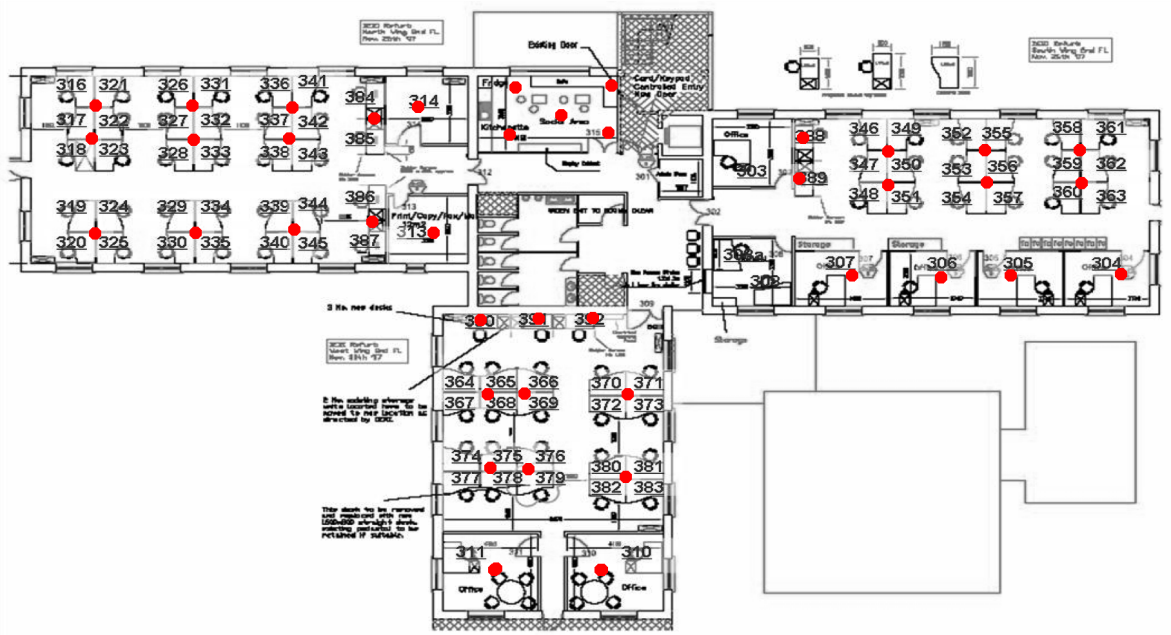
A Deductive System for Annotated RDFS

DERI Institute Meeting

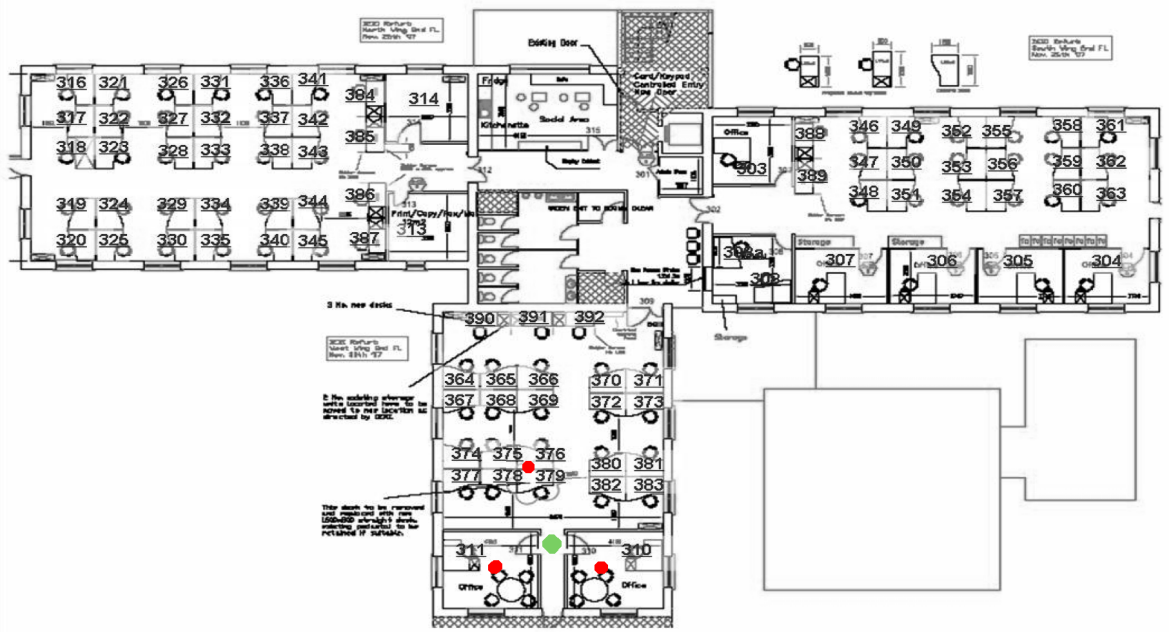
Umberto Straccia Nuno Lopes Gergely Lukácsy
Antoine Zimmermann Axel Polleres
Presented by: Nuno Lopes

