Data Mgmt

- Eclipse
  - ....

- Visual Studio
“Outside“
10 Understanding by the developer
 10 „Search + Code“
 10 „Browse + Code“
10 Triple-object mapping
10 Code generation
10 Process of federation

„Inside“ Data Mgmt

„Outside“ Data Mgmt

Eclipse

Visual Studio
"Outside"
1. Understanding by the developer
   a. "Search + Code"
   b. "Browse + Code"
2. Triple-object mapping
3. Code generation
4. Process of federation
   a. LD vs endpoint
“Outside“

10 Understanding by the developer
   10 “Search + Code“
   10 “Browse + Code“

10 Triple-object mapping

10 Code generation

10 Process of federation
   ▪ LD vs endpoint
   ▪ Models of federation
„Outside“

Understanding by the developer
- „Search + Code“
- „Browse + Code“

Triple-object mapping

Code generation

Process of federation
  - LD vs endpoint
  - Models of federation

Abstraction layers that facilitate a developer’s life
Programming with Data: What does it cost?

\[ C_{\text{total}} = t \cdot C_{\text{tool}} \]

\[ C_{\text{tool}} : \text{ Costs for building } t \text{ many } \textbf{tools}, \text{ shared; almost free} \]
Programming with Data: What does it cost?

\[ C_{\text{total}} = t*C_{\text{tool}} + d*t*C_{\text{learn}}. \]

\( C_{\text{tool}} \): Costs for building \( t \) many tools, shared; almost free

\( C_{\text{learn}} \): Costs for learning how to use technology per developer \( d \)
\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}}. \]

- **\( C_{\text{tool}} \):** Costs for building \( t \) many **tools**, shared; almost free
- **\( C_{\text{learn}} \):** Costs for **learning** how to use technology per developer \( d \)
- **\( C_{\text{deu}} \):** Costs for data engineering/understanding \( s \) sources
Programming with Data: What does it cost?

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} \]

- \( C_{\text{tool}} \): Costs for building \( t \) many \textbf{tools}, shared; almost free
- \( C_{\text{learn}} \): Costs for \textbf{learning} how to use technology per developer \( d \)
- \( C_{\text{deu}} \): Costs for data engineering/understanding \( s \) sources
- \( C_{\text{map}} \): Costs for \textbf{mapping} data structure for \( s \) sources to objects
Programming with Data: What does it cost?

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]

- **\( C_{\text{tool}} \)**: Costs for building \( t \) many **tools**, shared; almost free
- **\( C_{\text{learn}} \)**: Costs for **learning** how to use technology per developer \( d \)
- **\( C_{\text{deu}} \)**: Costs for data engineering/understanding \( s \) sources
- **\( C_{\text{map}} \)**: Costs for **mapping** data structure for \( s \) sources to objects
- **\( C_{\text{code}} \)**: Actual costs for **accessing/manipulating** data \( n \) times
Programming with Data: What does it cost?

\[ C_{\text{total}} = t*C_{\text{tool}} + d*t*C_{\text{learn}} + s*C_{\text{deu}} + s*C_{\text{map}} + n*C_{\text{code}} \]

- **Costs for building** \( t \) many **tools**, shared; almost free
- **Costs for learning** how to use technology per developer \( d \)
- **Costs for data engineering/understanding** \( s \) sources
- **Costs for mapping** data structure for \( s \) sources to objects
- **Actual costs for accessing/manipulating** data \( n \) times
$C_{\text{total}} = t \times C_{\text{tool}} + d \times t \times C_{\text{learn}} + s \times C_{\text{deu}} + s \times C_{\text{map}} + n \times C_{\text{code}}$
\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]
SWOT of Semantic Web Programming

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]
\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]
SWOT of Semantic Web Programming

\[ C_{total} = t \cdot C_{tool} + d \cdot t \cdot C_{learn} + s \cdot C_{deu} + s \cdot C_{map} + n \cdot C_{code} \]

threat

as good as RelDB is not good enough!
SWOT of Semantic Web Programming

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]

- **Threat**: as good as RelDB is not good enough!
SWOT of Semantic Web Programming

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]

- **threat**
  - as good as RelDB is not good enough!

