1 Managing Semantic Data

1.1 Overview

Current Trends for Semantic Technologies

- Linked Data
- data quality and data provenance
- shallow semantics
- data validation
- Business Intelligence
- IoT and RDF streaming
- Knowledge graphs

1.2 Semantic Data Pipelines

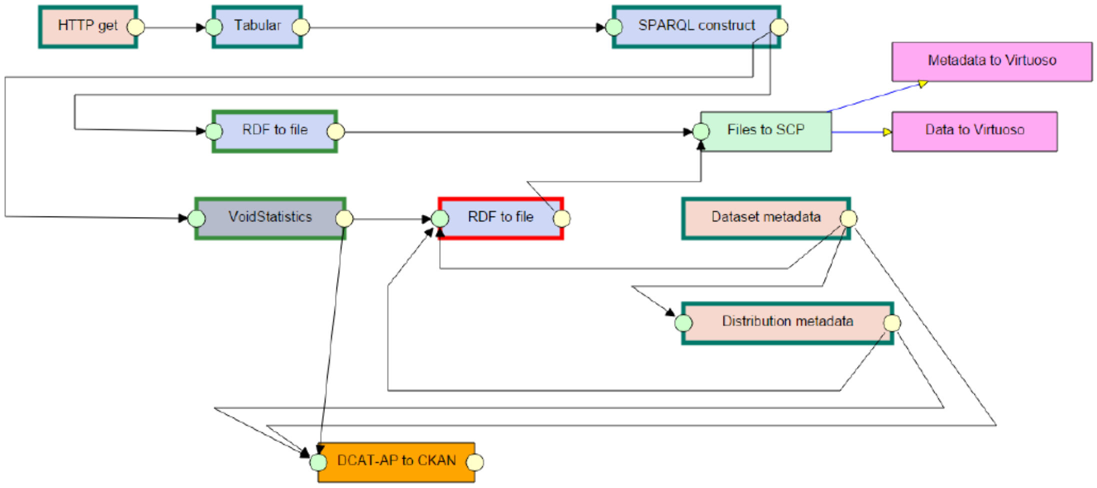
Semantic Data Pipelines

- semantic technologies are widely used for data processing activities that includes heterogeneous data sources
- RDF within data pipelines is used for
  - annotation of data sources
  - representation/interpretation of data schema
  - providing common format for representation of data

LinkedPipes ETL

- web-based lightweight ETL tool
- used primarily for processing of Open Data and publication of Linked Data
- components are written in Java + Javascript UI
- most of the components transforms to/from RDF
- configuration of transformation pipelines is in RDF

LinkedPipes ETL Example Pipeline



ETL pipeline execution from [**klimek2016linkedpipes**]. Arrows represent flow of data between components, green/red edges of rectangles represent successful/unsuccessful execution of components.

SPARQLMotion

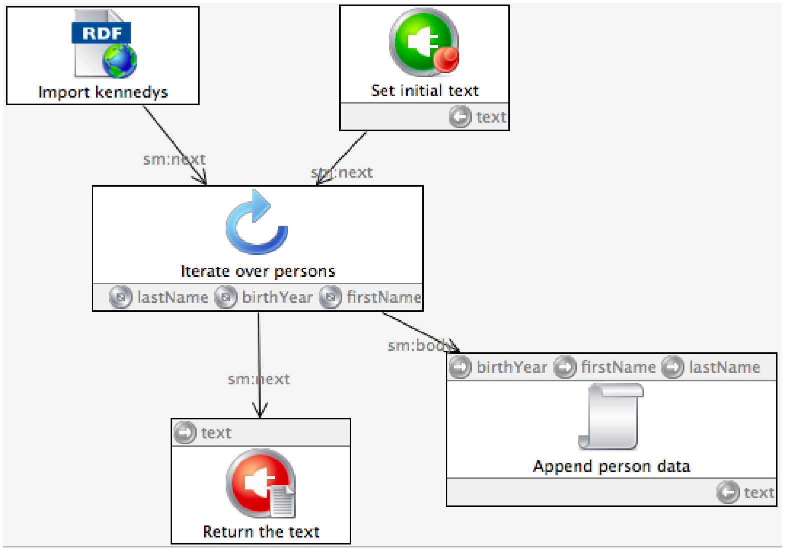
- an RDF-based scripting language with a graphical notation to describe data processing pipelines
- pipelines editable by commercial version TopBraid Composer [1]
- provides extensions through RDF, Java, Javascript
- only SPARQL variable bindings and a RDF graph are passed between nodes of pipeline
- whole configuration is in RDF
- well integrated with SPIN[2], SHACL[3]

