

A SPARQL Query Engine over Web Ontologies using Contextual Logic Programming

Mestrado em Engenharia Informática

Nuno Lopes

Universidade de Évora

19 de Dezembro de 2007

- 1 Trabalho Proposto
- 2 Semantic Web
- 3 Programação em Lógica Contextual
- 4 Sistema XPTO
 - Representação de Ontologias
 - Interrogação de Ontologias
- 5 Resposta a queries SPARQL
- 6 Conclusões

Motivação e trabalho proposto

- Programação em Lógica Contextual (CxLP) e ISCO

Motivação e trabalho proposto

- Programação em Lógica Contextual (CxLP) e ISCO
- Ontologias Web

Motivação e trabalho proposto

- Programação em Lógica Contextual (CxLP) e ISCO
- Ontologias Web

Representação para ontologias em CxLP

- Integração com o ISCO e outras fontes de dados
- Permitir fazer queries internas (top-level)

Motivação e trabalho proposto

- Programação em Lógica Contextual (CxLP) e ISCO
- Ontologias Web

Representação para ontologias em CxLP

- Integração com o ISCO e outras fontes de dados
- Permitir fazer queries internas (top-level)

Com base na representação:

- Permitir interrogar a representação com SPARQL

Motivação e trabalho proposto

- Programação em Lógica Contextual (CxLP) e ISCO
- Ontologias Web

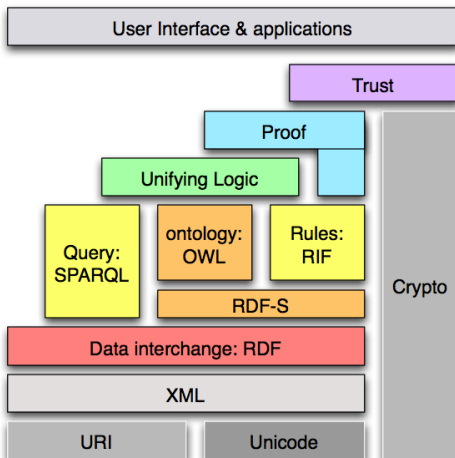
Representação para ontologias em CxLP

- Integração com o ISCO e outras fontes de dados
- Permitir fazer queries internas (top-level)

Com base na representação:

- Permitir interrogar a representação com SPARQL
- Permitir fazer queries a serviços web SPARQL (Cláudio Fernandes)

Camadas da Semantic Web



GNU Prolog/CX

- Units:

```
:- unit(foo(A)).
```

```
item(A).
```

```
:- unit(bar(A)).
```

```
item(A).
```

```
item(A) :- ^ item(A).
```

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).  
  
item(A).
```

```
:- unit(bar(A)).  
  
item(A).  
item(A) :- ^ item(A).
```

- Contextos:

```
foo(b) :> item(X).
```

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).  
  
item(A).
```

```
:- unit(bar(A)).  
  
item(A).  
item(A) :- ^ item(A).
```

- Contextos:

foo(b) :> item(X).

X = b

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).  
  
item(A).
```

```
:- unit(bar(A)).  
  
item(A).  
item(A) :- ^ item(A).
```

- Contextos:

foo(b) :> item(X).

X = b

foo(1) :> bar(a) :> item(X).

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).  
  
item(A).
```

```
:- unit(bar(A)).  
  
item(A).  
item(A) :- ^ item(A).
```

- Contextos:

foo(b) :> item(X). X = b

foo(1) :> bar(a) :> item(X). X = a

GNU Prolog/CX

- Units:

```
:- unit(foo(A)).  
  
item(A).
```

```
:- unit(bar(A)).  
  
item(A).  
item(A) :- ^ item(A).
```

- Contextos:

```
foo(b) :> item(X).
```

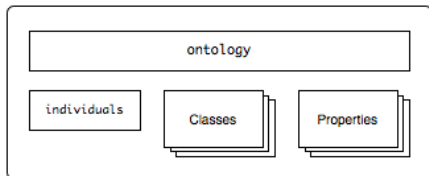
```
X = b
```

```
foo(1) :> bar(a) :> item(X).
```

```
X = a ;
```

```
X = 1
```

Representação de Ontologias



Ontologias são representadas através de units:

- uma unit que lista as classes e propriedades da ontologia;
- uma unit que representa dos indivíduos;
- uma unit para cada classe OWL;
- uma unit para cada propriedade

Unit “Ontology”

Esta unit representa a informação da ontologia:

- namespaces XML
- headers
- classes
- propriedades

Unit “Individuals”

Indivíduos Os indivíduos são guardados, juntamente com a sua classe, no predicado:

```
individual_class(CLASS, INDIVIDUAL).
```

Unit “Individuals”

Indivíduos Os indivíduos são guardados, juntamente com a sua classe, no predicado:

`individual_class(CLASS, INDIVIDUAL).`

Propriedades As propriedades de cada indivíduo são guardadas como triplos no predicado: `property/3`.

