Ontology Design: Semantic data transformations

Aldo Gangemi, Andrea G. Nuzzolese
Lecture 3 @ Corso Dottorato 2011
Dipartimento di Scienze dell’Informazione, Bologna, Italy
Outline

• Introduction
• DB vs Linked data
• From any data to the Semantic Web
• * From thesauri to ontologies
• Semion method and tool
• Stanbol rules
• Appendix: SPARQL Primer
Why do I need transformations?

• A scenario
  – my system fetches knowledge from different sources in LOD
  – each of these sources uses its own ontology/vocabulary
Why do I need transformations?

• A scenario
  – my system fetches knowledge from different sources in LOD
  – each of these sources uses its own ontology/vocabulary

How to gather a homogeneous representation of knowledge expressed with heterogeneous vocabularies?
the Web of Data is fed by “triplifiers”, tools able to transform content to Linked Data

triplifiers implement various methods typically based on bulk recipes which allow for no or limited customization of the process

lack of good practices for knowledge representation and organization

the transformation relies on predetermined implicit assumptions on the domain semantics of the non-RDF data source
An example

- dbpedia:Bob_Marley
  - foaf:name: "Bob Marley"
  - rdf:type: dbpedia:Person

- skos:Concept
  - rdf:type: dbpedia:Bob_Marley
    - skos:prefLabel: "Marley, Bob"
  - rdf:type: nyt:65169961111056171853

- nyt:65169961111056171853
  - skos:prefLabel: "Bob Marley"
An example

I want to aggregate the two graphs
Another example

```
skos: Concept

"Marley, Bob"

rdf:type

skos:prefLabel

nyt:
65169961111056171853
```
Another example

I want to use a different vocabulary!
I want to use FOAF!
Another example

I want to use a different vocabulary!
I want to use FOAF!
DB data vs. Linked data?
A DB stored data and answers queries

- Aldo is 48
- Aldo works in Rome
- Aldo makes research on Semantic Web

- STLab has been founded in 2008
- STLab is in Italy
- STLab makes research on semantic technologies

<table>
<thead>
<tr>
<th>nome</th>
<th>data_di_nascita</th>
<th>luogo_di_lavoro</th>
<th>temi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aldo</td>
<td>08-16-962</td>
<td>Roma</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>Alfio</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>laboratori</th>
<th>data_di_fondazione</th>
<th>sede</th>
<th>temi</th>
</tr>
</thead>
<tbody>
<tr>
<td>STLab</td>
<td>2008</td>
<td>Italia</td>
<td>Tecnologie Semantiche</td>
</tr>
<tr>
<td>LOA</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>
Complex queries?

- Who is interested in “Semantic Web” and is working in the same country as STLab is located?

<table>
<thead>
<tr>
<th>nome</th>
<th>data di nascita</th>
<th>luogo di lavoro</th>
<th>temi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aldo</td>
<td>08-16-962</td>
<td>Roma</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>Alfio</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>laboratori</th>
<th>data_di_fondazione</th>
<th>sede</th>
<th>temi</th>
</tr>
</thead>
<tbody>
<tr>
<td>STLab</td>
<td>2008</td>
<td>Italia</td>
<td>Tecnologie Semantiche</td>
</tr>
<tr>
<td>LOA</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

workplace
Rome

same concept? → seat
Italy

same country?

Persons

Labs
Complex queries?

- Who is interested in “Semantic Web” and is working in the same country as STLab is located?

<table>
<thead>
<tr>
<th>nome</th>
<th>data di nascita</th>
<th>luogo_lavoro</th>
<th>temi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aldo</td>
<td>08-16-962</td>
<td>Roma</td>
<td>Semantico Web</td>
</tr>
<tr>
<td>Alfio</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>laboratori</th>
<th>data_di_fondazione</th>
<th>sede</th>
<th>temi</th>
</tr>
</thead>
<tbody>
<tr>
<td>STLab</td>
<td>2008</td>
<td>Italia</td>
<td>Tecnologie Semantico</td>
</tr>
<tr>
<td>LOA</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>

No answer
Is mapping enough?

If mapping is approximate?

If we add another DB, e.g. an anagragific one?

