SPARQL++ for Mapping between RDF Vocabularies

Axel Polleres (DERI Galway)

Joint work with:
F. Scharffe (LFU Innsbruck), R. Schindlauer (Univ Calabria/TU Vienna)

ODBASE 2007 - November 27, 2007

Outline

Motivation

- Mapping by SPARQL

Examples

Implementation

- HEX-Programs
- Demo
- RDFS

Wrap-up

Motivation – Ontology Alignment/Mapping

- Typically: Description of correspondences and overlaps between ontological entities (properties, classes, individuals, etc.)
- W3C standards for writing ontologies in place (RDFS, OWL), but limited expressivity for describing mappings.
- Which language to use?
- How to **publish** mappings/alignments? This is important to make *Open Linked Data*[1] happen!

[1] Combining RDF data that is "out there", e.g. Sindice, DBPedia, SWRipes etc.

Motivation – Ontology Alignment/Mapping

- Typically: Description of correspondences and overlaps between ontological entities (properties, classes, individuals, etc.)
- W3C standards for writing ontologies in place (RDFS, OWL), but limited expressivity for describing mappings.
- Which language to use?
- How to **publish** mappings/alignments? This is important to make *Open Linked Data*[1] happen!

[1] Combining RDF data that is "out there", e.g. Sindice, DBPedia, SWPipes etc.

Motivation – Ontology Alignment/Mapping

- Typically: Description of correspondences and overlaps between ontological entities (properties, classes, individuals, etc.)
- W3C standards for writing ontologies in place (RDFS, OWL), but limited expressivity for describing mappings.
- Which language to use?
- How to **publish** mappings/alignments? This is important to make *Open Linked Data*[1] happen!

[1] Combining RDF data that is "out there", e.g. Sindice, DBPedia, SWPipes etc.

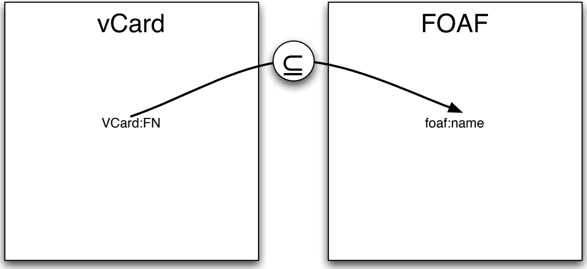
Motivation – Ontology Alignment/Mapping

- Typically: Description of correspondences and overlaps between ontological entities (properties, classes, individuals, etc.)
- W3C standards for writing ontologies in place (RDFS, OWL), but limited expressivity for describing mappings.
- Which language to use?
- How to **publish** mappings/alignments? This is important to make *Open Linked Data*[1] happen!

[1] Combining RDF data that is "out there", e.g. Sindice, DBPedia, SWPipes etc.

Motivation – Scenario

Map from vCard to FOAF:

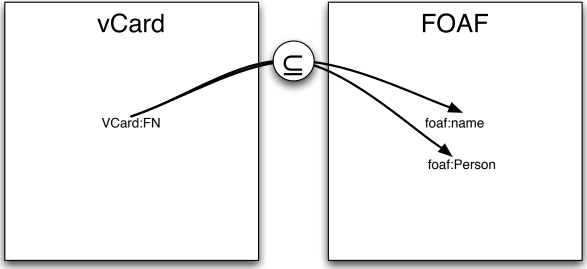


Expressible by `rdfs:subPropertyOf`:

```
VCard:FN rdfs:subPropertyoF foaf:name .
```

Motivation – Scenario

Map from vCard to FOAF:

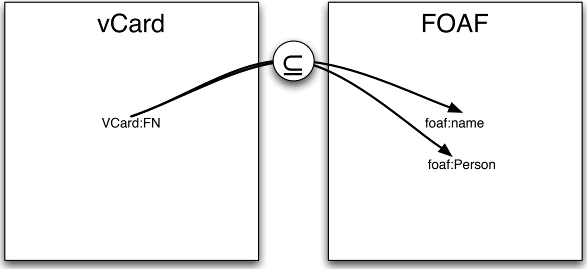


Also expressible in RDFS or in OWL DL:

```
VCard:FN rdfs:subPropertyoF foaf:name.
VCard:FN rdfs:domain foaf:Person.
```

Motivation – Scenario

Map from vCard to FOAF:



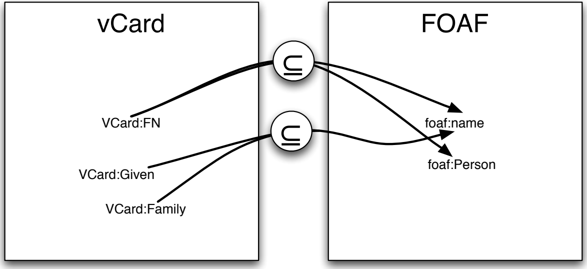
Also expressible in RDFS or in OWL DL:

$\mathtt{VCard{:}FN} \sqsubseteq \mathtt{foaf{:}name}$

$\exists\mathtt{VCard{:}FN}.\top \sqsubseteq \mathtt{foaf{:}Person}$

Motivation – Scenario

Map from vCard to FOAF:



Needs string concatenation, not expressible in OWL or RDFS...

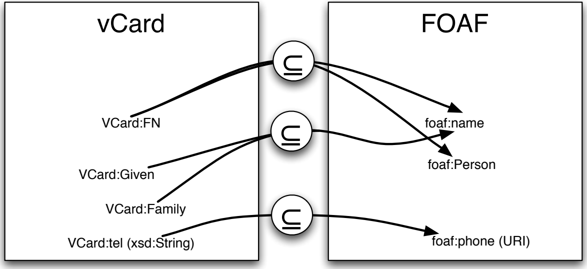
maybe SWRL can help, but

(1) implementations missing

(2) no W3C stamp

Motivation – Scenario

Map from vCard to FOAF:



What shall we do here?

Needs conversion from String to rdf:Resource (URI)...how?

Let's see what SPARQL can do for us...