- **strength/weakness**
  - Strong in flexibility
  - Somewhat weak in performance

Steffen Staab
staab@uni-koblenz.de
C_{total} = t*C_{tool} + d*t*C_{learn} + s*C_{deu} + s*C_{map} + n*C_{code}

- **threat**
  - as good as RelDB is not good enough!

- **strength/weakness**
  - Strong in flexibility
  - Somewhat weak in performance

- **opportunity**
  - not good enough!
\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]

- **Threat**: as good as RelDB is not good enough!
- **Strength/Weakness**: Strong in flexibility. Somewhat weak in performance.
- **Opportunity**: Not a strength, yet!
- **Opportunity**: „Outside“
Target cost scenario 1: Automated setup

\[ C_{\text{total}} \]

\[ n \quad (\text{as in } n \times C_{\text{code}}) \]

- ReIDB/XML coding efforts
- SemWeb coding efforts
Target cost scenario 1: Automated setup

\( C_{\text{total}} \)

Setup cost

0

\( n \) (as in \( n \cdot C_{\text{code}} \))

RelDB/XML coding efforts

SemWeb coding efforts
Target cost scenario 1: Automated setup

\[ C_{\text{total}} \]

\[ n \text{ (as in } n \cdot C_{\text{code}} ) \]

SemWeb coding efforts

RelDB/XML coding efforts

**Setup cost**

**Diff. costs for learning**
Target cost scenario 1: Automated setup

$C_{\text{total}}$ (as in $n \cdot C_{\text{code}}$)

SemWeb coding efforts

RelDB/XML coding efforts

Applies to small $n$

$n$ (as in $n \cdot C_{\text{code}}$)
Target cost scenario 2: Manually perfected setup

\[ C_{\text{total}} \]

\[ n \text{ (as in } n*C_{\text{code}}) \]

- RelDB/XML coding efforts
- SemWeb coding efforts

„Perfect“ object model shields developer from database idiosyncracies
Target cost scenario 2: Manually perfected setup

Setup costs

Setup costs

C_{\text{total}}

n (as in n*C_{\text{code}})

ReLDB/XML coding efforts

SemWeb coding efforts

“Perfect“ object model shields developer from database ideosyncracies
Intermediate conclusion

Minimize costs for setup:

\( C_{\text{learn}} \): Costs for \textbf{learning} how to use technology per developer \( d \)

\( C_{\text{de}} \): Costs for data engineering/understanding \( s \) sources

\( C_{\text{map}} \): Costs for \textbf{mapping} data structure for \( s \) sources to objects

Minimize costs for core programming:

\( C_{\text{code}} \): Actual costs for \textbf{accessing/manipulating} data \( n \) times
Intermediate conclusion

Minimize costs for setup:

\[ C_{\text{learn}} : \text{Costs for learning how to use technology per developer} \]

\[ C_{\text{de}} : \text{Costs for data engineering/understanding sources} \]

\[ C_{\text{map}} : \text{Costs for mapping data structure for sources to objects} \]

Minimize costs for core programming:

\[ C_{\text{code}} : \text{Actual costs for accessing/manipulating data n times} \]

Costs for learning and understanding constitute a threat!
Need to be overcome!
SEMANTIC WEB PROGRAMMING: IDENTIFYING SOME PITFALLS
Example scenario: Jamendo

Data about license free music
- ~ 1 Million triples
- classes and predicates from 18 different ontologies
  - FOAF, Tag ontology, music ontology, ...

Simple programming task:
- List for every music artist, all the records they made
Creation of initial data model ➔ data model design
Software Development Process Overview

1. Creation of initial data model
2. Exploration of the data source
3. Revised data model design
4. Data model design
5. Creation of initial data model
Software Development Process Overview

1. Creation of initial data model
2. Data model design
3. Exploration of the data source
4. Revised data model design
5. Creation of model in code
6. Data model prototype
Software Development Process Overview

1. Creation of initial data model
2. Exploration of the data source
3. Creation of model in code
4. Data model design
5. Revised data model design
6. Data model prototype
7. Query design / implementation
8. Data queries
Software Development Process Overview

1. Creation of initial data model
2. Exploration of the data source
3. Creation of model in code
4. Revised data model design
5. Query design / implementation
6. Data model prototype
7. Mapping of query results
8. Final data model
// Selecting all artists from database
String queryString = "SELECT ?artist " +
    " WHERE { " +
    " ?artist rdf:type mo:MusicArtist ." +
    " } "