`property(INDIVIDUAL, PROPERTY, VALUE).`

Unit “Individuals”

Indivíduos Os indivíduos são guardados, juntamente com a sua classe, no predicado:

`individual_class(CLASS, INDIVIDUAL).`

Propriedades As propriedades de cada indivíduo são guardadas como triplos no predicado: `property/3`.

`property(INDIVIDUAL, PROPERTY, VALUE).`

Relações entre indivíduos:

- `differentFrom(IND1, IND2).`

- `sameAs(IND1, IND2).`

Units de classes

- Cada unit representa uma classe da ontologia
- Guarda a informação sobre a classe como factos na unit:
 - Restrições sobre as propriedades dos indivíduos
 - Herança de classes
- Define predicados usados para interrogar a ontologia:
 - `class_name(NAME)`
 - `superClassOf(CLASS)`

Units de Propriedades

Cada unit contém informação sobre a propriedade que representa:

- tipo da propriedade (datatype ou object)
- “domain” e “range” (domínio e contra-domínio) da propriedade
- herança e relações entre propriedades.

Units de Propriedades

Cada unit contém informação sobre a propriedade que representa:

- tipo da propriedade (datatype ou object)
- “domain” e “range” (domínio e contra-domínio) da propriedade
- herança e relações entre propriedades.

Definem também o predicado para aceder ao seu valor, dado o nome do indivíduo.

```
:- unit('propName'(A)).  
  
item(B) :-  
    :^ item(B),  
    property(B, 'propName', A).
```

Interrogar uma ontologia

- A maneira mais directa é usar o “goal” item/1 (com uma unit de classe no contexto)
- Este predicado instância, por backtrack, o seu argumento com cada indivíduo da classe
- Existe também a possibilidade de interrogar todos os individuos da ontologia (omitindo a classe)

```
| ?- 'ClassName' /> item(A).  
A = 'IndividualName'
```

- É possível aceder ao valor das propriedades dos indivíduos através da inclusão da unit que representa a propriedade no contexto
- O argumento dessa unit vai ser instânciado com o valor da propriedade para o indivíduo

```
| ?- 'IceWine' /> hasFlavor(F) :> hasBody(B) :>  
  item(I).  
B = 'Medium'  
F = 'Moderate'  
I = 'SelaksIceWine' ?
```

Outras units para interrogação

- individual/1** unifica o seu argumento com o nome do indivíduo (como o goal `item/1`)
- class/1** unifica o argumento com a classe do indivíduo
- property/2** permite interrogar o nome da propriedade baseado no seu valor
- optional/1** Esta unit recebe como argumento um contexto a ser executado e sucede com os resultados se o contexto suceder. Caso contrário irá suceder com as variáveis não instanciadas.

Resposta a interrogações SPARQL

- parser SPARQL escrito em Flex e Bison

Resposta a interrogações SPARQL

- parser SPARQL escrito em Flex e Bison
- gera um contexto que representa a interrogação

Resposta a interrogações SPARQL

- parser SPARQL escrito em Flex e Bison
- gera um contexto que representa a interrogação
- o contexto é executado para obter os resultados

Resposta a interrogações SPARQL

- parser SPARQL escrito em Flex e Bison
- gera um contexto que representa a interrogação
- o contexto é executado para obter os resultados
- e formatado de acordo com as especificações XML do protocolo SPARQL

```
1 SELECT
2     ?flavor ?body
3 WHERE {
4     ?t :hasFlavor    ?flavor .
5     ?t :hasBody      ?body .
6 }
```

```
1 [where([set([
2     triple(A,hasFlavor,B),
3     triple(A,hasBody,C) ] )
4     ]),
5     select([flavor=B,body=C]),
6     vars([flavor=B,body=C,t=A]),
7 ]]
```

SPARQL engine

- Cada “funcionalidade” SPARQL é implementada como uma unit
- A unit triple/3 é responsável pela instanciação das variáveis com base na representação da ontologia.

SPARQL engine

- Cada “funcionalidade” SPARQL é implementada como uma unit
- A unit triple/3 é responsável pela instanciação das variáveis com base na representação da ontologia.

```
1 :- unit(triple(S, P, 0)).  
2  
3 item :-  
4     /> property(P,0) :> item(S).
```

Conclusões

- Representação para uma ontologia
- Integração com bases de dados relacionais
- “query engine” SPARQL

Conclusões

- Representação para uma ontologia
- Integração com bases de dados relacionais
- “query engine” SPARQL

Trabalho Futuro:

- Permitir representar várias ontologias
- Melhorar o suporte para semântica do OWL
- Completar o suporte SPARQL

Obrigado