Mapping by SPARQL

Observation:
SPARQL (Proposed W3C Rec since two weeks, BTW) offers CONSTRUCT queries to generate new graphs from existing ones

```
CONSTRUCT { Basic triple patterns }
FROM dataset (source graph)
WHERE {Pattern}
```

- This may be read as a *view* definition ...
- ... and views can be understood as *(mapping) rules*

Attention: if you allow such views to mutually refer to each other, you get a recursive rules language!

- By OPTIONAL patterns you get even non-monotonicity (negation as failure)
- By bnodes in the CONSTRUCT part, you might run into non-termination issues!

BTW: How can this interact with ontological inferences of OWL and RDFS? (SPARQL is only defined in terms of simple RDF entailment)

Mapping by SPARQL

Observation:
SPARQL (Proposed W3C Rec since two weeks, BTW) offers CONSTRUCT queries to generate new graphs from existing ones

```
CONSTRUCT { Basic triple patterns }
FROM dataset (source graph)
WHERE {Pattern}
```

- This may be read as a *view* definition ...
- ... and views can be understood as *(mapping) rules*

Attention: if you allow such views to mutually refer to each other, you get a recursive rules language!

- By OPTIONAL patterns you get even non-monotonicity (negation as failure)
- By bnodes in the CONSTRUCT part, you might run into non-termination issues!

BTW: How can this interact with ontological inferences of OWL and RDFS?
(SPARQL is only defined in terms of simple RDF entailment)

Mapping by SPARQL

Observation:
SPARQL (Proposed W3C Rec since two weeks, BTW) offers CONSTRUCT queries to generate new graphs from existing ones

```
CONSTRUCT { Basic triple patterns }
FROM dataset (source graph)
WHERE {Pattern}
```

- This may be read as a *view* definition ...
- ... and views can be understood as *(mapping) rules*

Attention: if you allow such views to mutually refer to each other, you get a recursive rules language!

- By OPTIONAL patterns you get even non-monotonicity (negation as failure)
- By bnodes in the CONSTRUCT part, you might run into non-termination issues!

BTW: How can this interact with ontological inferences of OWL and RDFS?
(SPARQL is only defined in terms of simple RDF entailment)

Mapping by SPARQL

Observation:
SPARQL (Proposed W3C Rec since two weeks, BTW) offers CONSTRUCT queries to generate new graphs from existing ones

```
CONSTRUCT { Basic triple patterns }
FROM dataset (source graph)
WHERE {Pattern}
```

- This may be read as a *view* definition ...
- ... and views can be understood as *(mapping) rules*

Attention: if you allow such views to mutually refer to each other, you get a recursive rules language!

- By OPTIONAL patterns you get even non-monotonicity (negation as failure)
- By bnodes in the CONSTRUCT part, you might run into non-termination issues!

BTW: How can this interact with ontological inferences of OWL and RDFS? (SPARQL is only defined in terms of simple RDF entailment)

Mapping by SPARQL

Observation:
SPARQL (Proposed W3C Rec since two weeks, BTW) offers CONSTRUCT queries to generate new graphs from existing ones

```
CONSTRUCT { Basic triple patterns }
FROM dataset (source graph)
WHERE {Pattern}
```

- This may be read as a *view* definition ...
- ... and views can be understood as *(mapping) rules*

Attention: if you allow such views to mutually refer to each other, you get a recursive rules language!

- By OPTIONAL patterns you get even non-monotonicity (negation as failure)
- By bnodes in the CONSTRUCT part, you might run into non-termination issues!

BTW: How can this interact with ontological inferences of OWL and RDFS? (SPARQL is only defined in terms of simple RDF entailment)

Mapping by SPARQL

Observation:
SPARQL (Proposed W3C Rec since two weeks, BTW) offers CONSTRUCT queries to generate new graphs from existing ones

```
CONSTRUCT { Basic triple patterns }
FROM dataset (source graph)
WHERE {Pattern}
```

- This may be read as a *view* definition ...
- ... and views can be understood as *(mapping) rules*

Attention: if you allow such views to mutually refer to each other, you get a recursive rules language!

- By OPTIONAL patterns you get even non-monotonicity (negation as failure)
- By bnodes in the CONSTRUCT part, you might run into non-termination issues!

BTW: How can this interact with ontological inferences of OWL and RDFS? (SPARQL is only defined in terms of simple RDF entailment)

Mapping by SPARQL

Observation:
SPARQL (Proposed W3C Rec since two weeks, BTW) offers CONSTRUCT queries to generate new graphs from existing ones

```
CONSTRUCT { Basic triple patterns }
FROM dataset (source graph)
WHERE {Pattern}
```

- This may be read as a *view* definition ...
- ... and views can be understood as *(mapping) rules*

Attention: if you allow such views to mutually refer to each other, you get a recursive rules language!

- By OPTIONAL patterns you get even non-monotonicity (negation as failure)
- By bnodes in the CONSTRUCT part, you might run into non-termination issues!

BTW: How can this interact with ontological inferences of OWL and RDFS? (SPARQL is only defined in terms of simple RDF entailment)

Outline

Motivation

- Mapping by SPARQL

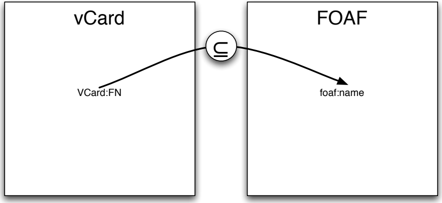
Examples

Implementation

- HEX-Programs
- Demo
- RDFS

Wrap-up

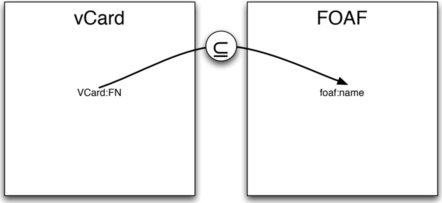
Example 1



```
CONSTRUCT { ?X foaf:name ?Y }
WHERE     { ?X VCard:FN  ?Y }
```

Easy!

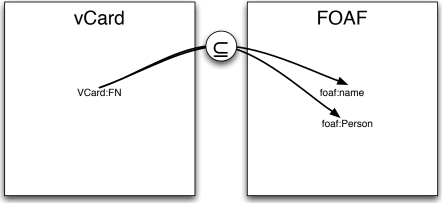
Example 1



```
CONSTRUCT { ?X foaf:name ?Y }
WHERE     { ?X VCard:FN  ?Y }
```

Easy!

