Deep integration of Python with Semantic Web technologies

Marian Babik, Ladislav Hluchy

Intelligent and Knowledge Technologies Group

Institute of Informatics, SAS

Goals of the presentation

- Brief introduction of Semantic Web technologies
- Introduce deep integration ideas
- Show initial implementation of the Python integrated with OWL
- Discuss architecture

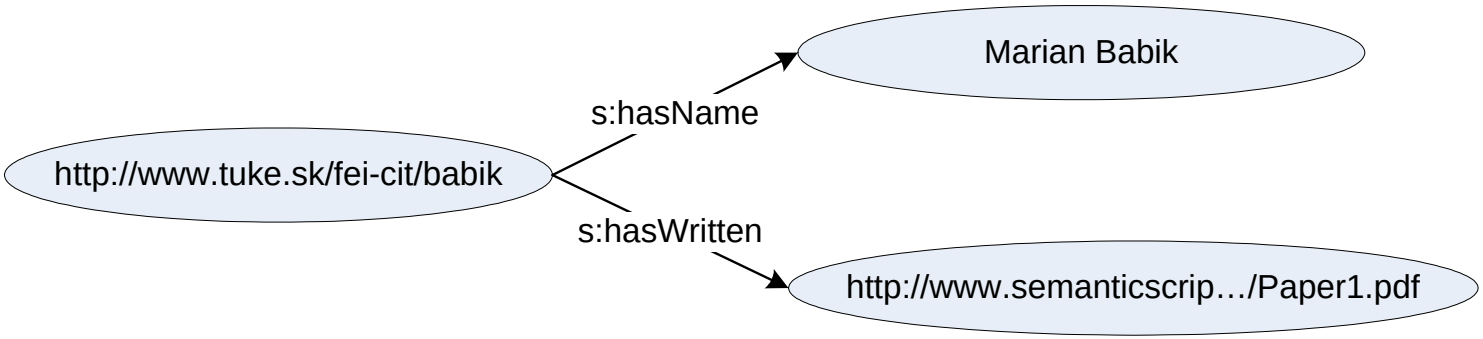
Semantic Web

- Extension of the current Web, providing infrastructure for the integration of data on the Web
  - Data should be available for further processing
  - It should be possible to combine and merge data on a Web scale
  - Metadata, reasoning
- Making data machine processable:
  - Names for resources
  - Common data model: RDF
  - Access to the data: SPARQL, OWL-QL
  - Define vocabularies: RDFS, OWL, SKOS
  - Reasoning: OWL, Rules

Resource Description Framework (RDF)

- Based on triples (subject, predicate, object)
  - Labelled connection btw. two resources (subject, object)
  - Subject, predicate are URI-s; Object is URI (e.g. Subject) or Literal
  - (http://www.marianbabik.org, http://.../myCalendar, http://.../calendar)
- (subject, predicate, object) can be seen as a labeled edge in a graph
- Serialization XML, Turtle, N3, etc.

RDF example



XML/RDF

```
<rdf:Description
   rdf:about="http://www.tuke.sk/fei-cit/babik">
<s:hasName>Marian Babik</s:hasName>
<s:hasWritten
    rdf:resource="http://www.semanticscripting.org/SFSW2006/Paper1.pdf"/>
</rdf:Description>
```