SPARQLMotion Script Example

[1] https://www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition/

[2] SPARQL Inferencing Notation – RDF-based vocabulary to represent SPARQL rules and constraints on RDF models

[3] Shapes Constraint Language that is regarded as successor of SPIN

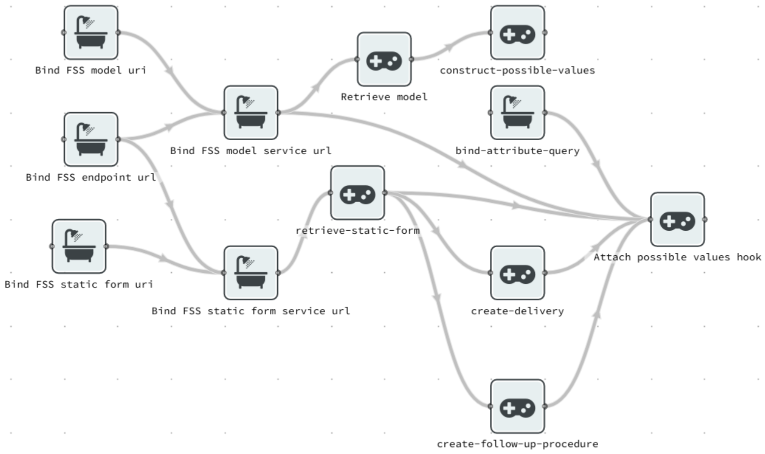


Visual representation of SPARQLMotion script from tutorial at `http://sparqlmotion.org/`. Rectangles represent modules while arrows represent order of execution/flow of data. Upper/lower part of rectangles shows input/output SPARQL variables of modules.

SPipes

- SPipes language – an extension of SPARQLMotion scripting language
- SPipes engine
  - command-line and REST interface
  - only partial support for SPARQLMotion language (e.g. no iteration over set of pipeline nodes)
  - semantic logging of execution
- SPipes components
  - most of the modules from SPARQLMotion core libraries are missing
  - new modules for generation of semantic forms, Linked Data processing and publishing, processing tabular data, NLP
- SPipes editor
  - web based editor/debugger of SPipes scripts
  - first release within next month

SPipes Editor

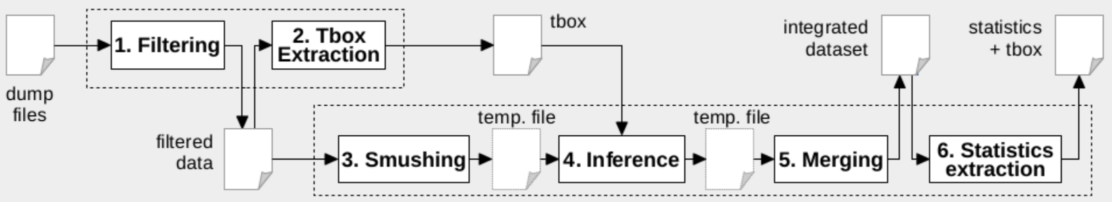

Bind FSS model uri
Bind FSS endpoint url
Bind FSS static form uri
Bind FSS model service url
Bind FSS static form service url
Retrieve model
retrieve-static-form
construct-possible-values
bind-attribute-query
create-delivery
create-follow-up-procedure
Attach possible values hook

Visual representation of SPARQLMotion script in SPipes Editor Prototype

RDFpro

- Java command-line tool and library for RDF processing
- stream-oriented highly optimized RDF processors
- primarily for Linked Data integration tasks at large scale
- Javascript and Groovy scripting support
- support for RDF quads filtering and replacement, SPARQL-like inference rules, TBox/VOID statistics extraction, *owl:sameAs* smushing, RDF deduplication, set/-multiset operations ...