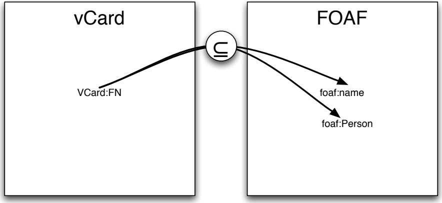
Example 2



```
CONSTRUCT { ?X foaf:name ?Y . ?X rdf:type foaf:person . }
WHERE     { ?X VCard:FN  ?Y }
```

No problem either.

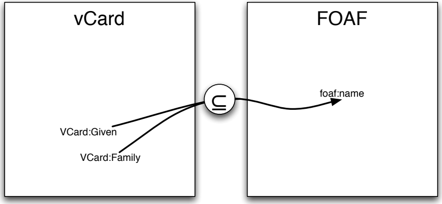
Example 2



```
CONSTRUCT { ?X foaf:name ?Y . ?X rdf:type foaf:person . }
WHERE     { ?X VCard:FN  ?Y }
```

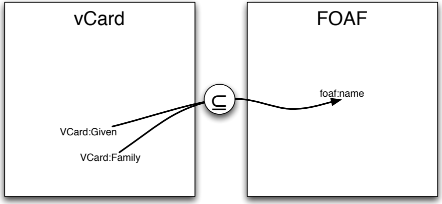
No problem either.

Example 3



```
CONSTRUCT { ?X foaf:name ???  }
WHERE     { ?X VCard:Given ?N. ?X VCard:Family ?F
          }
```

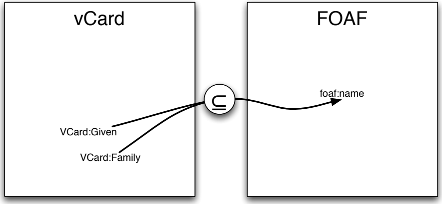
Example 3



```
CONSTRUCT { ?X foaf:name ???  }
WHERE     { ?X VCard:Given ?N. ?X VCard:Family ?F
          }
```

How to tackle? FILTERs?

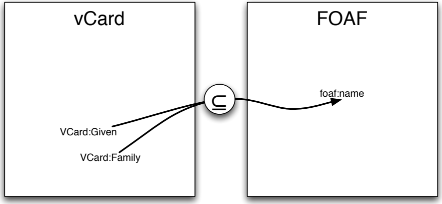
Example 3



```
CONSTRUCT { ?X foaf:name ?FN }
WHERE     { ?X VCard:Given ?N. ?X VCard:Family ?F
            FILTER( ?FN = fn:concat(?N," ",?F))}
```

Doesn't work :-| FILTERs only bind variables, can't create new bindings

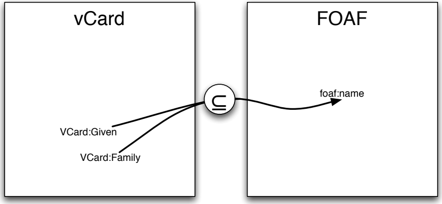
Example 3



```
CONSTRUCT { ?X foaf:name fn:concat(?N," ",?F) }
WHERE     { ?X VCard:Given ?N. ?X VCard:Family ?F
          }
```

You rather want built-in functions in the CONSTRUCT part.
This is what SPARQL++ provides.

Example 3



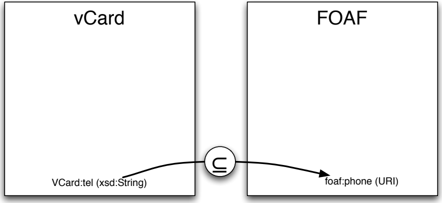
```
CONSTRUCT { ?X foaf:name fn:concat(?N," ",?F) }
WHERE     { ?X VCard:Given ?N. ?X VCard:Family ?F
          }
```

You rather want built-in functions in the CONSTRUCT part.

This is what SPARQL++ provides.

Attention: Value generation in the CONSTRUCT part might again raise non-termination issues!

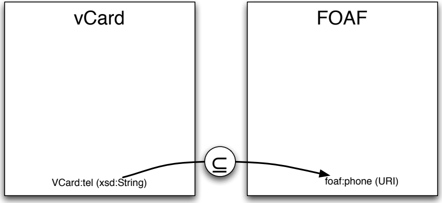
Example 4



With value generation in CONSTRUCTs and respective built-in support, this becomes easy again in SPARQL++:

```
CONSTRUCT { ?X foaf:phone
   rdf:Resource(fn:concat("tel:",fn:encode-for-uri(?T)) . }
WHERE { ?X VCard:tel ?T . }
```

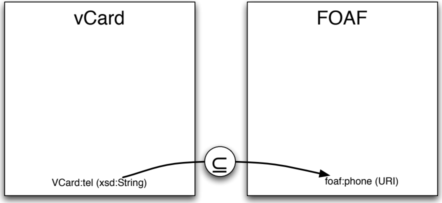
Example 4



With value generation in CONSTRUCTs and respective built-in support, this becomes easy again in SPARQL++:

```
CONSTRUCT { ?X foaf:phone
   rdf:Resource(fn:concat("tel:",fn:encode-for-uri(?T)) . }
WHERE { ?X VCard:tel ?T . }
```

Example 4



With value generation in CONSTRUCTs and respective built-in support, this becomes easy again in SPARQL++:

```
CONSTRUCT { ?X foaf:phone
   rdf:Resource(fn:concat("tel:",fn:encode-for-uri(?T)) . }
WHERE { ?X VCard:tel ?T . }
```

Example 5

We want more: Aggregates!

Example: Map from DOAP to RDF Open Source Software Vocabulary:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

Example 5

We want more: Aggregates!