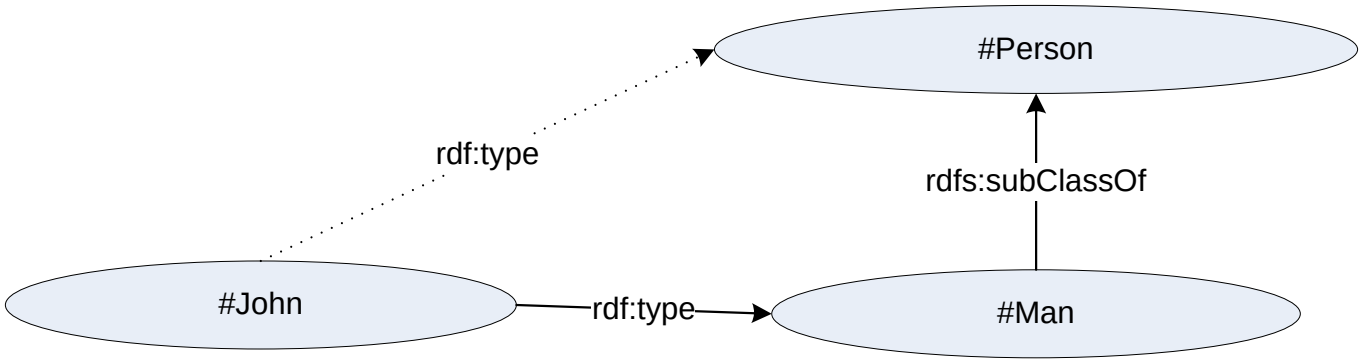
Turtle (N3)

```
<http://www.tuke.sk/fei-cit/babik>
   s:hasName  “Marian Babik”;
   s:hasWritten <http://www.semantic scripting.org/SFSW2006/Paper1.pdf> .
```

RDF Vocabulary Description Language (RDFS)

- RDF lacks vocabulary with semantic meaning
- RDFS defines resources and classes and relationships among classes/resource
  - rdf:type, rdfs:Class, rdfs:subClassOf
  - rdfs:subPropertyOf,
  - Resource may belong to several classes (range, domain)
- Special kind of type system

RDFS example




```
:Person rdfs:subClassOf :Man .

:John rdf:type :Man .
```

```
<rdf:Description rdf:ID="John">
        <rdf:type rdf:resource="Man"/>
</rdf:Description>
<rdf:Description rdf:ID="Man">
        <rdfs:subClassOf
rdf:resource="Person"/>
</rdf:Description>
```

RDFS example (2)

```
:Person rdfs:subClassOf :Man .
:John rdf:type :Man .
:isProgrammer
  rdf:type rdf:Property;
  rdf:domain  :Person;
  rdf:range     rdfs:Literal .
```

Web Ontology Language (OWL)

- RDFS doesn't solve all the issues
  - Reasoning about terms
  - Behavior of properties (symmetric, transitive, etc.)
  - Cardinality constraints
  - Equivalence of classes
  - Enumeration (oneOf), union of classes
  - Datatype properties
- Compromise btw
  - Rich semantics
  - Feasibility, implementability
- Three layers:
  - OWL-Lite,
  - OWL-DL (based on Description Logic – SHOIN(D))
  - OWL-Full (might be undecidable)

OWL-DL

- Class definition, axioms
  - Complex class definitions based on the class descriptions
- Property definition, axioms
  - RDF Schema constructs (rdfs:subPropertyOf, rdfs:domain and rdfs:range)
  - relations to other properties: owl:equivalentProperty and owl:inverseOf)
  - global cardinality constraints: owl:FunctionalProperty and owl:InverseFunctionalProperty
  - logical property characteristics owl:SymmetricProperty and owl:TransitiveProperty
- Individuals
  - facts about class membership and property values of individuals

OWL-DL example

```
@prefix people:  <http://sample.org/people> .

people:Person a owl:Class .
people:Man a owl:Class; rdfs:subClassOf people:Person
people:hasChild a owl:Property .
people:hasSon a owl:Propery; rdfs:subPropertyOf people:hasChild

PersonWithSingleSon a owl:Class ; rdfs:subClassOf
  [ a owl:Restriction ; owl:cardinality "1"^^ ;
     owl:onProperty :hasSon ];
rdfs:subClassOf
  [ a owl:Restriction ; owl:cardinality "1"^^ ;
     owl:onProperty :hasChild ] .

People:John a people:Person .
People:Bob a people:PersonWithSingleSon .
```

SW for developers

Classes are types for instances	Classes are sets of individuals
Each instance has one class (as its type). Classes don't share instances	Each individual can belong to multiple classes
List of classes is known at compile time	Classes can be created and changed at runtime
Compilers are used	Reasoners are used for classification and consistency
Properties are defined locally to a class	Properties are standalone entities (can exist without classes)
Instances have values for attached properties. Values must be correctly typed. Range constraints are used for type checking	Instance can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.
Closed world assumption	Open world assumption
Class behavior through functions and methods	Classes make their meaning explicit in terms of OWL statements.