Persons

Labs

<table>
<thead>
<tr>
<th>nome</th>
<th>data di nascita</th>
<th>luogo_di_lavoro</th>
<th>temi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aldo</td>
<td>08-16-962</td>
<td>Roma</td>
<td>Semanti c Web</td>
</tr>
<tr>
<td>Alfio</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>laboratori</th>
<th>data_di_fondazione</th>
<th>sede</th>
<th>temi</th>
</tr>
</thead>
<tbody>
<tr>
<td>STLab</td>
<td>2008</td>
<td>Italia</td>
<td>Tecnologie Semantiche</td>
</tr>
<tr>
<td>LOA</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

mapped ...

And this one?

same country?

seat

Italy
Publishing DB data as RDF on the Web
Data linking and querying on the web of data

- Who is interested in “Semantic Web” and is working in the same country as STLab is located?
How to answer really tough queries?

- Who is interested in “Semantic Web” and is working in a country where the unemployment rate is lower than 5%?
From any data to the SW

a pragmatic view
Knowledge transformation issues

• Heterogeneity of knowledge hampers interoperability
  – *see next examples*

• Syntactic interoperability bottleneck (platform)
  – e.g. *rdb, eav, xml, text, prolog, N3*

• Syntactic interoperability bottleneck (data model, cf. Villazon-Terrazas)
  – e.g. *kos rdb with adjacency list, snowflake, path enumeration, flattened data models*

• Semantic interoperability bottleneck (logical level)
  – e.g. *rdb with: anagraphic, lexical, statistical, formal data*

• Semantic interoperability bottleneck (conceptual structure)
  – e.g. two different databases on a same topic

• Social, pragmatic interoperability bottleneck (privacy, sustainability, policy)
  – e.g. different requirements, organizational contexts, etc.
Dealing with web semantics: current state

• Much enthusiasm, a lot of nice, different ideas
• Much confusion and mutual misunderstanding between “scruffies” and “neats”
  – Pushing formal semantics beyond its limits (e.g. the “owl:sameAs” dispute)
  – Doing ad-hoc apps
  – Mixing up strings, classes, terms, concepts, topics, tags, etc.
  – Trivializing transformation from social to formal semantics (e.g. when translating a syntactic frame directly to an OWL construct)
  – Suppose we need to design a desire ontology: where to start from?
Varieties of knowledge formats

- Text (or other) documents
- Extracted document fragments: terms, NE, sentences, bags of words, relations, factoids, ...
- Logs
- Folksonomies: keywords, tag sets, directories, topic trees, subject indexes, ...
- Metadata and metadata schemas
- Microformats, infoboxes, rich snippets
- Lexica: dictionaries, wordnets, terminologies, nomenclatures, ...
- Knowledge organization systems: thesauri, classification schemes, ...
- Frames, semantic networks
- DB schemas, RDB, XML schemas and data, ...
- Embedded RDF
- RDF datasets
- (Computational) ontologies
Plena mujer, manzana carnal, luna caliente,  
espeso aroma de algas, lodo y luz machacados,  
qué oscura claridad se abre entre tus columnas?  
Qué antigua noche el hombre toca con sus sentidos?  
Ay, amar es un viaje con agua y con estrellas,  
con aire ahogado y bruscas tempestades de harina:  
amar es un combate de relámpagos  
y dos cuerpos por una sola miel derrotados.  
Beso a beso recorro tu pequeño infinito,  
tus márgenes, tus ríos, tus pueblos diminutos,  
y el fuego genital transformado en delicia  
corre por los delgados caminos de la sangre  
hasta precipitarse como un clavel nocturno,  
hasta ser y no ser sino un rayo en la sombra.

[Pablo Neruda, Cien sonetos de amor]
Spinoza's Psychological Theory

In Part III of his Ethics, "On the Origin and Nature of the Affects," Spinoza addresses two of the most serious challenges facing his thoroughgoing naturalism. First, he attempts to show that human beings ... desire, and Hobbes in his physiology uses the term to refer to the physical causes of human desire (Leviathan VI). So 'conatus' has both broad, physical and specifically human, psychological ...

Michael LeBuffe
http://plato.stanford.edu/entries/spinoza-psychological/

Plotinus

Plotinus (204/5 -- 270 C.E.), is generally regarded as the founder of Neoplatonism. He is one of the most influential philosophers in antiquity after Plato and Aristotle. The term 'Neoplatonism' ... desire. But in the highest life, the life of Intellect, where we find the highest form of desire, that desire is eternally satisfied by contemplation of the One through the entire array of Forms that ...