Example: Map from DOAP to RDF Open Source Software Vocabulary:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

Example 6

Note: “Views” – as we use them here for mappings – are also good for defining implicit knowledge within an RDF graph:

Example: “Import” my co-authors in my FOAF file, mapping from `myPubl.rdf` which uses the Dublin Core (DC) Vocabulary: “I know all my co-authors”

`foafWithImplicitdData.rdf`

```
:me a foaf:Person.
:me foaf:name "Axel Polleres".
CONSTRUCT{ :me foaf:knows _:P . _:P foaf:name ?N }
FROM <http://www.polleres.net/myPubl.rdf>
WHERE { ?P rdf:type :Publ.
        ?P dc:author ?N. FILTER(?N != "Axel Polleres".)  }
:me foaf:knows [foaf:name "Stefan Decker"].
:me foaf:knows [foaf:name "Manfred Hauswirth"].
```

SPARQL++ allows such extended RDF Graphs!

Example 6

Note: “Views” – as we use them here for mappings – are also good for defining implicit knowledge within an RDF graph:

Example: “Import” my co-authors in my FOAF file, mapping from `myPubl.rdf` which uses the Dublin Core (DC) Vocabulary: “I know all my co-authors”

`foafWithImplicitdData.rdf`

```
:me a foaf:Person.
:me foaf:name "Axel Polleres".
CONSTRUCT{ :me foaf:knows _:P . _:P foaf:name ?N }
FROM <http://www.polleres.net/myPubl.rdf>
WHERE { ?P rdf:type :Publ.
        ?P dc:author ?N. FILTER(?N != "Axel Polleres".)  }
:me foaf:knows [foaf:name "Stefan Decker"].
:me foaf:knows [foaf:name "Manfred Hauswirth"].
```

SPARQL++ allows such extended RDF Graphs!

Example 6

Note: “Views” – as we use them here for mappings – are also good for defining implicit knowledge within an RDF graph:

Example: “Import” my co-authors in my FOAF file, mapping from `myPubl.rdf` which uses the Dublin Core (DC) Vocabulary: “I know all my co-authors”

`foafWithImplicitdData.rdf`

```
:me a foaf:Person.
:me foaf:name "Axel Polleres".
CONSTRUCT{ :me foaf:knows _:P . _:P foaf:name ?N }
FROM <http://www.polleres.net/myPubl.rdf>
WHERE { ?P rdf:type :Publ.
        ?P dc:author ?N. FILTER(?N != "Axel Polleres".)   }
:me foaf:knows [foaf:name "Stefan Decker"].
:me foaf:knows [foaf:name "Manfred Hauswirth"].
```

SPARQL++ allows such extended RDF Graphs!

Example 6

Note: “Views” – as we use them here for mappings – are also good for defining implicit knowledge within an RDF graph:

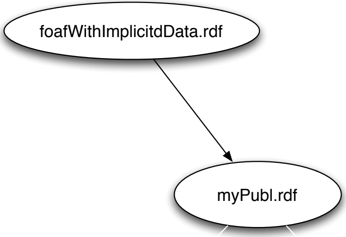
Example: “Import” my co-authors in my FOAF file, mapping from `myPubl.rdf` which uses the Dublin Core (DC) Vocabulary: “I know all my co-authors”

`foafWithImplicitdData.rdf`

```
:me a foaf:Person.
:me foaf:name "Axel Polleres".
CONSTRUCT{ :me foaf:knows _:P . _:P foaf:name ?N }
FROM <http://www.polleres.net/myPubl.rdf>
WHERE { ?P rdf:type :Publ.
        ?P dc:author ?N. FILTER(?N != "Axel Polleres".)   }
:me foaf:knows [foaf:name "Stefan Decker"].
:me foaf:knows [foaf:name "Manfred Hauswirth"].
```

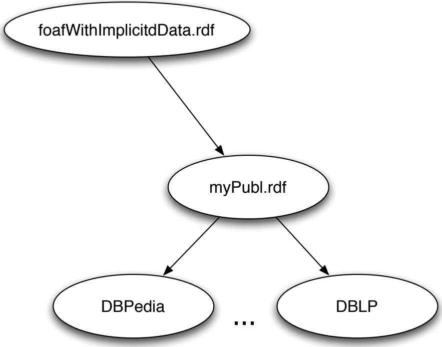
SPARQL++ allows such extended RDF Graphs!

Open Linked data with extended RDF Graphs:



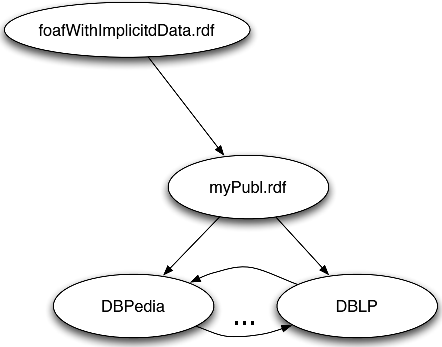
Goal: you can publish extended RDF Graphs, linked via mappings!

Open Linked data with extended RDF Graphs:



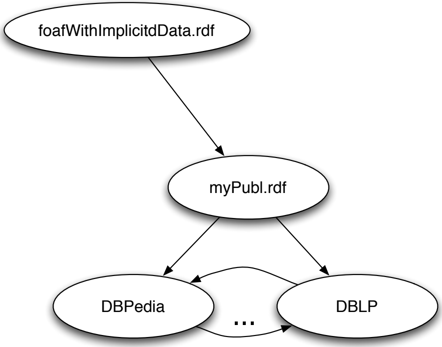
Goal: you can publish extended RDF Graphs, linked via mappings!

Open Linked data with extended RDF Graphs:



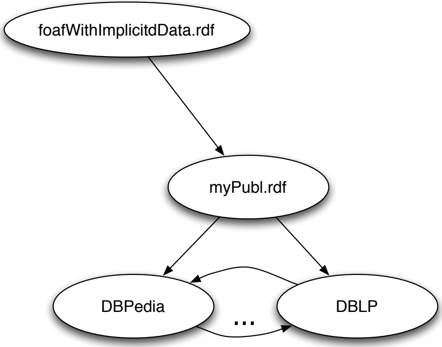
Goal: you can publish extended RDF Graphs, linked via mappings!

Open Linked data with extended RDF Graphs:



Goal: you can publish extended RDF Graphs, linked via mappings!