May 28, 2010

Sensor data



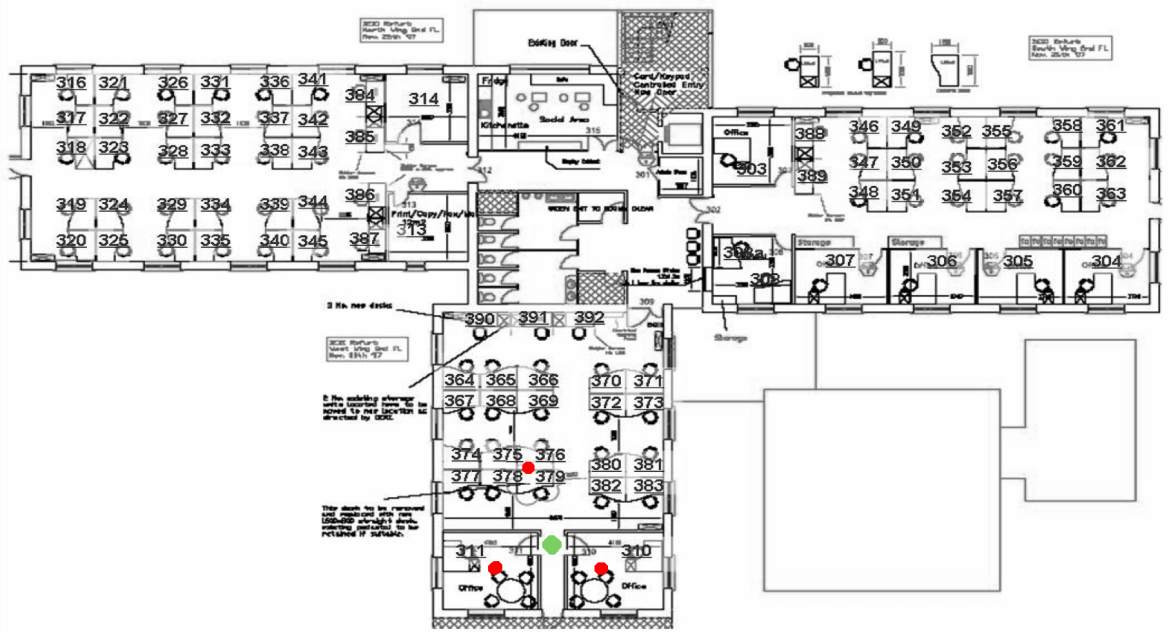
Sensor data



sensors readings output:

```
2010-05-28 14:57:51   10.254.2.15   4302   83
2010-05-28 14:57:51   10.254.3.1    4302   83
2010-05-28 14:57:51   10.254.2.6    4302   83
```

Sensor data

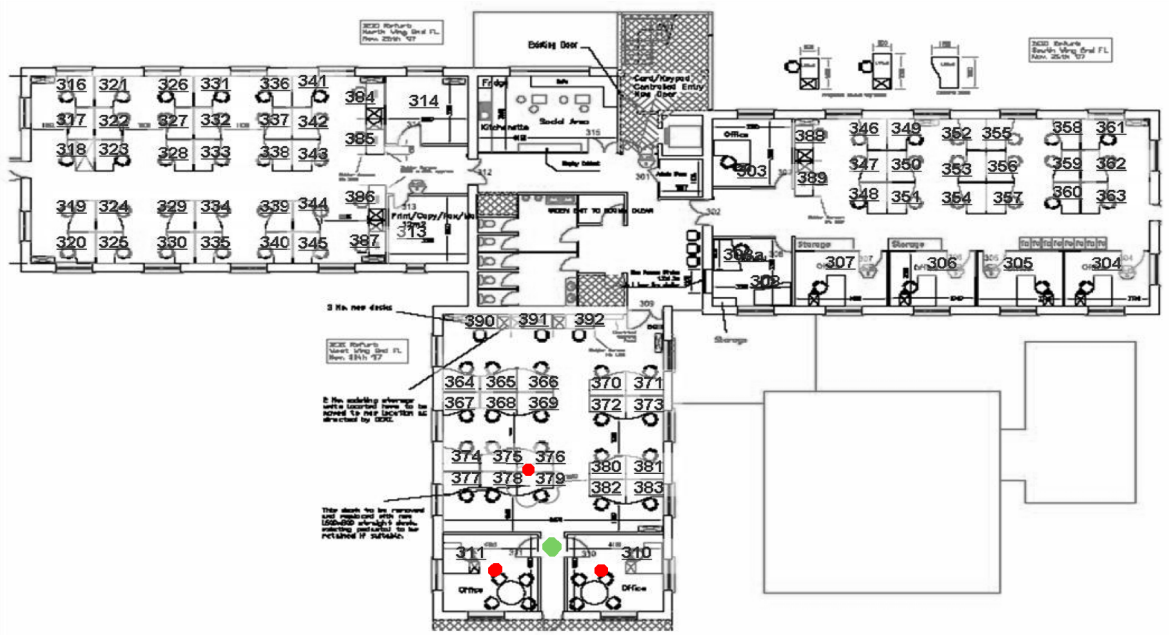


sensors readings output:

```
2010-05-28 14:57:51   10.254.2.15   4302   83
2010-05-28 14:57:51   10.254.3.1    4302   83
2010-05-28 14:57:51   10.254.2.6    4302   83
```