// Executing query and retrieving results
Query query = QueryFactory.create(queryString);
QueryExecution qexec = QueryExecutionFactory
    .sparqlService(repoUrl, query);
ResultSet results = qexec.execSelect();
while (results.hasNext()) {
    QuerySolution soln = results.nextSolution();
    printSongs(soln.getResource("?artist"));
}
public void printSongs(Resource artist) {
    StmtIterator iter = artist.listProperties(MO.MADE);
    while (iter.hasNext()) {
        System.out.println(iter.nextStatement()
            .getObject());
    }
}
public void printSongs(Resource artist) {
    StmtIterator iter = artist.listProperties(MO.MADE);
    while (iter.hasNext()) {
        System.out.println(iter.nextStatement().getObject());
    }
}

Observations

- SPARQL queries are strings
- Results are strings
From artists to songs

```java
public void printSongs(Resource artist) {
    StmtIterator iter = artist.listProperties(MO.MADE);
    while (iter.hasNext()) {
        System.out.println(iter.nextStatement().getObject());
    }
}
```

Observations
- SPARQL queries are strings
- Results are strings
- RDF Typing is lost
From artists to songs

public void printSongs(Resource artist) {
    StmtIterator iter = artist.listProperties(MO.MADE);
    while (iter.hasNext()) {
        System.out.println(iter.nextStatement().
                           getObject());
    }
}

Observations

- SPARQL queries are strings
- Results are strings
- Requires good understanding of the data source

RDF Typing is lost
Related Work on RDF Access

Code generation

- Sommer
- Winter
- OntoMDE
Related Work on RDF Access

Code generation
- Sommer
- Winter
- OntoMDE

Dynamic Typing
- E.g. ActiveRDF (Oren et al 2007)
  - “convention over configuration”
- dynamic metaprogramming allows for slick code
Related Work on RDF Access

Code generation
- Sommer
- Winter
- OntoMDE

Dynamic Typing
- E.g. ActiveRDF (Oren et al 2007))
  - “convention over configuration”
- dynamic 
  metaprogramming allows for slick code

Static Typing
- Errors detected before execution
  - Misspelling discovered by compiler!
  - Anecdote: 2nd place because of misspelt var.

Criticism
Related Work on RDF Access

**Code generation**
- Sommer
- Winter
- OntoMDE

**Dynamic Typing**
- E.g. ActiveRDF (Oren et al 2007))
  - “convention over configuration”
- dynamic metaprogramming allows for slick code

**Static Typing**
- Errors detected before execution
  - Misspelling discovered by compiler!
  - Anectode: 2nd place because of misspelt var.
- Static types are form of documentation
  - Less knowledge about data source required
Related Work on RDF Access

Code generation
- Sommer
- Winter
- OntoMDE

Dynamic Typing
- E.g. ActiveRDF (Oren et al 2007))
  - “convention over configuration”
- dynamic metaprogramming allows for slick code

Static Typing
- Errors detected before execution
  - Misspelling discovered by compiler!
  - Anectode: 2nd place because of misspelt var.
- Static types are form of documentation
  - Less knowledge about data source required
- Better IDE integration / autocompletion

Criticism
SEMANTIC WEB PROGRAMMING:
LITEQ – OUR OUTSIDE APPROACH
Programming against Jamendo
Exploration of classes

```activecode
Store.NPQL().\``foaf:Agent``
```
```
Exploration of classes

Exploration of relations

Store.NPQL()``mo:MusicArtist``.``-``>``.mo:m

Eigenschaft Temporary classes.http://purl.org/ontology/mo/
MusicArtistPropNav.mo:made: RDFStore.RDFStore<...>.mo:madeProperty

Indicates that a record was made by this artist.
Node Path Query Language: Query Formulation

Exploration of classes
Exploration of relations
Querying for instances

17 let allArtists = Store.NPQL()``mo:MusicArtist``.Extension
Node Path Query Language: Query Formulation

Exploration of classes
Exploration of relations
Querying for instances

17 let allArtists = Store.NPQL()."\`mo:MusicArtist\`".Extension
Exploration of classes
Exploration of relations
Querying for instances

17 let allArtists = Store.NPQL().`mo:MusicArtist``.Extension

Type
set of mo:MusicArtist
Node Path Query Language: Query Formulation

Exploration of classes
Exploration of relations
Querying for instances

17 let allArtists = Store.NPQL().`mo:MusicArtist`.Extension

Type
set of mo:MusicArtist

No definition or declaration needed
Node Path Query Language for Code Development

Exploration of classes
Exploration of relations
Querying for instances
Developing code with queries

17 let allArtists = Store.NPQL().``mo:MusicArtist``.Extension
18 for artist in allArtists do
19  printfn "Artist %A recorded %A" artist.``foaf:name``
   artist.``foaf:made``

Node Path Query Language for Code Development

Exploration of classes
Exploration of relations
Querying for instances
Developing code with queries

One language to bind them all