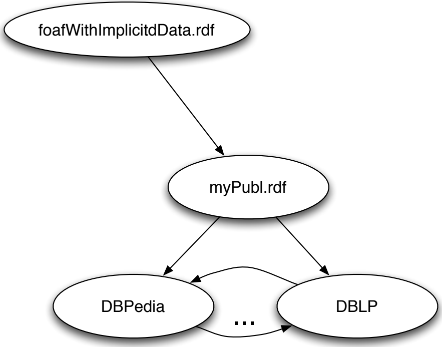
Open Linked data with extended RDF Graphs:



Goal: you can publish extended RDF Graphs, linked via mappings!

Web = HTML + Links

Open Linked data with extended RDF Graphs:



Goal: you can publish extended RDF Graphs, linked via mappings!

Semantic Web = RDF + Mappings

Outline

Motivation

- Mapping by SPARQL

Examples

Implementation

- HEX-Programs
- Demo
- RDFS

Wrap-up

Our Implementation: HEX-Programs

- We translate (possibly nested and cross-referencing) SPARQL queries to so-called HEX programs
- HEX-programs are Datalog programs with negation as failure and a very generic Built-in mechanism.
- A HEX-program is a set of rules:

$$h \leftarrow b_1, \ldots, b_m, \ \texttt{not} \ \ b_{m+1}, \ldots \ \texttt{not} \ \ b_n \tag{1}$$

- where so-called *external atoms* of the form

$$EXT[Input](Output) \tag{2}$$

are allowed.

- Note: External Atoms can take *predicates* as inputs → More generic than “normal” built-in predicates in logic programming!

Our Implementation: HEX-Programs

- We translate (possibly nested and cross-referencing) SPARQL queries to so-called HEX programs
- HEX-programs are Datalog programs with negation as failure and a very generic Built-in mechanism.
- A HEX-program is a set of rules:

$$h \leftarrow b_1, \ldots, b_m, \ \texttt{not} \ \ b_{m+1}, \ldots \ \texttt{not} \ \ b_n \qquad (1)$$

- where so-called *external atoms* of the form

$$EXT[Input](Output) \qquad (2)$$

are allowed.

- Note: External Atoms can take *predicates* as inputs ⟶ More generic than "normal" built-in predicates in logic programming!

Our Implementation: HEX-Programs

- We translate (possibly nested and cross-referencing) SPARQL queries to so-called HEX programs
- HEX-programs are Datalog programs with negation as failure and a very generic Built-in mechanism.
- A HEX-program is a set of rules:

$$h \leftarrow b_1, \ldots, b_m, \ \mathtt{not} \ \ b_{m+1}, \ldots \ \mathtt{not} \ \ b_n \tag{1}$$

- where so-called *external atoms* of the form

$$EXT[Input](Output) \tag{2}$$

are allowed.

- Note: External Atoms can take *predicates* as inputs → More generic than "normal" built-in predicates in logic programming!

Our Implementation: HEX-Programs

- We translate (possibly nested and cross-referencing) SPARQL queries to so-called HEX programs
- HEX-programs are Datalog programs with negation as failure and a very generic Built-in mechanism.
- A HEX-program is a set of rules:

$$h \leftarrow b_1, \ldots, b_m, \ \texttt{not} \ \ b_{m+1}, \ldots \ \texttt{not} \ \ b_n \qquad (1)$$

- where so-called *external atoms* of the form

$$EXT[Input](Output) \qquad (2)$$

  are allowed.
- Note: External Atoms can take *predicates* as inputs → More generic than "normal" built-in predicates in logic programming!

Our Implementation: HEX-Programs

- We translate (possibly nested and cross-referencing) SPARQL queries to so-called HEX programs
- HEX-programs are Datalog programs with negation as failure and a very generic Built-in mechanism.
- A HEX-program is a set of rules:

$$h \leftarrow b_1, \ldots, b_m, \ \mathtt{not} \ \ b_{m+1}, \ldots \ \mathtt{not} \ \ b_n \tag{1}$$

- where so-called *external atoms* of the form

$$EXT[Input](Output) \tag{2}$$

  are allowed.
- Note: External Atoms can take *predicates* as inputs → More generic than "normal" built-in predicates in logic programming!

SPARQL-specific external Atoms:

- `rdf[URL](S,P,O)` ...imports all RDF Triples from a given URL
- $\texttt{CONCAT[Str}_1\texttt{,...,Str}_n\texttt{](Str)}$ concatenates Strings.
- `COUNT[Predicate, BindingPattern](Cnt)` ...returns the count of a certain predicate extension, given a certain binding pattern.
- `MAX[Predicate, BindingPattern](MaxVal)` ...returns the is the lexicographically greatest value among the parameters of `Predicate` in the whole extension (`MIN` analogously).
- $\texttt{SK[Id,V}_1\texttt{,...,V}_n\texttt{](SKTerm)}$ ...similar to CONCAT, but returns a Skolem term, with Skolem function id `Id`. We need this for bnode generation in CONSTRUCTs.
- ...plus some more for handling FILTERs in SPARQL .

SPARQL-specific external Atoms:

- `rdf[URL](S,P,O)` ...imports all RDF Triples from a given URL
- $\texttt{CONCAT[Str}_1\texttt{,...,Str}_n\texttt{](Str)}$ concatenates Strings.
- `COUNT[Predicate, BindingPattern](Cnt)` ...returns the count of a certain predicate extension, given a certain binding pattern.
- `MAX[Predicate, BindingPattern](MaxVal)` ...returns the is the lexicographically greatest value among the parameters of `Predicate` in the whole extension (`MIN` analogously).
- $\texttt{SK[Id,V}_1\texttt{,...,V}_n\texttt{](SKTerm)}$ ...similar to CONCAT, but returns a Skolem term, with Skolem function id `Id`. We need this for bnode generation in CONSTRUCTs.
- ...plus some more for handling FILTERs in SPARQL .

SPARQL-specific external Atoms:

- `rdf[URL](S,P,O)` …imports all RDF Triples from a given URL
- `CONCAT[`$\mathtt{Str}_1$`,...,`$\mathtt{Str}_n$`](Str)` concatenates Strings.
- `COUNT[Predicate, BindingPattern](Cnt)` …returns the count of a certain predicate extension, given a certain binding pattern.
- `MAX[Predicate, BindingPattern](MaxVal)` …returns the is the lexicographically greatest value among the parameters of `Predicate` in the whole extension (`MIN` analogously).
- `SK[Id,`$\mathtt{V}_1$`,...,`$\mathtt{V}_n$`](SKTerm)` …similar to CONCAT, but returns a Skolem term, with Skolem function id `Id`. We need this for bnode generation in CONSTRUCTs.
- …plus some more for handling FILTERs in SPARQL .