Convert to

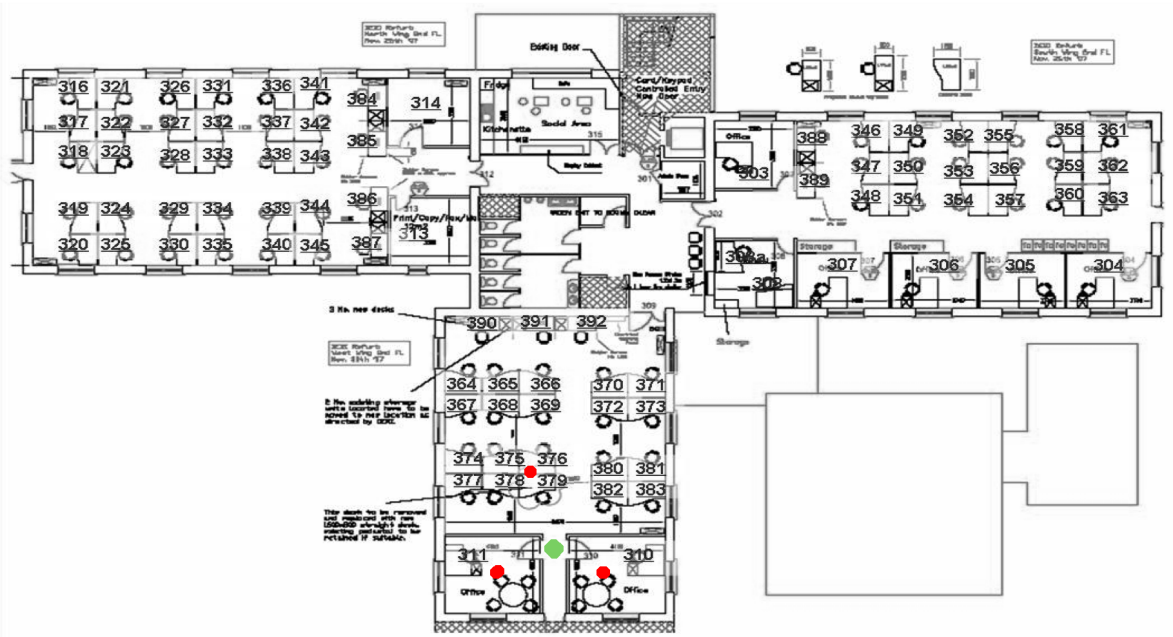
Sensor data



Queries over sensor data:

Q1: *"Where was Stefan before this institute meeting?"*

Sensor data



Queries over sensor data:

Q1: *"Where was Stefan before this institute meeting?"*

Q2: *"When were Stefan and Manfred in the same room?"*

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
```

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room311 .
```

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room310 .
```

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room310 .
```

Not enough info!

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room310 .
```

Not enough info!

- Domain vocabulary/ontology

```
:record1 a           :SensorRecord;
         :tag        :tag4302;
         :locatedIn  :room311;
         :timestamp "2010-05-28 14:57:51" .
```

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room310 .
```

Not enough info!

- Reification

```
:record1 rdf:type rdf:Statement
         rdf:subject :tag4302;
         rdf:predicate :locatedIn ;
         rdf:object :room311 ;
         :timestamp "2010-05-28 14:57:51" .
```

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room310 .
```

Not enough info!

- Reification

```
:record1 rdf:type rdf:Statement
         rdf:subject :tag4302;
         rdf:predicate :locatedIn ;
         rdf:object :room311 ;
         :timestamp "2010-05-28 14:57:51" .
```

No defined semantics!

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room310 .
```

Not enough info!

- Reification

```
:record1 rdf:type rdf:Statement
         rdf:subject :tag4302;
         rdf:predicate :locatedIn ;
         rdf:object :room311 ;
         :timestamp "2010-05-28 14:57:51" .
```

No defined semantics!

- Annotated RDF

```
:tag4302 :locatedIn :room311 . "2010-05-28 14:57:51"
```

Represent sensor data as RDF

- RDF triples

```
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room311 .
:tag4302 :locatedIn :room310 .
```

Not enough info!

- Reification

```
:record1 rdf:type rdf:Statement
         rdf:subject :tag4302;
         rdf:predicate :locatedIn ;
         rdf:object :room311 ;
         :timestamp "2010-05-28 14:57:51" .
```

No defined semantics!

- Annotated RDF

```
:tag4302 :locatedIn :room311 . "2010-05-28 14:57:51"
```

Annotations refer to a specific **domain**, like temporal.

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = [-\infty, +\infty]$ $\bot = []$

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = [-\infty, +\infty]$ $\bot = []$
  operator ($\wedge$) for conjunction of annotations

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = [-\infty, +\infty]$ $\bot = []$
  operator ($\wedge$) for conjunction of annotations: $\cap$

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = [-\infty, +\infty]$ $\bot = []$
  operator ($\wedge$) for conjunction of annotations: $\cap$



`["09:25","11:49"]` $\wedge$ `["10:35", "12:57"] = ["10:35", "11:49"]`

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = [-\infty, +\infty]$ $\bot = []$
  operator ($\wedge$) for conjunction of annotations: $\cap$
  operator ($\vee$) for combining annotations

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = [-\infty, +\infty]$ $\bot = []$
  operator ($\wedge$) for conjunction of annotations: $\cap$
  operator ($\vee$) for combining annotations: $\cup$