Source: www.w3.org/TR/sw-oosd-primer

Deep integration ideas

- Import OWL classes alongside classes defined normally (native API)
- Support for intensional definition of classes/properties (OWL statements)
- Separation of concerns among declarative and procedural aspects
- Similar to what SQLObject does for databases

Benefits

Python

Ontologies

- Definition of data and domain using Semantic Web tools (Protégé, SWOOP)
- Native OWL API
- OWL Inference and Web sharing of concepts
- New programming paradigm for Python
- Native interface for ontologies, developers can work directly with OWL classes and their instances
- Existing Python Web frameworks and access to large set of libraries

OWL APIs

- Jena, OWL API
  - Different notion of polymorphism means APIs have to introduce sophisticated design patterns

```
Resource r = myModel.getResource( myNS + "DigitalCamera" );
OntClass cls = (OntClass) r.as( OntClass.class );
Restriction rest = (Restriction) cls.as( Restriction.class );
```

RDF/OWL APIs

Python

- RDFLib – statement-centric (Daniel Krech)
- CWM – model centric
- Pychinko – model, statement-centric, rule based
- MetaLog – statement, resource-centric, based on prolog
- Sparta, Tramp – resource centric

Others

- Active RDF (Ruby) – native mapping, RDF, integrated with Rails
- RDFHomepage (PHP) – native mapping

OWL-DL and Python

- Ontology corresponds to Python module
- OWL Class corresponds to Python Class
- Instance of OWL Class corresponds to Python Object
- OWL Property corresponds to Python Class or Python method

Classes

- OWL Class corresponds to Python Class
- Instances of OWL Class correspond to Python Objects
- Class definition
  - Each python class has two attributes:
    - URI (owl:Class URI)
    - definedBy (intensional definition)

```
>>> from seth import Thing, Property
>>> Person = Thing.new(‘ :Person a owl:Class .')
>>> print Person
<class seth.Meta.Person>

>>> Man = Thing.new(' :Man a owl:Class ; rdfs:subClassOf :Person .')
```

Classes (2)

```
>>> PersonWithSingleSon = Thing.new("""PersonWithSingleSon
    a owl:Class ;
rdfs:subClassOf
  [ a owl:Restriction ; owl:cardinality "1"^^ ;
    owl:onProperty :hasSon ];
rdfs:subClassOf
  [ a owl:Restriction ; owl:cardinality "1"^^ ;
    owl:onProperty :hasChild ] .""")
```

```
>>> model = OntModel()
>>> model.bind("people","http://somewhere.net/people#")
>>> Thing.setOntModel(model)
>>> Person = Thing.new(" people:Person a owl:Class .")
```

Properties

- owl:Property corresponds to special kind of python object
  - Similar to Class definition
  - But such classes don't have instances

```
>>> hasChild = Property.new('hasChild a owl:ObjectProperty .')
>>> print hasChild
<class 'seth.Meta.hasChild'>
>>> hasSon = Property.new('hasSon a owl:ObjectProperty ;
rdfs:subPropertyOf :hasChild .')
```

- owl:Property corresponds to python object attribute (method)

```
>>> John.hasChild(Bob)
```

Individuals

- Individual is python object (class instance)

```
>>> Bob = PersonWithSingleSon(‘Bob’)
>>> John = Person('John‘)
>>> print John
<seth.Meta.Person object at 0xb7a29e4c>
>>> hasChild(Bob, John)
```

Ontologies

- owl:Ontology corresponds to Python's module

```
>>> ModuleFactory("people", "http://somewhere.net/people.owl")
>>> import people
>>> print people.Person
<seth.people.Person
:Man a owl:Class ; rdfs:subClassOf :Person .
>>> print people.John
<seth.people.Person object at 0xb7a29e4c>
```