SPARQL-specific external Atoms:

- `rdf[URL](S,P,O)` …imports all RDF Triples from a given URL
- `CONCAT[`$\mathtt{Str}_1$`,...,`$\mathtt{Str}_n$`](Str)` concatenates Strings.
- `COUNT[Predicate, BindingPattern](Cnt)` …returns the count of a certain predicate extension, given a certain binding pattern.
- `MAX[Predicate, BindingPattern](MaxVal)` …returns the is the lexicographically greatest value among the parameters of `Predicate` in the whole extension (`MIN` analogously).
- `SK[Id,`$\mathtt{V}_1$`,...,`$\mathtt{V}_n$`](SKTerm)` …similar to CONCAT, but returns a Skolem term, with Skolem function id `Id`. We need this for bnode generation in CONSTRUCTs.
- …plus some more for handling FILTERs in SPARQL .

SPARQL-specific external Atoms:

- `rdf[URL](S,P,O)` . . . imports all RDF Triples from a given URL
- $\texttt{CONCAT[Str}_1\texttt{,...,Str}_n\texttt{](Str)}$ concatenates Strings.
- `COUNT[Predicate, BindingPattern](Cnt)` . . . returns the count of a certain predicate extension, given a certain binding pattern.
- `MAX[Predicate, BindingPattern](MaxVal)` . . . returns the is the lexicographically greatest value among the parameters of `Predicate` in the whole extension (`MIN` analogously).
- $\texttt{SK[Id,V}_1\texttt{,...,V}_n\texttt{](SKTerm)}$ . . . similar to CONCAT, but returns a Skolem term, with Skolem function id `Id`. We need this for bnode generation in CONSTRUCTs.
- . . . plus some more for handling FILTERs in SPARQL .

SPARQL-specific external Atoms:

- `rdf[URL](S,P,O)` ...imports all RDF Triples from a given URL
- `CONCAT[`$\mathtt{Str}_1$`,...,`$\mathtt{Str}_n$`](Str)` concatenates Strings.
- `COUNT[Predicate, BindingPattern](Cnt)` ...returns the count of a certain predicate extension, given a certain binding pattern.
- `MAX[Predicate, BindingPattern](MaxVal)` ...returns the is the lexicographically greatest value among the parameters of `Predicate` in the whole extension (`MIN` analogously).
- `SK[Id,`$\mathtt{V}_1$`,...,`$\mathtt{V}_n$`](SKTerm)` ...similar to CONCAT, but returns a Skolem term, with Skolem function id `Id`. We need this for bnode generation in CONSTRUCTs.
- ...plus some more for handling FILTERs in SPARQL .

Demo Translation

Data in `myPubl.rdf`:

```
:p1 a :Publ.
:p1 dc:author "Axel Polleres".
:p1 dc:author "Francois Scharffe".
:p1 dc:author "Roman Schindlauer".
...
```

Query:

```
CONSTRUCT{ :me foaf:knows _:P . _:P foaf:name ?N }
FROM <http://www.polleres.net/myPubl.rdf>
WHERE { ?P a :Publ. ?P dc:author ?N.
        FILTER(?N != "Axel Polleres") }
```

Demo Translation

Data in `myPubl.rdf`:

```
:p1 a :Publ.
:p1 dc:author "Axel Polleres".
:p1 dc:author "Francois Scharffe".
:p1 dc:author "Roman Schindlauer".
...
```

Translated HEX Program:

```
 triple(S,P,O) :- &rdf["http://www.polleres.net/myPubl.rdf"](S,P,O).
```

Demo Translation

Data in myPubl.rdf:

```
:p1 a :Publ.
:p1 dc:author "Axel Polleres".
:p1 dc:author "Francois Scharffe".
:p1 dc:author "Roman Schindlauer".
...
```

Translated HEX Program:

```
triple(S,P,O) :- &rdf["http://www.polleres.net/myPubl.rdf"](S,P,O).
answer(N,P) :- triple(P,"rdf:type",":Publ"),
               triple(P,"dc:author",N),
               N != "Axel Polleres".
```

Demo Translation

Data in `myPubl.rdf`:

```
:p1 a :Publ.
:p1 dc:author "Axel Polleres".
:p1 dc:author "Francois Scharffe".
:p1 dc:author "Roman Schindlauer".
...
```

Translated HEX Program:

```
 triple(S,P,O) :- &rdf["http://www.polleres.net/myPubl.rdf"](S,P,O).
 answer(N,P) :- triple(P,"rdf:type",":Publ"),
                triple(P,"dc:author",N),
                N != "Axel Polleres".
 triple_result(":me","foaf:knows",Blank_P) :-
    answer(N,P), &SK[ "#genid_P",N,P](Blank_P).
 triple_result(Blank_P,"foaf:name",N) :-
    answer(N,P), &SK[ "#genid_P",N,P](Blank_P).
```

Demo Translation

Data in `myPubl.rdf`:

```
:p1 a :Publ.
:p1 dc:author "Axel Polleres".
:p1 dc:author "Francois Scharffe".
:p1 dc:author "Roman Schindlauer".
...
```

Result:

```
triple_result(":me","foaf:knows","#genid_P('Francois Scharffe',:p1)")
triple_result("#genid_P('Francois Scharffe',:p1)","foaf:name","Francois Scharffe")
triple_result(":me","foaf:knows","#genid_P('Roman Schindlauer',:p1)")
triple_result("#genid_P('Roman Schindlauer',:p1)","foaf:name","Roman Schindlauer")
```

Demo Translation

Data in `myPubl.rdf`:

```
:p1 a :Publ.
:p1 dc:author "Axel Polleres".
:p1 dc:author "Francois Scharffe".
:p1 dc:author "Roman Schindlauer".
...
```