Lloyd Gerson
http://plato.stanford.edu/entries/plotinus/

Moral Motivation

In our everyday lives, we confront a host of moral issues. Once we have deliberated and formed judgments about what is right or wrong, good or bad, these judgments tend to have a marked hold on us ... desire or disposition of the individual herself. Second, moral properties not only motivate entirely on their own: they provide overriding motivation. Of course, their motivational power depends on an ...

Connie S. Rosati
http://plato.stanford.edu/entries/moral-motivation/

Well-Being

Well-being is most commonly used in philosophy to describe what is non-instrumentally or ultimately good for a person. The question of what well-being consists in is of independent interest, but it ... desire theories, or objective list theories. According to the view known as welfarism, well-being is the only value. Also important in ethics is the question of how a
Wikipedia ...

**Desire**

From Wikipedia, the free encyclopedia.

Desire is a strong wish or craving.

Desire may also refer to:

**Concepts**

- Desire (in Philosophy)
- Desire (psychoanalysis)
- Interpersonal attraction
- **Lust**, intense craving for self gratification
- Libido, sexual desire
- Greed, selfish pursuit of wealth, power, or possessions
- Want, in economics
- Preference, on which microeconomic theory is based
- Motivation, thought that leads to an action
- Tanha in Buddhist psychology, as described in the Four Noble Truths
- A concept in Lacanian psychoanalytic theory related to the Oedipus complex

**Music**

- *Desire (album)*, by Bob Dylan (1976)
- *Desire (Tuxedomoon album)*, an album by Tuxedomoon (1981)
- *Desire (Tom Scott album)*, (1982)
- *Desire (Pharoah Monch album)*, (2007)
- "Desire" (Andy Gibb song), (1980)
- "Desire" (U2 song), (1988)
- "Desire" (Eurovision song), a song by Claudia Pace (2000)
- "Desire" (Do As Infinity song), (2001)
- "Desire" (Ryan Adams song), (2002)
- "Desire" (Geri Halliwell song), (2005)
Linguistic dictionaries and thesauri

- **Oxford American Dictionary**
  
  desire |də'zɪə(r)| |də,zaɪə(r)| |di,zaɪə(r)| |dɪˌzaɪə|  
  noun  
  a strong feeling of wanting to have something or wishing for something to happen: [with infinitive] a desire to work in the dirt with your bare hands.  
  - strong sexual feeling or appetite: they were clinging together in fierce mutual desire.  
  verb [trans.]  
  strongly wish for or want (something): he never achieved the status he so desired | [as adj.] (desired) it failed to create the desired effect.  
  - want (someone) sexually: there had been a time, years ago, when he had desired her.  
  - archaic express a wish to (someone); request or entreat.  
  
  ORIGIN Middle English: from Old French desir (noun), desirer (verb), from Latin desiderare (see desiderate).

- **Thesaurus**
  
  desire  
  noun  
  1 a desire to see the world wish, want, aspiration, fancy, inclination, impulse; yearning, longing, craving, hankering, hunger; eagerness, enthusiasm, determination; informal yen, itch, jones.  
  2 his eyes glittered with desire lust, sexual attraction, passion, sensuality, sexuality; lasciviousness, lechery, salaciousness, libidinosness; informal the hots, raunchiness, hominess.  
  verb  
  1 they desired peace want, wish for, long for, yearm for, crave, hanker after, be desperate for, be bent on, covet, aspire to; fancy; informal have a yen for, have a jones for, yen for, hanker after/for.  
  2 she desired him be attracted to, lust after, burn for, be infatuated by; informal fancy, have the hots for, have a crush on, be mad about, be crazy about.
WordNets …
FrameNets ...

Frame Report (recent data)

Desiring

Definition:
An **Experiencer** desires that an **Event** occur. (Note that commonly a resultant state of the **Event** will stand in for the **Event**.) In some cases, the **Experiencer** is an active participant in the **Event**, and in such cases the **Event** itself is often not mentioned, but rather some **Focal participant** which is subordinately involved in the **Event**.

Generally, the use of a word in this frame implies that the specific **Event** has not yet happened, but that the **Experiencer** believes that they would be happier if it did. Sometimes the **Time of Event**, **Purpose of Event**, or the **Location of Event** are mentioned without the explicit mention of the **Event**.