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = [-\infty, +\infty]$ $\bot = []$
  operator ($\wedge$) for conjunction of annotations: $\cap$
  operator ($\vee$) for combining annotations: $\cup$



$$\texttt{["09:25","11:49"]} \vee \texttt{["10:35", "12:57"]} = \texttt{["09:25", "12:57"]}$$

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . ["09:25", "11:49"]
:tag4302 :locatedIn :room311 . ["10:35", "12:57"]
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `["14:35", "14:57"]`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = [-\infty, +\infty]$ $\bot = []$
  operator ($\wedge$) for conjunction of annotations: $\cap$
  operator ($\vee$) for combining annotations: $\cup$



["09:25","11:49"] $\vee$ ["14:35", "15:57"] = ["09:25","11:49"],
["14:35", "15:57"]

Annotation Domains

Temporal domain example:

```
:tag4302 :locatedIn :room311 . {["09:25", "11:49"]}
:tag4302 :locatedIn :room311 . {["10:35", "12:57"]}
```

To define a **new domain** you need to specify:

- the *representation* of the annotations: `{["14:35", "14:57"]}`
- an *order* between the elements: $\subseteq$

  universal ($\top$) and *empty* ($\bot$) annotations: $\top = \{[-\infty, +\infty]\}$ $\bot = \{[]\}$
  operator ($\wedge$) for conjunction of annotations: $\cap$
  operator ($\vee$) for combining annotations: $\cup$



`["09:25","11:49"]` $\vee$ `["14:35", "15:57"] = {["09:25","11:49"], ["14:35", "15:57"]}`

Other domains: Examples

Trust/Fuzzy

```
:tag4302 :locatedIn :room311 . 0.9
:tag4302 :locatedIn :room310 . 0.2
```

annotations: $[0,1]$

order: $\leq$

$\wedge$: *min* $\vee$: *max*

$\top = 1, \quad \bot = 0$

Other domains: Examples

Trust/Fuzzy

```
:tag4302 :locatedIn :room311 . 0.9
:tag4302 :locatedIn :room310 . 0.2
```

annotations: [0,1]

order: $\leq$

$\wedge$: *min* $\vee$: *max*

$\top = 1, \quad \bot = 0$

Provenance

```
foaf:Person subClassOf foaf:Agent .
            {http://xmlns.com/foaf/0.1/}
```

annotations: {set of URIs}

order: $\subseteq$

$\wedge$: $\cap$ $\vee$: $\cup$

$\top$ = {list of all URIs},
$\bot = \{\}$

Other domains: Examples

Trust/Fuzzy

```
:tag4302 :locatedIn :room311 . 0.9
:tag4302 :locatedIn :room310 . 0.2
```

annotations: [0,1]

order: $\leq$

$\wedge$: *min* $\vee$: *max*

$\top = 1, \quad \bot = 0$

Provenance*

```
foaf:Person subClassOf foaf:Agent .
            {http://xmlns.com/foaf/0.1/}
```

annotations: {set of URIs}

order: $\subseteq$

$\wedge$: $\cap$ $\vee$: $\cup$

$\top$ = {list of all URIs},
$\bot = \{\}$

* this representation of provenance is a first draft

Other domains: Examples

Trust/Fuzzy

```
:tag4302 :locatedIn :room311 . 0.9
:tag4302 :locatedIn :room310 . 0.2
```

annotations: [0,1]

order: $\leq$

$\wedge$: *min* $\vee$: *max*

$\top = 1, \quad \bot = 0$

Provenance*

```
foaf:Person subClassOf foaf:Agent .
            {http://xmlns.com/foaf/0.1/}
```

annotations: {set of URIs}

order: $\subseteq$

$\wedge$: $\cap$ $\vee$: $\cup$

$\top$ = {list of all URIs},
$\bot = \{\}$

* this representation of provenance is a first draft

Our generic semantics allows to combine domains:

```
:tag4302 :locatedIn :room311 . (["14:25", "14:57"], 0.8)
```

Integration with RDF

Transparent integration of annotated and classical RDF

```
:stefan foaf:name "Stefan Decker" .
:tag4302 :assignedTo :stefan .
:tag4302 :locatedIn :room311 .      ["14:25", "14:57"]
```

Integration with RDF

Transparent integration of annotated and classical RDF

```
:stefan foaf:name "Stefan Decker" .   [−∞, +∞]
:tag4302 :assignedTo :stefan .        [−∞, +∞]
:tag4302 :locatedIn :room311 .        ["14:25", "14:57"]
```

Possible approaches:

- use $\top$ as annotation

Integration with RDF

Transparent integration of annotated and classical RDF

```
:stefan foaf:name "Stefan Decker" .  [_:a, _:b]
:tag4302 :assignedTo :stefan .       [_:a, _:b]
:tag4302 :locatedIn :room311 .       ["14:25", "14:57"]
```

Possible approaches:

- use $\top$ as annotation
- triple is valid at a time interval common throughout the graph

  requires blank nodes in annotations

Integration with RDF

Transparent integration of annotated and classical RDF

```
:stefan foaf:name "Stefan Decker" .  [−∞, now]
:tag4302 :assignedTo :stefan .       [−∞, now]
:tag4302 :locatedIn :room311 .       ["14:25", "14:57"]
```

Possible approaches:

- use $\top$ as annotation
- triple is valid at a time interval common throughout the graph
  requires blank nodes in annotations
- triple is valid until "now" ([Temporal RDF, Gutierrez et al, 2005])
  represents current time