Result:

```
triple_result(":me","foaf:knows","#genid_P(’Francois Scharffe’,:p1)")
triple_result("#genid_P(’Francois Scharffe’,:p1)","foaf:name","Francois Scharffe")
triple_result(":me","foaf:knows","#genid_P(’Roman Schindlauer’,:p1)")
triple_result("#genid_P(’Roman Schindlauer’,:p1)","foaf:name","Roman Schindlauer")
```

Can in turn be translated back to RDF Triples:

```
:me foaf:knows _:b1.
_:b1 foaf:name "Francois Scharffe".
:me foaf:knows _:b2.
_:b2 foaf:name "Roman Schindlauer".
```

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                         triple(P,rdf:type,doap:Project,def).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def),
                   triple(R,doap:revision,V,def).
```

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                         triple(P,rdf:type,doap:Project,def).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def),
                   triple(R,doap:revision,V,def).
```

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                         triple(P,rdf:type,doap:Project,def).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def),
                  triple(R,doap:revision,V,def).
```

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V$_a$) :- MAX[aux$_a$,P,mask](V$_a$),
                                        triple(P,rdf:type,doap:Project,def).
aux$_a$(P,V) :- answer$_a$(P,R,V).
answer$_a$(P,R,V) :- triple(P,doap:release R,def),
                  triple(R,doap:revision,V,def).
```

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                          triple(P,rdf:type,doap:Project,def).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def),
                  triple(R,doap:revision,V,def).
```

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                         triple(P,rdf:type,doap:Project,def).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def),
                   triple(R,doap:revision,V,def).
```

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                         triple(P,rdf:type,doap:Project,def).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def),
                   triple(R,doap:revision,V,def).
```

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                         triple(P,rdf:type,doap:Project,def).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def),
                   triple(R,doap:revision,V,def).
```

`aux` predicate used for for projection; result of automatic translation.

Aggregates Translation:

```
CONSTRUCT { ?P os:latestRelease
   MAX(?V : ?P doap:release ?R. ?R doap:revision ?V) }
WHERE { ?P rdf:type doap:Project .  }
```

will become:

```
triple_result(P,os:latestRelease,V_a) :- MAX[aux_a,P,mask](V_a),
                                         triple(P,rdf:type,doap:Project,def).
aux_a(P,V) :- answer_a(P,R,V).
answer_a(P,R,V) :- triple(P,doap:release R,def),
                   triple(R,doap:revision,V,def).
```

`aux` predicate used for for projection; result of automatic translation.

Find more details on the translation in the paper.

RDFS Inference:

- RDFS Semantics can be expressed in Rules
- So, it is expressible as CONSTRUCT queries

```
CONSTRUCT {?A :subPropertyOf ?C}
  WHERE {?A :subPropertyOf ?B. ?B :subPropertyOf ?C.}
CONSTRUCT {?A :subClassOf ?C}
  WHERE { ?A :subClassOf ?B. ?B :subClassOf ?C. }
CONSTRUCT {?X ?B ?Y}
  WHERE { ?A :subPropertyOf ?B. ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :subClassOf ?B. ?X rdf:type ?A. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :domain ?B. ?X ?A ?Y. }
CONSTRUCT {?Y rdf:type ?B}
  WHERE { ?A :range ?B.  ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :domain ?B. ?C :subPropertyOf ?A. ?X ?C ?Y.}
CONSTRUCT {?Y rdf:type ?B}
  WHERE { ?A :range ?B.  ?C :subPropertyOf ?A. ?X ?C ?Y.}
```

- Simply add these to you extended graph, if RDFS needed. Will be evaluated (recursively) by our translation.

RDFS Inference:

- RDFS Semantics can be expressed in Rules
- So, it is expressible as CONSTRUCT queries

```
CONSTRUCT {?A :subPropertyOf ?C}
  WHERE {?A :subPropertyOf ?B. ?B :subPropertyOf ?C.}
CONSTRUCT {?A :subClassOf ?C}
  WHERE { ?A :subClassOf ?B. ?B :subClassOf ?C. }
CONSTRUCT {?X ?B ?Y}
  WHERE { ?A :subPropertyOf ?B. ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :subClassOf ?B. ?X rdf:type ?A. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :domain ?B. ?X ?A ?Y. }
CONSTRUCT {?Y rdf:type ?B}
  WHERE { ?A :range ?B.  ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :domain ?B. ?C :subPropertyOf ?A. ?X ?C ?Y.}
CONSTRUCT {?Y rdf:type ?B}
  WHERE { ?A :range ?B.  ?C :subPropertyOf ?A. ?X ?C ?Y.}
```

- Simply add these to you extended graph, if RDFS needed. Will be evaluated (recursively) by our translation.

RDFS Inference:

- RDFS Semantics can be expressed in Rules
- So, it is expressible as CONSTRUCT queries

```
CONSTRUCT {?A :subPropertyOf ?C}
  WHERE {?A :subPropertyOf ?B. ?B :subPropertyOf ?C.}
CONSTRUCT {?A :subClassOf ?C}
  WHERE { ?A :subClassOf ?B. ?B :subClassOf ?C. }
CONSTRUCT {?X ?B ?Y}
  WHERE { ?A :subPropertyOf ?B. ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :subClassOf ?B. ?X rdf:type ?A. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :domain ?B. ?X ?A ?Y. }
CONSTRUCT {?Y rdf:type ?B}
  WHERE { ?A :range ?B.  ?X ?A ?Y. }
CONSTRUCT {?X rdf:type ?B}
  WHERE { ?A :domain ?B. ?C :subPropertyOf ?A. ?X ?C ?Y.}
CONSTRUCT {?Y rdf:type ?B}
  WHERE { ?A :range ?B.  ?C :subPropertyOf ?A. ?X ?C ?Y.}
```

- Simply add these to you extended graph, if RDFS needed. Will be evaluated (recursively) by our translation.

Outline

Motivation

Mapping by SPARQL

Examples

Implementation

HEX-Programs

Demo

RDFS

Wrap-up

Summary

Take-home message:

- Even simple ontologies are not so easy to align.
- Current standards don’t provide the right “ingredients” to describe the necessary mappings
- SPARQL++ fills this gap and adds more...
- SPARQL++ allows the definition of “Extended Graphs”, i.e. Mappings+RDF Data in one file, similar to “Networked Graphs” [Schenk and Staab, 2007][2]

What more will you find in the paper:

- Formal Semantics of Extended Graphs, based on Stable Model Semantics for HEX-Programs.
- A “safety condition” for recursive mappings with bnodes and value-generating CONSTRUCTs.