```
17  let allArtists = Store.NPQL().`mo:MusicArtist```.Extension
18  for artist in allArtists do
19    printfn "Artist %A recorded %A" artist.`foaf:name`
        artist.`foaf:made`
```
Node Path Query Language for Code Development

- Exploration of classes
- Exploration of relations
- Querying for instances
- Developing code with queries

One language to bind them all

```haskell
17 let allArtists = Store.NPQL().`\mo:MusicArtist``.Extension
18 for artist in allArtists do
19  println "Artist %A recorded %A" artist.`\foaf:name`
    artist.`\foaf:made`
Node Path Query Language for Code Development

Exploration of classes
Exploration of relations
Querying for instances
Developing code with queries

One language to bind them all

```
17  let allArtists = Store.NPQL().```mo:MusicArtist```.Extension
18  for artist in allArtists do
19   printfn "Artist %A recorded %A"
       artist.```foaf:name``
       artist.```foaf:made``
```
Node Path Query Language for Code Development

Exploration of classes
Exploration of relations
Querying for instances
Developing code with queries

One language to bind them all

17 let allArtists = Store.NPQL().`~mo:MusicArtist``.Extension
18 for artist in allArtists do
19 printfn "Artist %A recorded %A"
20 artist.`~foaf:name``
artist.`~foaf:made``
Node Path Query Language for Code Development

Exploration of classes
Exploration of relations
Querying for instances
Developing code with queries

All translated into SPARQL queries at
• Development time
• Type inference at compile time (but also as part of IDE)
• Querying again at run time

17  let allArtists = Store.NPQL().`\mo:MusicArtist``.Extension
18  for artist in allArtists do
19   printfn "Artist %A recorded %A"
    artist.``foaf:name``
    artist.``foaf:made``
Exploration of classes
Exploration of relations
Querying for instances
Developing code with queries
Developing code with new classes

11 type MusicArtist = Store.``mo:MusicArtist``.Intension
12 let newArtist = new MusicArtist("http://.../artist/1234")
13 newArtist.``foaf:name`` <- [ "myBandName" ]
Node Path Query Language for Code Development

Exploration of classes
Exploration of relations
Querying for instances
Developing code with queries
Developing code with new classes

All translated into SPARQL queries at
• Development time
• Run time update
• Persistence!

11 type MusicArtist = Store.~`mo:MusicArtist~.Intension
12 let newArtist = new MusicArtist("http://.../artist/1234")
13 newArtist~`foaf:name~ <- [ "myBandName" ]
NPQL (Node Path Query Language)

- **Intensional Queries**
  - Describing RDF classes and properties for reuse in IDE and in host language metaprogramming.
NPQL (Node Path Query Language)

- **Intensional Queries**
  - Describing RDF classes and properties for reuse in IDE and in host language metaprogramming

- **Extensional Queries**
  - Class instances and property instances
**NPQL** (Node Path Query Language)

- **Intensional Queries**
  - Describing RDF classes and properties for reuse in IDE and in host language metaprogramming

- **Extensional Queries**
  - Class instances and property instances

- Compilation to SPARQL for reuse of existing endpoints
NPQL (Node Path Query Language)

- **Intensional Queries**
  → Describing RDF classes and properties
    for reuse in IDE and in host language metaprogramming