Integration with RDF

Transparent integration of annotated and classical RDF

```
:stefan foaf:name "Stefan Decker" .   [−∞, +∞]
:tag4302 :assignedTo :stefan .        [−∞, +∞]
:tag4302 :locatedIn :room311 .        ["14:25", "14:57"]
```

Possible approaches:

- use $\top$ as annotation "upwards compatible"
- triple is valid at a time interval common throughout the graph
  requires blank nodes in annotations
- triple is valid until "now" ([Temporal RDF, Gutierrez et al, 2005])
  represents current time

Annotated RDFS Inference rules

Inference rules are **independent** of the annotation domain

Annotated RDFS Inference rules

Inference rules are **independent** of the annotation domain

RDFS "rdfs:domain" rule:

```
   ?SomeProp rdfs:domain ?SomeClass
   ?x ?SomeProp ?y
⇒  ?x rdf:type ?SomeClass
```

Annotated RDFS Inference rules

Inference rules are **independent** of the annotation domain

RDFS "rdfs:domain" rule:

```
   ?SomeProp rdfs:domain ?SomeClass
   ?x ?SomeProp ?y
⇒  ?x rdf:type ?SomeClass
```

Example:

```
   :worksFor rdfs:domain :Employee
   :nuno :worksFor :DERI
⇒  :nuno rdf:type :Employee
```

Annotated RDFS Inference rules

Inference rules are **independent** of the annotation domain

Annotated RDFS "rdfs:domain" rule:

```
   ?SomeProp rdfs:domain ?SomeClass    ?v1
   ?x ?SomeProp ?y                     ?v2
⇒  ?x rdf:type ?SomeClass              ?v1 ∧ ?v2
```

Example:

```
   :worksFor rdfs:domain :Employee
   :nuno :worksFor :DERI
⇒  :nuno rdf:type :Employee
```

Annotated RDFS Inference rules

Inference rules are **independent** of the annotation domain

Annotated RDFS "rdfs:domain" rule:

`?SomeProp rdfs:domain ?SomeClass`	$?v1$
`?x ?SomeProp ?y`	$?v2$
$\Rightarrow$ `?x rdf:type ?SomeClass`	$?v1 \wedge ?v2$

Example:

`:worksFor rdfs:domain :Employee`	$[-\infty, +\infty]$
`:nuno :worksFor :DERI`	["2009-01-01", "2010-05-28"]
$\Rightarrow$ `:nuno rdf:type :Employee`	["2009-01-01", "2010-05-28"]

Annotated RDFS Inference rules

Inference rules are **independent** of the annotation domain

Annotated RDFS "rdfs:domain" rule:

`?SomeProp rdfs:domain ?SomeClass`	$?v1$
`?x ?SomeProp ?y`	$?v2$
$\Rightarrow$ `?x rdf:type ?SomeClass`	$?v1 \wedge ?v2$

Example:

`:worksFor rdfs:domain :Employee`	$[-\infty, +\infty]$
`:nuno :worksFor :DERI`	["2009-01-01", "2010-05-28"]
$\Rightarrow$ `:nuno rdf:type :Employee`	["2009-01-01", "2010-05-28"]

- Extra rule to group annotations triples ($\vee$):

`:nuno :worksFor :DERI`	["2008-05-01", "2010-01-01"]
`:nuno :worksFor :DERI`	["2009-01-01", "2010-05-28"]

Annotated RDFS Inference rules

Inference rules are **independent** of the annotation domain

Annotated RDFS "rdfs:domain" rule:

`?SomeProp rdfs:domain ?SomeClass`	$?v1$
`?x ?SomeProp ?y`	$?v2$
$\Rightarrow$ `?x rdf:type ?SomeClass`	$?v1 \wedge ?v2$

Example:

`:worksFor rdfs:domain :Employee`	$[-\infty, +\infty]$
`:nuno :worksFor :DERI`	["2009-01-01", "2010-05-28"]
$\Rightarrow$ `:nuno rdf:type :Employee`	["2009-01-01", "2010-05-28"]

- Extra rule to group annotations triples ($\vee$):

`:nuno :worksFor :DERI`	["2008-05-01", "2010-01-01"]
`:nuno :worksFor :DERI`	["2009-01-01", "2010-05-28"]
$\Rightarrow$ `:nuno :worksFor :DERI`	["2008-05-01", "2010-05-28"]

Annotated RDFS Inference rules

Inference rules are **independent** of the annotation domain

Annotated RDFS "rdfs:domain" rule:

`?SomeProp rdfs:domain ?SomeClass`	$?v1$
`?x ?SomeProp ?y`	$?v2$
$\Rightarrow$ `?x rdf:type ?SomeClass`	$?v1 \wedge ?v2$

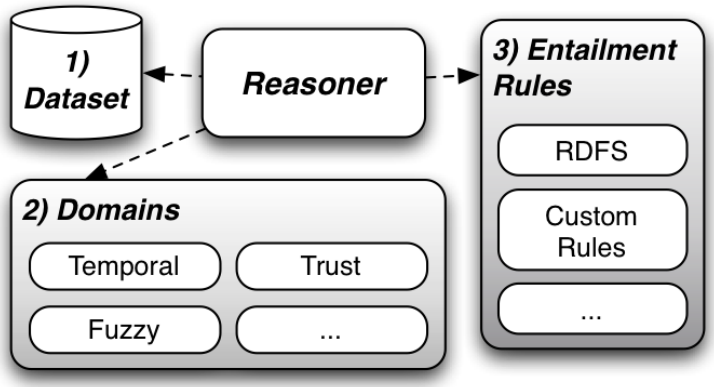
Example:

`:worksFor rdfs:domain :Employee`	$[-\infty, +\infty]$
`:nuno :worksFor :DERI`	["2008-05-01", "2010-05-28"]
$\Rightarrow$ `:nuno rdf:type :Employee`	["2008-05-01", "2010-05-28"]

- Extra rule to group annotations triples ($\vee$):

`:nuno :worksFor :DERI`	["2008-05-01", "2010-01-01"]
`:nuno :worksFor :DERI`	["2009-01-01", "2010-05-28"]
$\Rightarrow$ `:nuno :worksFor :DERI`	["2008-05-01", "2010-05-28"]

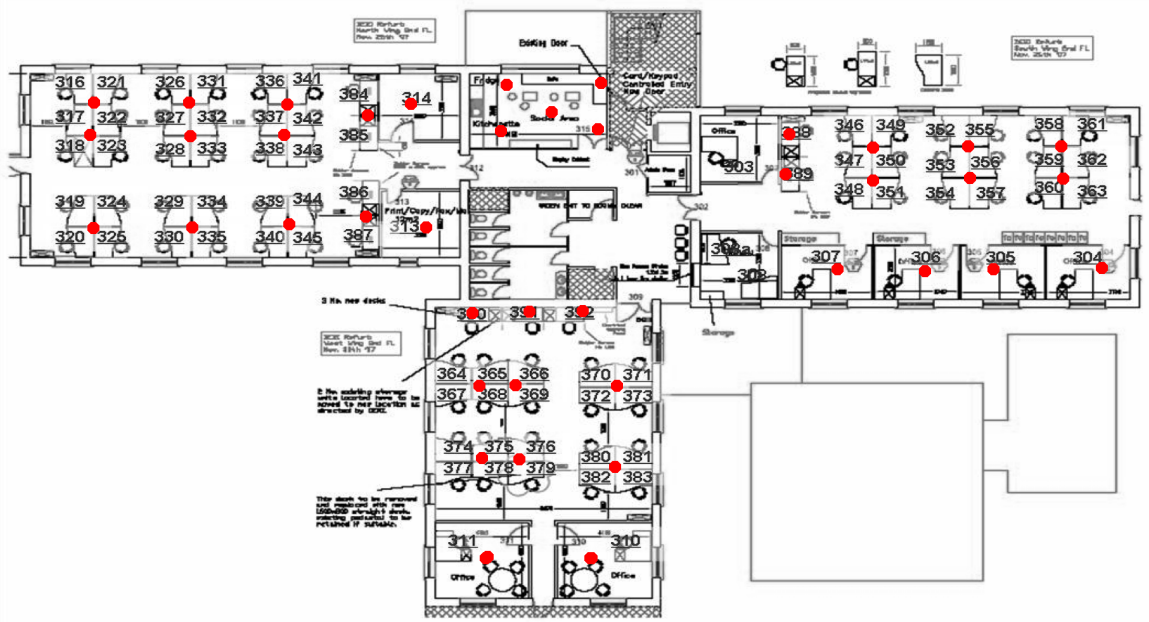
Implementation



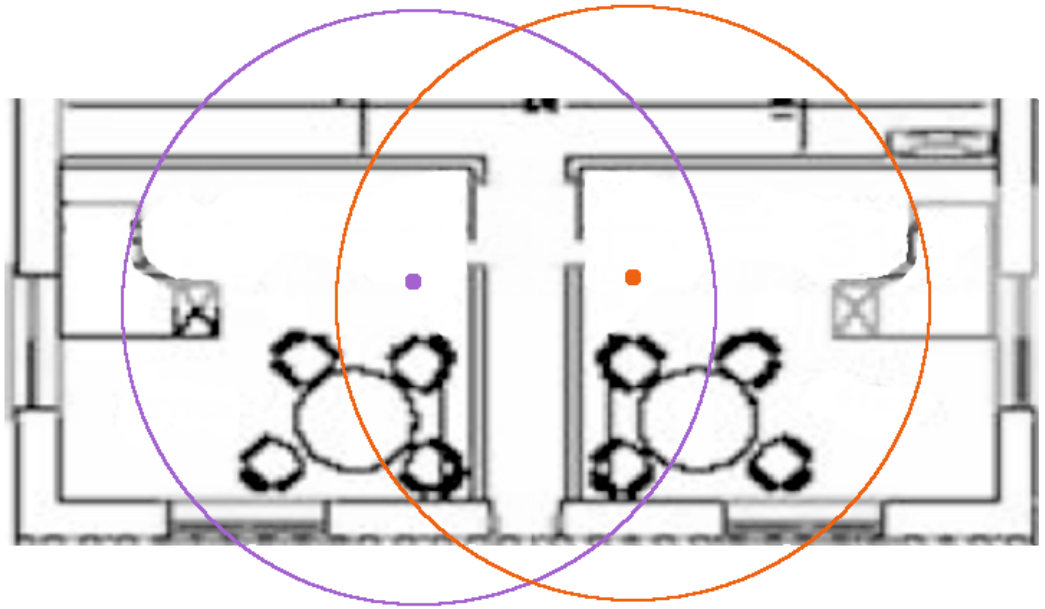
- Prototype implementation that computes the closure of an annotated RDF graph
- Modular system: can use different domains and rulesets