- serialization

```
>>> people.serialize("/tmp/test.owl", "RDF/XML-ABBREV")
```

Queries

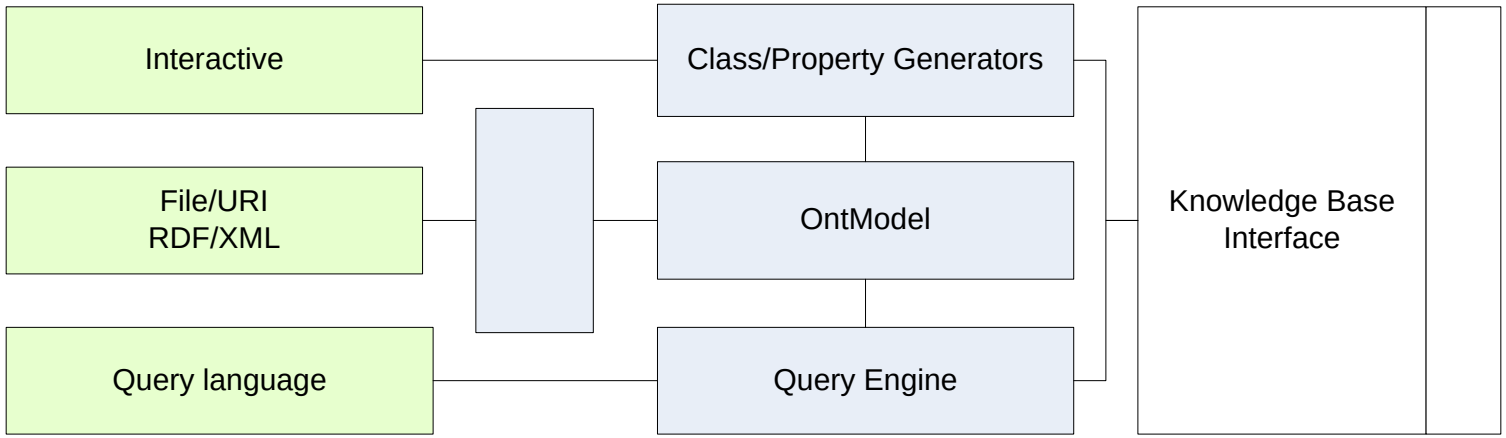
- Triple like queries

```
>>> for individual in Person.findInstances():
...     print individual, individual.name
<seth.Meta.Man object at 0xb7d0b64c> Peter
<seth.Meta.Person object at 0xb7d0b50c> John
<seth.Meta.Person object at 0xb7d0b6ec> Jane
>>> for who in hasSon.q(Bob):
...     who.name
'John'
>>> print hasSon.query(Bob, John)
1
```

- OWL-QL, SPARQL
- Native queries

Architecture

- Interaction with Java-based libraries (reasoners)



Python and Java

- Jython - Python implemented for the Java Virtual Machine
- JPE(Java-Python Extension) - uses JNI to provide a bridging mechanism between Java and a Python interpreter
- JPype - interfacing at the native level in both Virtual Machines
- SPIRO - uses ORB (Object-Request Broker) technology
- GCJ

Issues

- Performance (conversion between JVMs)
- Cache
- Datatype properties
- Open world semantics
- Debugging
- Rules

Summary

- SETH Homepage:
  http://seth-scripting.sourceforge.net
- Available through CVS under MIT license
- Discussion, support mailing list:
  seth.ui@savba.sk

- Contact:
  - Marian.Babik@gmail.com

References

- Introduction to Semantic Web
  - http://www.w3.org/2006/Talks/0524-Edinburgh-IH/Overview.pdf
- Deep Integration
  - http://www.semanticscripting.org/SFSW2005/papers/Vrandecic-Deep_Integration.pdf
  - http://gigaton.thoughtworks.net/~ofernand1/DeepIntegration.pdf
  - http://www.semanticscripting.org/SFSW2006/Paper1.pdf