- I only **WANTED** one piece of candy.
- The company was **EAGER** for him to leave as soon as possible.
- Susan **WISHES** that you’d listen to her.

FEs:

Core:

- **Event** (Event)
  - Semantic Type: State_of_affairs

- **Experiencer** (Exp)
  - Semantic Type: Sentient

- **Focal participant** (Foc)

- **Location of Event** (PEv)
  - The **Location of Event** is the place involved in the desired **Event**.
  - **I WANT** that box on top of the other one.

  The prince **WISHES** you **here** before matins.
AGROVOC Thesaurus

AGROVOC is a multilingual, structured and controlled vocabulary designed to cover the terminology of all subject fields in agriculture, forestry, fisheries, food and related domains (e.g. environment).

Learn more about AGROVOC by browsing: AGROVOC Flyer

Term code: 2790

Legend for relationships

Do you want to know more about AGROVOC? See some illustrations.
The general requirement

• To transform any type of content (documents, links, extracted document fragments, tags, metadata, schemas, data structures, KOS, etc.) into semantic knowledge resources
Research requirements

• We need information reengineering methods that create semantics-ready datasets
  – RDF-izers
  – Triplifiers
  – eRDF, SW annotation systems
  – DB lifters
  – Ontology-based DB access
  – NLP learning techniques (term extraction, taxonomy induction, NER, LSI, relation extraction, frame detection, …)
• As well as semantic reengineering methods to make use of the datasets while avoiding the mess
  – Content design patterns (time, space, situations, measures, sequences, components, control, …)
    • http://www.ontologydesignpatterns.org
  – SKOS one widely adopted metamodel
  – LMF an example of an external standard
  – LIR an example of a bridging vocabulary
  – Also: Gruber’s tag onto, MOAT, SIOC, IRW
  – LMM a comprehensive metamodel in the Semion framework
Standard languages help

• Transform all (or some data) in RDF, or even OWL
  – Cf. Triplify initiative

• Then search for RDF and make integrating apps (semantic mash-ups)
Integrated knowledge search

Yago

Umbel

everything based on a centralized ontology ...
of mixed quality

Freebase
Patterns for Re-engineering Non-Ontological Resources (PR-NOR)

- 8 Patterns for Re-engineering Classification Schemes
- 8 Patterns for Re-engineering Thesauri
- 4 Patterns for Re-engineering Lexica

PR-NORs define a procedure that transforms the non-ontological resource components into ontology representational primitives.

WP 5

A Method for Reusing and Re-engineering Non-Ontological Resources for Building Ontologies (Scenario 2)

Sustainability

This NeOn technology is maintained within the Ontology Design Patterns Portal, under the Re-engineering Patterns Category.
Some techniques for semantic data reuse

• **Virtual linked data**
  - Automatic RDB schema conversion to RDFS
  - RDB data browsing and on-demand automatic conversion to RDF
  - Sample tools: Sparql endpoint+D2R
  - Dataset example: IMDB
  - +Time to usage –Flexibility

• **Ontology-based access with ad-hoc queries**
  - (DL-Lite) ontology to be designed separately
  - Ad-hoc SQL query on RDB, “embedded” in class spec
  - On demand ontology-based navigation
  - Sample tools: Mastro+Quonto
  - +Complexity –Flexibility –Time to usage

• **Physical linked data with custom ontologies**
  - Custom conversion of RDB/XML to one or more OWL ontologies
  - Custom conversion of data to RDF-OWL datasets that can be published and queried
  - Sample tools: Sparql endpoint+Semion
  - Sample datasets: DBpedia, data.cnr.it
  - +Flexibility ±Time to usage ±Complexity

• **Key aspects**
  - Mapping specification
  - Consumable RDF data semantics
Transformation patterns

- Types of transformation patterns
  1. Direct structural morphism

<table>
<thead>
<tr>
<th>Broader</th>
<th>Narrow concept</th>
<th>Broad concept</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Paris</td>
<td>France</td>
</tr>
</tbody>
</table>

:Broader rdf:type dbs:Table
:Narrow_concept rdf:type dbs:Column
:France rdf:type dbs:Datum

2. Semantic interpretation

<table>
<thead>
<tr>
<th>Broader</th>
<th>Narrow concept</th>
<th>Broad concept</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Paris</td>
<td>France</td>
</tr>
</tbody>
</table>