- **Extensional Queries**
  → Class instances and property instances

- Compilation to SPARQL for reuse of existing endpoints

Ongoing discussion about details of NPQL
NPQL (Node Path Query Language)
- *Intensional Queries*
- *Extensional Queries*
- Compilation to SPARQL

LITEQ (Language Integrated Types, Extensions and Queries)
NPQL (Node Path Query Language)
- *Intensional Queries*
- *Extensional Queries*
- Compilation to SPARQL

**LITEQ** (Language Integrated Types, Extensions and Queries)
- *Implementation* of NPQL as F# Type Provider in Visual Studio

-
NPQL (Node Path Query Language)
- *Intensional Queries*
- *Extensional Queries*
- Compilation to SPARQL

LITEQ (Language Integrated Types, Extensions and Queries)
- *Implementation* of NPQL as F# Type Provider in Visual Studio
- *Autocompletion* using NPQL queries
NPQL (Node Path Query Language)
- *Intensional Queries*
- *Extensional Queries*
- Compilation to SPARQL

**LITEQ** (Language Integrated Types, Extensions and Queries)
- *Implementation* of NPQL as F# Type Provider in Visual Studio
- *Autocompletion* using NPQL queries
- *Automatic typing*
  of extensional query results
  by intensional queries
Cost savings

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]
Cost savings

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]

\( C_{\text{tool}} \): open source

•

•

•

•
Cost savings

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]

\( C_{\text{tool}} \): open source

\( C_{\text{learn}} \): not free – though autocompletion reduces cognitive load
Cost savings

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]

- \( C_{\text{tool}} \): open source
- \( C_{\text{learn}} \): not free – though autocompletion reduces cognitive load
- \( C_{\text{deu}} \): not free – understanding the RDF schema from your IDE

...
Cost savings

\[ C_{\text{total}} = t \cdot C_{\text{tool}} + d \cdot t \cdot C_{\text{learn}} + s \cdot C_{\text{deu}} + s \cdot C_{\text{map}} + n \cdot C_{\text{code}} \]

- \( C_{\text{tool}} \): open source
- \( C_{\text{learn}} \): not free – though autocompletion reduces cognitive load
- \( C_{\text{deu}} \): not free – understanding the RDF schema from your IDE
- \( C_{\text{map}} \): 0
Cost savings

\[ C_{total} = t*C_{tool} + d*t*C_{learn} + s*C_{deu} + s*C_{map} + n*C_{code} \]

- **C\textsubscript{tool}**: open source
- **C\textsubscript{learn}**: not free – though autocompletion reduces cognitive load
- **C\textsubscript{deu}**: not free – understanding the RDF schema from your IDE
- **C\textsubscript{map}**: 0
- **C\textsubscript{code}**: a lot less than for dotNet RDF (Apache Jena?!!)
  little bit more than for a fictitious perfect object model
### Halstead metrics for different tasks:

<table>
<thead>
<tr>
<th></th>
<th>dotNetRDF</th>
<th>LITEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of distinct operators ($n_1$)</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>No. of distinct operands ($n_2$)</td>
<td>25</td>
<td>16</td>
</tr>
<tr>
<td>Total no. of operators ($N_1$)</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>Total no. of operands ($N_2$)</td>
<td>34</td>
<td>22</td>
</tr>
<tr>
<td>Program Vocabulary</td>
<td>34</td>
<td>23</td>
</tr>
<tr>
<td>Program length</td>
<td>53</td>
<td>40</td>
</tr>
<tr>
<td>Program Volume</td>
<td>269,64</td>
<td>180,94</td>
</tr>
<tr>
<td>Difficulty</td>
<td>6,12</td>
<td>4,81</td>
</tr>
<tr>
<td>Effort</td>
<td>1650,17</td>
<td>870,79</td>
</tr>
<tr>
<td>Time needed</td>
<td>92s</td>
<td>49s</td>
</tr>
</tbody>
</table>

Conventional Semantic Web programming approaches waste up to 50% of your efforts!
Speculation 1

C_{total}

Diff. costs for learning

n (as in n*C_{code})

ReIDB/XML coding efforts

SemWeb coding efforts

Applies to small n
Speculation 1: Using ontologies and RDF schemata, we can develop more efficiently using the right tools!