More info and downloads available at:
http://sprinql.deri.ie

Sensor data



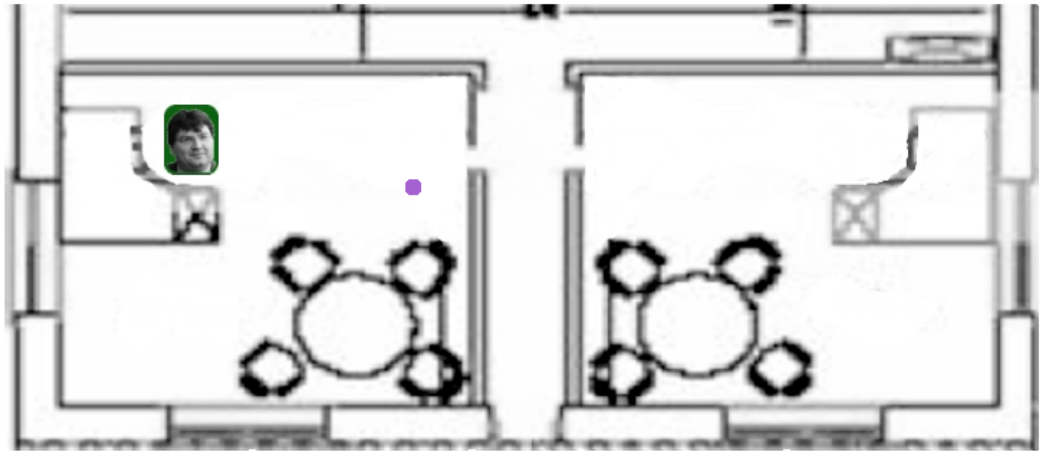
Sensor data



Each sensor record is composed of `timestamp`, `ip`, `tag` and `ssi`.

```
2010-05-28 14:57:51   10.254.2.15   4302   83
2010-05-28 14:57:51   10.254.3.1    4302   83
2010-05-28 14:57:51   10.254.2.6    4302   83
```

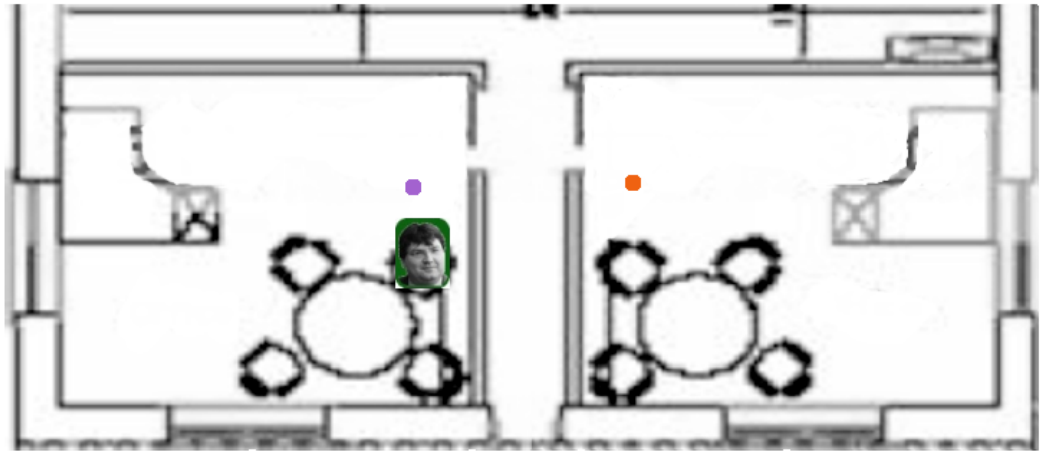
Sensor data



Time = 14:34

```
:tag4302 :locatedIn ":room311" (["13:58", "14:34"],0.9)
```

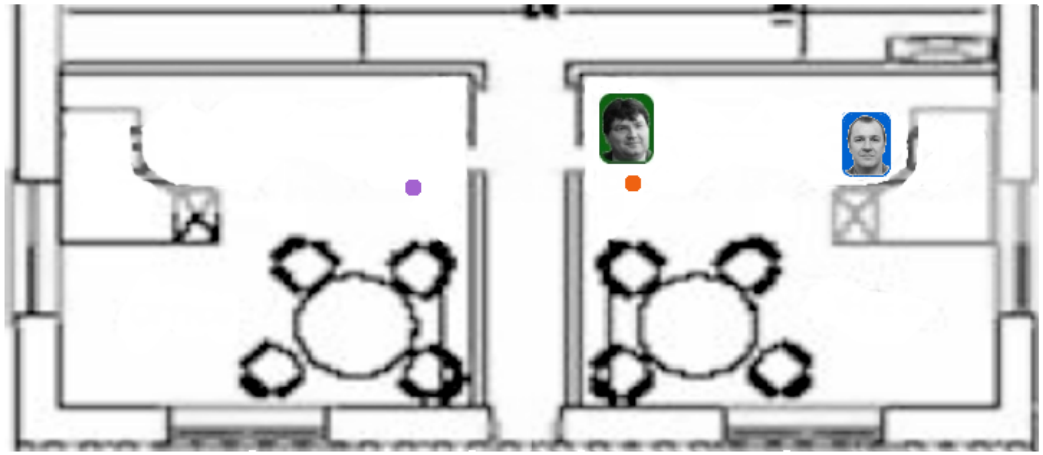
Sensor data



Time = 14:40

```
:tag4302 :locatedIn ":room311" (["13:58", "14:34"],0.9)
:tag4302 :locatedIn ":room311, :room310" (["14:35", "14:52"],0.6)
```