:Broader rdf:type owl:ObjectProperty
:Narrow_concept rdf:type owl:Class
:Paris :broader :France
3. Re-interpretation, e.g. through alignment patterns
   - :Narrow_concept $\rightarrow$ skos:Concept
     - also as mediated semantic interpretation
     - also as revised semantic interpretation

4. Production: new entities, vocabulary/string manipulation
   - :Narrow_concept $\rightarrow$ :Concept
   - “%p” $\rightarrow$ :hyponym
   - “plant flora plant_life” $\rightarrow$ :Plant, :Flora, :PlantLife
“Knowledge soup” in legacy FIGIS DTDs
Semion method and tool
What Reengineering is

- Reengineering is a transformation
  - allows to model a system, i.e. a data source, into a new form
  - is a set of activities that restructure a “legacy system” into a new one

- In the RDF context reengineering means to syntactically transform a non-RDF source to RDF
LOD and reengineering

• LOD is fed by reengineers also called “triplifiers”, tools able to transform content to Linked Data

• Triplifiers implement various methods typically based on recipes which describe the mapping of the original source to RDF

• Typically the transformation relies on predetermined implicit assumptions on the domain semantics of the non-RDF data source
A typical DB2RDF mapping

• Table -> Class
• Column -> Property
• Record -> individual
Example

Class: Person

DatatypeProperty: firstName

DatatypeProperty: lastName

Individual: Person1
  Types: Person
  Facts:
    - firstName "Aldo"
    - lastName "Gangemi"

Individual: Person2
  Types: Person
  Facts:
    - firstName "Valentina"
    - lastName "Presutti"

...
Implications

• limited customization of the transformation process

• difficulty in adopting good practices of knowledge reengineering and ontology design

• limited exploitation of OWL expressivity for describing the domain
The Semion Approach

- original source schema/data
  - syntactic reengineering
  - refactoring
- ontology from schema
- individuals from data
- ontology from data
- Mediator ontology
- A-box refactoring
- T-box refactoring
The reengineer

• It does not add any semantics, but the RDF format
• Semion needs the meta-model of structure of the source and some code
• Currently supports RDBMS and XML
• Supported sources can be extended by providing new reengineering services as an OSGi bundle (not available yet)
Basic idea
A meta-model for RDBs
Example of transformation of a DB

Class: Table

ObjectProperty: hasRecord
  Domain: Table
  Range: Record
  inverseOf: isRecordOf

ObjectProperty: isRecordOf
  Domain: Table
  Range: Record
  inverseOf: hasRecord

Individual: Person
  Type: Table
  Facts: hasRecord AldoGangemi

Individual: AldoGangemi
  Types: Record
  Facts:
    hasDatum AldoGangemiFirstName
    hasDatum AldoGangemiLastName
  Facts: isRecordOf Person
Reengineering Rules

• Table -> individual of dbs:Table
• Column -> individual of dbs:Column
• Record -> individual of dbs:Record
• Field -> individual of dbs:Datum
Primary Keys

- primary keys are used for URI generation

<table>
<thead>
<tr>
<th>Person</th>
<th>FirstName</th>
<th>LastName</th>
<th>id (PK)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Aldo</td>
<td>Gangemi</td>
<td>stlab.istc-cnr1</td>
</tr>
</tbody>
</table>

Individual: Person_stlab.istc-cnr1
Type: Record
Foreign Keys

- foreign keys identifies relations between tables and are mapped to relations between individuals

<table>
<thead>
<tr>
<th>Person</th>
<th>FirstName</th>
<th>LastName</th>
<th>id (PK)</th>
<th>affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Aldo</td>
<td>Gangemi</td>
<td>stlab.istc-cnr1</td>
<td>istc-cnr</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Institute</th>
<th>id (PK)</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>istc-cnr</td>
<td>Istituto di Scienze e Tecnologie della Cognizione – CNR</td>
</tr>
</tbody>
</table>

Individual: Person_stlab.istc-cnr1
  Type: Record
  Facts: hasDatum Person_stlab.istc-cnr1_affiliation

Individual: Person_stlab.istc-cnr1_affiliation
  Type: Datum
  Facts: hasContent Institute_istc-cnr

Individual: Institute_istc-cnr
  Type: Record
  Facts: hasDatum Institute_istc-cnr_name
...and for XML
Example XML