RDFpro Pipeline Example



Sketch of a RDFpro pipeline [**corcoglioniti2014rdf**] – integrating RDF data from Freebase, GeoNames and DBpedia in the four languaged EN, ES, IT and NL, performing smushing, inference, deduplication and statistics extraction.

```
rdfpro @read smushed.tql.gz \
       @rdfs -c '<graph:vocab>' -e rdfs4a,rdfs4b,rdfs8 -d tbox.tql.gz \
       @transform '-o owl:Thing schema:Thing foaf:Document bibo:* con:* -p dc:subject foaf:page dct:relation bibo:* con:*' \
       @write inferred.tql.gz
```

Configuration of 4.Inference – a deductive closure of data is computed and saved, using the extracted TBox and excluding RDFS rules rdfs4a, rdfs4b and rdfs8 to avoid inferring uninformative X rdf:type rdfs:Resource quads. Output of the closed TBox is filtered (@transform) and placed in graph <graph:vocab>. For details see `http://rdfpro.fbk.eu/example.html`.

1.3 Data Validation

Data validation languages

- Shape based
  - ShEx
  - SHACL
- SPIN
- OWL integrity constraints[**sirin2010data**]

Shapes Constraint Language (SHACL)

- RDF vocabulary for validating RDF graphs against a set of conditions (called *shapes*)
- W3C Recommendation[4] from July 20, 2017
- partial support for inference (`rdf:type`, `rdf:Class`, `rdfs:subClassOf`, `owl:imports`)
- optional support for entailment (`sh:entailment`)
- support for closed shapes
- modular and reusable (`owl:import`, composition/inheritance of shapes)
- support validation report, fixes for constraint violations
- constraints extensions (SHACL-SPARQL, Javascript ...)

SHACL resources

- SHACL playground – `http://shacl.org/playground/`
- SHACL by example – `https://www.slideshare.net/jelabra/shacl-by-example`
- Reusable SHACL constrains, data model for test cases and fixes for constraint violations – `http://datashapes.org/`
- SHACL shapes for schema.org – `http://datashapes.org/schema.ttl`

[4] `https://www.w3.org/TR/shacl/`

1.4 Integration into Programming Languages

Integration of Data Models into Programming Languages

There are three major ways to integrate using

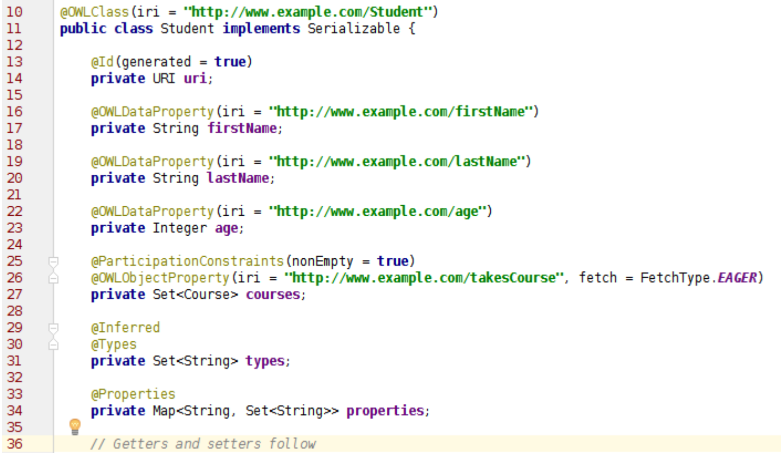
- generic types (e.g. Jena)
- mapping to the type system of a programming language (e.g. JOPA)
- using custom type system (e.g. $\lambda - DL$)

JOPA

JOPA[5] is a persistence API and implementation for accessing OWL ontologies with features:

- Object-ontological mapping based on integrity constraints,
- Explicit access to inferred knowledge,
- Access to unmapped properties and individual's types
- Transactions
- Separate storage access layer

JOPA Entity Example



```java
@OWLClass(iri = "http://www.example.com/Student")
public class Student implements Serializable {

    @Id(generated = true)
    private URI uri;

    @OWLDataProperty(iri = "http://www.example.com/firstName")
    private String firstName;

    @OWLDataProperty(iri = "http://www.example.com/lastName")
    private String lastName;

    @OWLDataProperty(iri = "http://www.example.com/age")
    private Integer age;

    @ParticipationConstraints(nonEmpty = true)
    @OWLObjectProperty(iri = "http://www.example.com/takesCourse", fetch = FetchType.EAGER)
    private Set<Course> courses;

    @Inferred
    @Types
    private Set<String> types;

    @Properties
    private Map<String, Set<String>> properties;

    // Getters and setters follow
```