[2] diff: stable vs. well-founded semantics, safe value-generation allowed, aggregates, built-ins

Summary

Take-home message:

- Even simple ontologies are not so easy to align.
- Current standards don't provide the right "ingredients" to describe the necessary mappings
- SPARQL++ fills this gap and adds more...
- SPARQL++ allows the definition of "Extended Graphs", i.e. Mappings+RDF Data in one file, similar to "Networked Graphs" [Schenk and Staab, 2007][2]

What more will you find in the paper:

- Formal Semantics of Extended Graphs, based on Stable Model Semantics for HEX-Programs.
- A "safety condition" for recursive mappings with bnodes and value-generating CONSTRUCTs.

[2] diff: stable vs. well-founded semantics, safe value-generation allowed, aggregates, built-ins

Summary

Take-home message:

- Even simple ontologies are not so easy to align.
- Current standards don't provide the right "ingredients" to describe the necessary mappings
- SPARQL++ fills this gap and adds more...
- SPARQL++ allows the definition of "Extended Graphs", i.e. Mappings+RDF Data in one file, similar to "Networked Graphs" [Schenk and Staab, 2007][2]

What more will you find in the paper:

- Formal Semantics of Extended Graphs, based on Stable Model Semantics for HEX-Programs.
- A "safety condition" for recursive mappings with bnodes and value-generating CONSTRUCTs.

[2] diff: stable vs. well-founded semantics, safe value-generation allowed, aggregates, built-ins

Summary

Take-home message:

- Even simple ontologies are not so easy to align.
- Current standards don't provide the right "ingredients" to describe the necessary mappings
- SPARQL++ fills this gap and adds more...
- SPARQL++ allows the definition of "Extended Graphs", i.e. Mappings+RDF Data in one file, similar to "Networked Graphs" [Schenk and Staab, 2007][2]

What more will you find in the paper:

- Formal Semantics of Extended Graphs, based on Stable Model Semantics for HEX-Programs.
- A "safety condition" for recursive mappings with bnodes and value-generating CONSTRUCTs.

[2] diff: stable vs. well-founded semantics, safe value-generation allowed, aggregates, built-ins

Summary

Take-home message:

- Even simple ontologies are not so easy to align.
- Current standards don't provide the right "ingredients" to describe the necessary mappings
- SPARQL++ fills this gap and adds more...
- SPARQL++ allows the definition of "Extended Graphs", i.e. Mappings+RDF Data in one file, similar to "Networked Graphs" [Schenk and Staab, 2007][2]

What more will you find in the paper:

- Formal Semantics of Extended Graphs, based on Stable Model Semantics for HEX-Programs.
- A "safety condition" for recursive mappings with bnodes and value-generating CONSTRUCTs.

[2] diff: stable vs. well-founded semantics, safe value-generation allowed, aggregates, built-ins

Summary

Take-home message:

- Even simple ontologies are not so easy to align.
- Current standards don't provide the right "ingredients" to describe the necessary mappings
- SPARQL++ fills this gap and adds more...
- SPARQL++ allows the definition of "Extended Graphs", i.e. Mappings+RDF Data in one file, similar to "Networked Graphs" [Schenk and Staab, 2007][2]

What more will you find in the paper:

- Formal Semantics of Extended Graphs, based on Stable Model Semantics for HEX-Programs.
- A "safety condition" for recursive mappings with bnodes and value-generating CONSTRUCTs.

[2] diff: stable vs. well-founded semantics, safe value-generation allowed, aggregates, built-ins

Summary

Take-home message:

- Even simple ontologies are not so easy to align.
- Current standards don't provide the right "ingredients" to describe the necessary mappings
- SPARQL++ fills this gap and adds more...
- SPARQL++ allows the definition of "Extended Graphs", i.e. Mappings+RDF Data in one file, similar to "Networked Graphs" [Schenk and Staab, 2007][2]

What more will you find in the paper:

- Formal Semantics of Extended Graphs, based on Stable Model Semantics for HEX-Programs.
- A "safety condition" for recursive mappings with bnodes and value-generating CONSTRUCTs.

[2] diff: stable vs. well-founded semantics, safe value-generation allowed, aggregates, built-ins

Summary

Take-home message:

- Even simple ontologies are not so easy to align.
- Current standards don't provide the right "ingredients" to describe the necessary mappings
- SPARQL++ fills this gap and adds more...
- SPARQL++ allows the definition of "Extended Graphs", i.e. Mappings+RDF Data in one file, similar to "Networked Graphs" [Schenk and Staab, 2007][2]

What more will you find in the paper:

- Formal Semantics of Extended Graphs, based on Stable Model Semantics for HEX-Programs.
- A "safety condition" for recursive mappings with bnodes and value-generating CONSTRUCTs.

[2] diff: stable vs. well-founded semantics, safe value-generation allowed, aggregates, built-ins

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql
Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql
Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql

Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql
Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql
Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql

Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql
Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql

Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql

Thanks! Questions please! :-)

Next Steps

- SPARQL++, Extended Graphs are intended as a means to weave the Semantic **Web**...
- ... i.e. allow to publish mappings and implicit RDF data on the Web.
- As the community picks up SPARQL, people will be able to publish mappings for free, without having to learn a new syntax.
- Necessary next step: Optimization of distributed querying: We conceive a Linked Open Data Web rather a network of SPARQL++ endpoints than a network of RDF files.
- Full SPARQL spec compliance is tedious, as SPARQL semantics is not purely declarative.
- Ontological inference beyond RDFS, or OWL Horst at max. unlikely. (Personal opinion: Higher expressivity languages rather important for TBox only, than for instance semantics and query answering)
- More related efforts on the way, e.g. http://pipes.deri.org, http://www.sindice.com, dlvhex-server

Stay Tuned: http://www.polleres.net/dlvhex-sparql
Thanks! Questions please! :-)