<parameters applicable-location="SanFrancisco">
  <temperature type="maximum" units="Fahrenheit" time-layout="k-p12h-n14-3">
    <name>Daily Maximum Temperature</name>
    <value>38</value>
  </temperature>
  <temperature type="minimum" units="Fahrenheit" time-layout="k-p12h-n14-3">
    <name>Daily Minimum Temperature</name>
  </temperature>
  <probability-of-precipitation type="12 hour" units="percent" time-layout="k-p12h-n14-3">
    <name>12 Hourly Probability of Precipitation</name>
    <value>27</value>
  </probability-of-precipitation>
  <weather time-layout="k-p12h-n14-3">
    <name>Weather Type, Coverage, and Intensity</name>
    <weather-conditions weather-summary="Mostly Cloudy"/>
  </weather>
</parameters>
Example XML
The Refactor

• allows to align a data set expressed with a specific vocabulary/ontology to another vocabulary/ontology
• is expressed as a set of rules
• rules are expressed into a human readable syntax called SemionRule Syntax and can be transformed into
  – SWRL rules for reasoning
  – SPARQL CONSTRUCT for pure refactoring
• rules realize recipes that can be saved (refactoring patterns)
myRule[
  is(dbs:Table, ?x) . has(dbs:hasColumn, ?x, ?y)
  ->
  is/owl:Class, ?x)
]

as a SPARQL CONSTRUCT

PREFIX dbs: <http://ontologydesignpatterns.org/ont/iks/dbs_l1.owl#> .
PREFIX owl: <http://www.w3.org/2002/07/owl#> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

CONSTRUCT{ ?x rdf:type owl:Class }
WHERE{
  ?x rdf:type dbs:Table .
  ?x dbs:hasColumn ?y
}
...and as a SWRL rule

<swrl:Variable rdf:ID="x"/>
<swrl:Variable rdf:ID="y"/>
<swrl:Imp>
  < swrl:body rdf:parseType="Collection">
    < swrl:ClassAtom>
      < swrl:classPredicate rdf:resource="&dbs;Table"/>
      < swrl:argument1 rdf:resource="#x" />
    </swrl:ClassAtom>
    < swrl:IndividualPropertyAtom>
      < swrl:propertyPredicate rdf:resource="&dbs;hasColumn"/>
      < swrl:argument1 rdf:resource="#x" />
      <swrl:argument2 rdf:resource="#y" />
    </swrl:IndividualPropertyAtom>
  </swrl:body>
<swrl:head rdf:parseType="Collection">
  < swrl:ClassAtom>
    < swrl:classPredicate rdf:resource="&owl;Class"/>
    < swrl:argument1 rdf:resource="#x" />
  </swrl:ClassAtom>
</swrl:head>
</swrl:Imp>
The reengineering view
The refactoring view
What does Semion mean?

• it sounds like semiotics

• the aim was to put informal knowledge from structured sources into the context of the semiotic triangle

• LMM (Linguistic Meta-Model) is an OWL ontology that formalized the semiotic triangle
The Linguistic Meta-Model (LMM)

LMM plays the role of a mediator ontology.

LMM allows a semiotic cognitive representation of knowledge based on the so-called semiotic triangle.

Most knowledge representation schemata can be aligned to the semiotic triangle.

A mediator is an ontology that describes the entities used for expressing the informal semantics that we use for organizing our knowledge.
Reengineering process

- Data source
  - Define/acquire data source Meta-Model
  - Define/acquire mapping description

- Data source + MM + mapping
  - Apply reengineering

- RDF dataset

Refactoring process

- RDF dataset
  - Define alignments to LMM
  - Perform alignments to LMM

- LMM aligned dataset
  - Define alignments to a formal semantics MM
  - Perform alignments to the formal semantics MM

- LMM aligned dataset
  - Define alignments to a logic MM
  - Perform alignments to the logic MM

just extract RDF triples!

express the domain semantics
Stanbol Rules
Rule-based refactoring

- the Stanbol Refactor applies RDF graph transformation by means of transformation rules
- rules drive the transformation
- a set of rules for a tranformation task characterize a recipe
- a recipe identifies the kind of transformation and the rules need by the transformation task
Recipes, rules and Stanbol
Recipes, rules and Stanbol
Recipes, rules and Stanbol

input graph

Recipe

Rule1

Rule2

Rule3
Recipes, rules and Stanbol
Recipes, rules and Stanbol

input graph

Recipe

Rule1
Rule2
Rule3

output graph

Stanbol™
What is a rule?