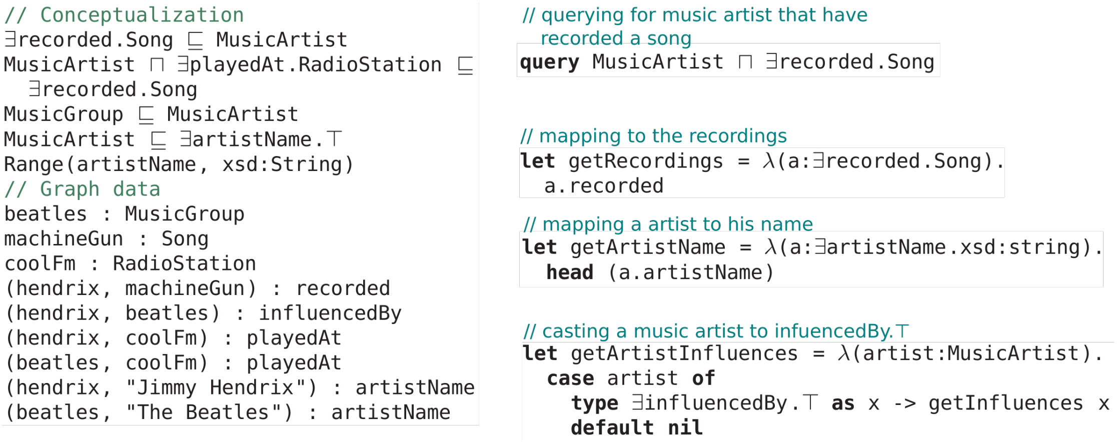
Object-ontological mapping from JOPA library [corcoglioniti2014rdf]

[5] https://kbss.felk.cvut.cz/web/kbss/jopa

$\lambda - DL$

- $\lambda - DL$[6] is a typed lambda calculus for semantic data
- Features
  - DL concepts as types
  - subtype inferences
  - typing of queries
  - DL-safe queries
  - open-world querying
- Prototype implementation in `F#` using OWL reasoner *HermiT*

Example of $\lambda - DL$ program and data in abstract syntax



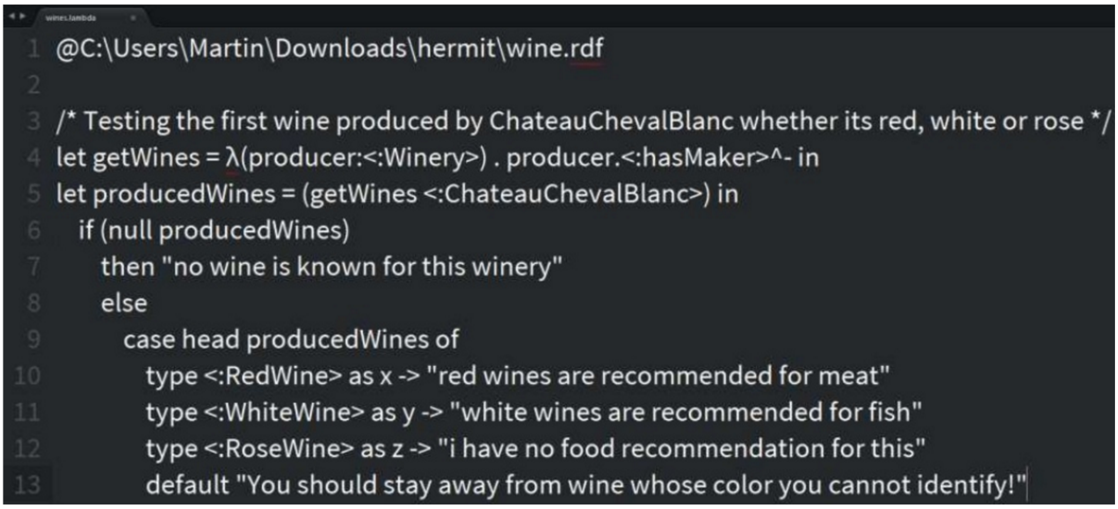
```
// Conceptualization
∃recorded.Song ⊑ MusicArtist
MusicArtist ⊓ ∃playedAt.RadioStation ⊑
  ∃recorded.Song
MusicGroup ⊑ MusicArtist
MusicArtist ⊑ ∃artistName.⊤
Range(artistName, xsd:String)
// Graph data
beatles : MusicGroup
machineGun : Song
coolFm : RadioStation
(hendrix, machineGun) : recorded
(hendrix, beatles) : influencedBy
(hendrix, coolFm) : playedAt
(beatles, coolFm) : playedAt
(hendrix, "Jimmy Hendrix") : artistName
(beatles, "The Beatles") : artistName
```

```
// querying for music artist that have
   recorded a song
query MusicArtist ⊓ ∃recorded.Song

// mapping to the recordings
let getRecordings = λ(a:∃recorded.Song).
  a.recorded

// mapping a artist to his name
let getArtistName = λ(a:∃artistName.xsd:string).
  head (a.artistName)

// casting a music artist to infuencedBy.⊤
let getArtistInfluences = λ(artist:MusicArtist).
  case artist of
    type ∃influencedBy.⊤ as x -> getInfluences x
    default nil
```