- **SemWeb coding efforts**
- **Applies to small $n$**

$n$ (as in $n \times C_{\text{code}}$)

$C_{\text{total}}$

Diff. costs for learning
If someone gives me a perfect RDF-to-OO mapping for free then I will not care about whether it is RDF or RelDB underneath!
Speculation 2

$C_{\text{total}}$

Diff. costs for setup

$n$ (as in $n \times C_{\text{code}}$)

"Perfect" object model shields developer from database idiosyncrasies

ReIDB/XML coding efforts

SemWeb coding efforts

Steffen Staab
staab@uni-koblenz.de

WeST
Speculation 2: For large programmes, our tools need to offer better support to reduce setup costs!
Issues

- No schema
  - many LOD sources do not have schema
Issues

- No schema
  - many LOD sources do not have schema
- Useless schema
Issues

- No schema
  - many LOD sources do not have schema
- Useless schema
Issues

- No schema
  - many LOD sources do not have schema
- Useless schema
Issues

- No schema
  - many LOD sources do not have schema
- Useless schema

New kinds of (?) schema induction!
Issues

- No schema
  - many LOD sources do not have schema
- Useless schema
- Property-based schemata
  - Cf. Homoceanu et al ISWC 2013
    - ask for all movies – not possible by asking for class movie, but only for different property combinations
Integration of schema induction
Outlook wrt LITEQ

- Integration of schema induction
- Programming the Semantic Web as a whole
  - Federated queries
  - Link traversal-based queries
Outlook wrt LITEQ

- Integration of schema induction
- Programming the Semantic Web as a whole
  - Federated queries
  - Link traversal-based queries
- More expressive query language
  - Composed data types in tractable DL
Outlook wrt LITEQ

- Integration of schema induction
- Programming the Semantic Web as a whole
  - Federated queries
  - Link traversal-based queries
- More expressive query language
  - Composed data types in tractable DL
- More precise integrated type inference
  - (composed) types from RDF data source
  - Type inference in host language
SCHEMEX: (PART OF) OUR „INSIDE“ APPROACH
SELECT ?x
WHERE {
  ?y rdf:type fb:Computer_Scientist
}
SELECT ?x
WHERE {
  ?y rdf:type fb:Computer_Scientist .
}
SELECT ?x
WHERE {
  ?y rdf:type fb:Computer_Scientist
}
SELECT ?x
WHERE {
  ?y rdf:type fb:Computer_Scientist
}
Schema-basierten Zugriff auf die LOD Cloud

SELECT ?x
WHERE {
  ?y rdf:type fb:Computer_Scientist
}
Schema-basierten Zugriff auf die LOD Cloud

SELECT ?x
WHERE {
  ?y rdf:type fb:Computer_Scientist
}
Index models (in general)

Data items / Payload

D
Index models (in general)

Keys

\[ k_1, k_2, k_3, \ldots, k_n \]

Data items / Payload

\[ K, D \]
Index models (in general)

- $D$: Daten, which are to be indexed (payload)
- $K$: Keys, used for indexing data (index elements)
- $\sigma$: selection function: How is key used for data selection?
Index models (in general)

- $D$: Daten, which are to be indexed (payload)
- $K$: Keys, used for indexing data (index elements)
- $\sigma$: selection function: How is key used for data selection?
(No) Schema?
Schema information from LOD Cloud

Guidelines/ Best Practices

(No)
Schema?
Schema information from LOD Cloud

Guidelines/ Best Practices

(No) Schema?

Social effects
Schema information from LOD Cloud

Guidelines/ Best Practices

(No) Schema?

Automatic tools

Social effects
Schema information from LOD Cloud

Guidelines/ Best Practices

Induced from observations

Emergent schema!

Automatic tools

Social effects
Simple examples for derived schema information

Type Set

\( \text{CA} \)

\( \text{y} \)

\( \text{CB} \)

\text{rdf:type}
Simple examples for derived schema information

Type Set

\{c_A, c_B\} \rightarrow y, ...
Simple examples for derived schema information

Type Set

\[
\{c_A, c_B\} \rightarrow y, \ldots
\]

Property Set

\[
x, \ p_1, \ p_2, \ p_3
\]
Simple examples for derived schema information

Type Set

{c_A, c_B} → y, ...

Property Set

{p_1, p_2, p_3} → x, ...