Sensor data



Time = 14:56

```
:tag4302 :locatedIn ":room311" (["13:58","14:34"], 0.9)
:tag4302 :locatedIn ":room311, :room310" (["14:35","14:52"], 0.6)
:tag4302 :locatedIn ":room311, :room310" (["14:53","14:56"], 0.6)
:tag4145 :locatedIn ":room310" (["14:40","14:56"], 0.9)
```

Data processing

- Sensor data for 1 hour approx. 434,000 records.

Data processing

- Sensor data for 1 hour approx. 434,000 records.

1. Group all the `ips` (with the lowest `ssi`) for a given `timestamp` and `tag`;

Data processing

- Sensor data for 1 hour approx. 434,000 records.

1. Group all the `ips` (with the lowest `ssi`) for a given `timestamp` and `tag`;
2. Merge all records that have consecutive `timestamp` and equal `tag` and `ip` into a single interval;

Data processing

- Sensor data for 1 hour approx. 434,000 records.

1. Group all the `ips` (with the lowest `ssi`) for a given `timestamp` and `tag`;
2. Merge all records that have consecutive `timestamp` and equal `tag` and `ip` into a single interval;
3. Compute the trust value of each merged record based on the average of the `ssi` ;

Data processing

- Sensor data for 1 hour approx. 434,000 records.

1. Group all the `ips` (with the lowest `ssi`) for a given `timestamp` and `tag`;
2. Merge all records that have consecutive `timestamp` and equal `tag` and `ip` into a single interval;
3. Compute the trust value of each merged record based on the average of the `ssi` ;
4. Based on the trust, we can discard the triples below a certain threshold:
   - threshold of 0.1 results in approx. 70,000 triples
   - threshold of 0.4 results in approx. 53,000 triples

Annotated SPARQL : SPRINQL*

Extend SPARQL to allow querying annotated RDF

- "Annotation aware" SPARQL

* SPaRql Inspired aNnotation Query Language

Annotated SPARQL : SPRINQL*

Extend SPARQL to allow querying annotated RDF

- "Annotation aware" SPARQL

  "Where was Stefan before this institute meeting?"

```
SELECT ?Room WHERE {
      ?Tag     :assignedTo   :stefan ;
               :locatedIn    ?Room . ["14:30", "15:00"]
}
```

* SPaRql Inspired aNnotation Query Language

Annotated SPARQL : SPRINQL*

Extend SPARQL to allow querying annotated RDF

- "Annotation aware" SPARQL

  "Where was Stefan before this institute meeting?"

```
SELECT ?Room WHERE {
      ?Tag     :assignedTo   :stefan ;
               :locatedIn    ?Room . ["14:30", "15:00"]
}
```

- Evaluation based on an extension of the SPARQL relational algebra to support annotations

* SPaRql Inspired aNnotation Query Language

Annotated SPARQL

"When were Stefan and Manfred in the same room?"

```
SELECT ?Room ?TimeInterval WHERE {
      ?Tag1     :assignedTo   :stefan ;
                :locatedIn    ?Room . ?TimeInterval
      ?Tag2     :assignedTo   :manfred ;
                :locatedIn    ?Room . ?TimeInterval
}
```

Annotated SPARQL

"When were Stefan and Manfred in the same room?"

```
SELECT ?Room ?TimeInterval WHERE {
      ?Tag1     :assignedTo   :stefan ;
                :locatedIn    ?Room . ?TimeInterval
      ?Tag2     :assignedTo   :manfred ;
                :locatedIn    ?Room . ?TimeInterval
}
```

Answers:

```
(?Room, ?TimeInterval) = (:room311, {["09:13", "10:35"],
                                     ["11:23", "12:47"]})

(?Room, ?TimeInterval) = (:conferenceRoom, {["14:25", "14:57"]})
```

Conclusions

- Concise way to attach information to triples
- Inference support over the annotations

Future work:

- Working on Annotated SPARQL
- Define other annotation domains (spatial, ...)
- Annotated DBPedia

Conclusions

- Concise way to attach information to triples
- Inference support over the annotations

Future work:

- Working on Annotated SPARQL
- Define other annotation domains (spatial, ...)
- Annotated DBPedia

- "A general framework for representing and reasoning with annotated semantic web data". AAAI 2010
- Position paper accepted for presentation at W3C RDF Next Steps Workshop

Conclusions

- Concise way to attach information to triples
- Inference support over the annotations

Future work:

- Working on Annotated SPARQL
- Define other annotation domains (spatial, ...)
- Annotated DBPedia

- "A general framework for representing and reasoning with annotated semantic web data". AAAI 2010
- Position paper accepted for presentation at W3C RDF Next Steps Workshop

Other possible uses?