• in logic, a *transformation rule* or *rule of inference* is a syntactic rule or function which takes premises and returns a conclusion

• the rule is *sound* with respect to the semantics of classical logic in the sense that if the premises are interpreted to be true then so is the conclusion
Rule Examples

• rule pattern (modus ponens)
  – if condition then consequent

• a rule example
  – if X is a person then X has a father
    (i.e. every person has a father)
    • ∀x∃y. Person(x) ⇒ hasFather(x, y)

  – if Y is the father of X and Z the brother of Y then Z is the uncle of X
    (i.e. the brother of the father is the uncle)
    • ∀xyz. hasFather(x,y) ∧ hasBrother(y,z) ⇒ hasUncle(x,z)
Stanbol Rule Syntax

in Stanbol a rule is defined as

```
ruleName[body -> head]
```

where:

- the `ruleName` identifies the rule
- the body is a set of `atoms` that must be satisfied by evaluating the rule
- the head or consequent is a set of `atoms` that must be true if the condition is evaluated to be true
Stanbol Rule Syntax

.ruleName[body -> head]

where

– both body and head consist of a list of conjunctive atoms
  • body = atom1 . atom2 . . . . atomN
  • head = atom1 . atom2 . . . . atomM

– the conjunction $\land$ in Stanbol Rules is expressed with the symbol “ . ”
Rule Atoms

• An atom is the smallest unit of the interpretation of a rule
  – e.g.: in predicate calculus
    \[ Person(x) \Rightarrow hasFather(x, y) \]
    Person(\textbullet) and hasFather(\textbullet,\textbullet) are two atoms

• In Stanbol basic atoms are
  – Class assertion atom
  – Individual assertion atom
  – Data value assertion atom
  – Range assertion atom

• There are also comparison atoms, string and integer manipulation atoms
Atom’s notation

• The atoms may contain
  – constants: they consist of URIs or Literals (values)
    • e.g. http://dbpedia.org/resource/Bob_Marley is a constant, but “Bob Marley”^^xsd:string is a constant too
  – variables: any identifier preceded by ?
    • e.g. ?x is a variable, but also ?y is a variable
Class assertion atom

A class assertion atom is identified by the operator

\[ is(\text{classPredicate}, \text{argument}) \]

where

- \text{classPredicate} is a URI that identifies a class
- \text{argument} is the resource that has to be proved as typed with the classPredicate. It can be either a constant (a URI) or a variable

E.g. \[ is(<\text{http://xmlns.com/foaf/0.1/Person}>, \ ?x) \] returns \text{true} if the concrete value associated to \( ?x \) is typed as \text{http://xmlns.com/foaf/0.1/Person}
Individual assertion atom

\[ \text{has}(\text{propertyPredicate}, \text{arg1}, \text{arg2}) \]

where

- \text{propertyPredicate} is the object property that has to be evaluated. It can be a constant (URI) or a variable (\(?x\))
- \text{arg1} and \text{arg2} are the two arguments of the property. They can be either constants (URI) or variables (\(?x\))
Datavalued assertion atom

\[ \text{values(propertyPredicate, arg1, arg2)} \]

where

- `propertyPredicate` is the datatype property that has to be evaluated. It can be a constant (URI) or a variable (\( ?x \))
- `arg1` can be either a constant (i.e. URI) or a variable (i.e. \( ?x \))
- `arg2` can be either a constant (i.e. a literal) or a variable (i.e. \( ?x \))
Range assertion atom

\[ \text{range} \left( \text{propertyPredicate}, \text{arg} \right) \]

where

- \( \text{propertyPredicate} \) is the object property that has to be evaluated. It can be a constant (URI) or a variable (\(?x\))
- \( \text{arg1} \) can be either a constant (i.e. URI) or a variable (i.e. \(?x\))
Sample rule

Considering Stanbol Rules, the FOL formula

\[ \text{hasFather}(x, y) \land \text{hasBrother}(y, z) \Rightarrow \text{hasUncle}(x, z) \]

does not become

\[
\text{myRule[ has(\langle http://myont.org/hasFather\rangle, ~x, ~y) . has(\langle http://myont.org/hasBrother\rangle, ~y, ~z) -> has(\langle http://myont.org/hasUncle\rangle, ~x, ~z) ]}
\]
Namespace Prefixes