\(\text{rdf:type}\)

\(\text{c}_A\) → \(\text{y}\) → \(\text{c}_B\)

\(\text{p}_1\) → \(\text{x}\) → \(\text{p}_2\) → \(\text{c}_B\) → \(\text{p}_3\) → \(\text{c}_A\)
New type of indices: Property Set Index

- $s_1 \ p_1 \ o_1 \ c_1$
- $s_1 \ p_2 \ o_1 \ c_1$
- $s_2 \ p_2 \ o_2 \ c_1$
- $s_3 \ p_1 \ o_3 \ c_1$
- $s_3 \ p_2 \ o_4 \ c_1$
- $s_4 \ p_3 \ o_1 \ c_1$
New type of indices: Property Set Index

Aggregate

s1 o1 p1 c1
s1 o1 p2 c1
s2 o2 p2 c1
s3 o3 p1 c1
s3 o4 p2 c1
s4 o1 p3 c1
New type of indices: Property Set Index

Aggregate

\[ \begin{align*}
  s_1 & \quad p_1 & \quad o_1 & \quad c_1 \\
  s_1 & \quad p_2 & \quad o_1 & \quad c_1 \\
  s_2 & \quad p_2 & \quad o_2 & \quad c_1 \\
  s_3 & \quad p_1 & \quad o_3 & \quad c_1 \\
  s_3 & \quad p_2 & \quad o_4 & \quad c_1 \\
  s_4 & \quad p_3 & \quad o_1 & \quad c_1
\end{align*} \]
New type of indices: Property Set Index
New type of indices: Property Set Index

Aggregate

Invert

Aggregate:

- $s_1$, $p_1$, $o_1$, $c_1$
- $s_1$, $p_2$, $o_1$, $c_1$
- $s_2$, $p_2$, $o_2$, $c_1$
- $s_3$, $p_1$, $o_3$, $c_1$
- $s_3$, $p_2$, $o_4$, $c_1$
- $s_4$, $p_3$, $o_1$, $c_1$

Invert:

- $p_1$, $p_2$
- $p_1$, $p_2$
- $p_1$, $p_2$
- $p_3$
- $p_2$
- $p_3$

Steffen Staab
staab@uni-koblenz.de
New type of indices: Property Set Index

Aggregate

Invert

Problem:
Computational effort!
Core idea for scalable indexing:
Data local computation of property sets and type sets in main memory has only little loss of precision/recall.
Quality of approximating index [JoWebSem12]

![Quality of approximating index chart](chart.png)
CONCLUSION
Semantic Web: Make Developers More Productive

Linked Data

Ontology

RDF file

SPARQL endpoint

"Inside" Data Mgmt

"Outside" Data Mgmt

Eclipse

Visual Studio
Thank You!

Semantic Web

Social Web & Web Retrieval

Interactive Web & Human Computing

Software & Services

Web & Economy

Computational Social Science
References


- C. Saathoff, S. Scheglmann, S. Schenk. Winter: Mapping RDF to POJOs revisited.


Data Characteristics

(i) provided by different people in an ad-hoc manner,
(ii) distributed,
(iii) semi-structured, self-describing,
(iv) (more or less) typed,
(v) supposed to be used serendipitously.

Tasks

(i) Select
(ii) Fetch
(iii) Map
(iv) Code

Tasks

(i) Explore schema
(ii) Code type creation
(iii) Query formulation
(iv) Data access + manip
(i) Explore schema
(ii) Create code type
(iii) Map data into code types
(iv) Select data
  - Formulate query
(v) Fetch data
(vi) Code with data

SPARQL based
(i) tbd
(ii) tbd
(iii) + (iv) by formulating query and mapping result string into objects
(v) Executing query to fetch data
(vi) Code with data
(i) Explore schema
(ii) Create code type
(iii) Map data into code types
(iv) Select data
   - Formulate query
(v) Fetch data
(vi) Code with data

LITEQ based
(i) (ii) (iii) (iv) done together
(v) Execute query to fetch data
(vi) Code with data
Thank You!

Semantic Web
Social Web & Web Retrieval
Interactive Web & Human Computing
Software & Services
Web & Economy
Computational Social Science

Steffen Staab
staab@uni-koblenz.de
150