For detailed explanation of the example see [**leinberger2016lambdadl**].

Example of $\lambda - DL$ program in F#

[6] `https://west.uni-koblenz.de/de/lambda-dl`



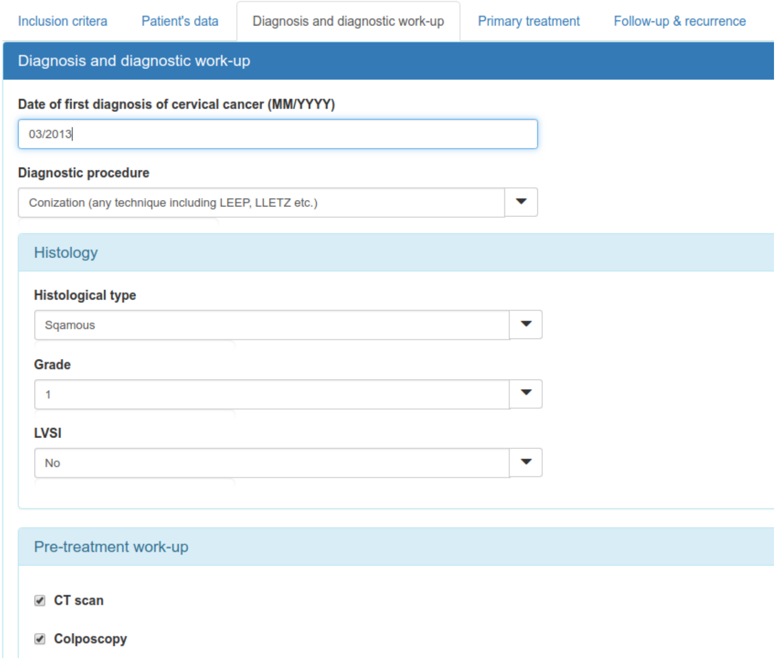
```
@C:\Users\Martin\Downloads\hermit\wine.rdf

/* Testing the first wine produced by ChateauChevalBlanc whether its red, white or rose */
let getWines = λ(producer:<:Winery>) . producer.<:hasMaker>^- in
let producedWines = (getWines <:ChateauChevalBlanc>) in
  if (null producedWines)
    then "no wine is known for this winery"
    else
      case head producedWines of
        type <:RedWine> as x -> "red wines are recommended for meat"
        type <:WhiteWine> as y -> "white wines are recommended for fish"
        type <:RoseWine> as z -> "i have no food recommendation for this"
        default "You should stay away from wine whose color you cannot identify!"
```

1.5 Ontology-based form generation

SForms

SForms is a JavaScript library for ontology-based interactive web forms. Forms are defined by JSON-LD ontology using predefined RDF vocabulary.

SForms UI



Diagnosis and diagnostic work-up

Date of first diagnosis of cervical cancer (MM/YYYY)
03/2013

Diagnostic procedure
Conization (any technique including LEEP, LLETZ etc.)

Histology

Histological type
Sqamous

Grade
1

LVSI
No

Pre-treatment work-up

☑ CT scan

☑ Colposcopy

SForms – dynamically generated web forms