• URIs are useful, but sometime too long for humans
• we could use namespace prefixes instead of full URIs in rule atoms
• e.g:

```xml
myont = <http://myont.org/> .
myRule[ has(myont:hasFather, ?x, ?y) .
   has(myont:hasBrother, ?y, ?z)
   ->
   has(myont:hasUncle, ?x, ?z) ]
```
Define a refactoring recipe

we want to use the FOAF vocabulary instead of SKOS
Define a refactoring recipe

```
skos = <http://www.w3.org/2004/02/skos/core#> .
foaf = <http://xmlns.com/foaf/0.1/> .

conceptToPerson[ is(skos:Concept, ?x) -> is (foaf:Person, ?x) ] .
labelRule[ values(skos:prefLabel, ?x, ?y) -> values (foaf:name, ?x, ?y) ]
```
How the refactor works

• each rule is executed individually starting from the first in the recipe

• each rule is interpreted and executed as a SPARQL CONSTRUCT
From rules to CONSTRUCT

The rule

```
skos = <http://www.w3.org/2004/02/skos/core#> .
foaf = <http://xmlns.com/foaf/0.1/> .
```

conceptToPerson[ is(skos:Concept, ?x) -> is(foaf:Person, ?x) ]

is interpreted as

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT { ?x rdf:type foaf:Person }
WHERE { ?x rdf:type skos:Concept }
```
Comparison atoms

– **same**(arg1, arg2): returns true if arg1 is equal to arg2
– **different**(arg1, arg2): returns true if arg1 is different from arg2
– **greaterThan**(arg1, arg2): returns true if arg1 > arg2
– **lessThan**(arg1, arg2): returns true if arg1 < arg2
– **startsWith**(arg1, arg2): returns true if the string associated to arg1 starts with the string associated to arg2
– **endsWith**(arg1, arg2): returns true if the string associated to arg1 ends with the string associated to arg2
String manipulation

- Used for name refactoring
  - `concat(arg1, arg2)`: returns a string that is the concatenation of `arg1+arg2`
  - `substring(arg, start, length)`: returns the sub-string of `arg` from position `start` for `length` chars
  - `lowercase(arg)`: returns the lower case representation of `arg`
  - `lowercase(arg)`: returns the upper case representation of `arg`
  - `str(arg)`: returns the literal value of any RDF object
  - `namespace(arg)`: returns the namespace as a string of any URI
    - e.g. `namespace(<http://www.foo.org#obj>)` -> “http://www.foo.org#”
  - `localname(arg)`: returns the local as a string of any URI
    - e.g. `localname(<http://www.foo.org#obj>)` -> “obj”
Assignment

\[ \text{let}(\text{arg1}, \text{arg2}) \]

where

- \text{arg1} is a variable
- \text{arg2} is a constant
- e.g. \text{let}(?x, "Aldo Gangemi") binds the variable \textit{?x} to the literal “Aldo Gangemi”
Production

`newNode(arg1, arg2)`

where

- `arg1` is a variable
- `arg2` is a Literal
- e.g. `newNode(?x, "http://stlab.istc.cnr/Aldo Gangemi")` binds the variable `?x` to the URI obtained from the literal `http://stlab.istc.cnr/Aldo Gangemi`, namely
  - `<http://stlab.istc.cnr/Aldo Gangemi>`
Arithmetical atoms

- $\text{sum}(\text{arg1}, \text{arg2})$: returns a new integer that is equal to $\text{arg1} + \text{arg2}$
- $\text{sub}(\text{arg1}, \text{arg2})$: returns a new integer that is equal to $\text{arg1} - \text{arg2}$
Refactor REST services

• we may use a recipe on the fly

```
$ curl -X POST -d recipe="skos = <http://www.w3.org/2004/02/skos/core#> . foaf = <http://xmlns.com/foaf/0.1/> . conceptToPerson[ is (skos:Concept, ?x) -> is(foaf:Person, ?x) ] . labelRule[ values (skos:prefLabel, ?x, ?y) -> values(foaf:name, ?x, ?y) ]" -d input=@my/local/graph.rdf http://localhost:8080/rules/recipe/apply
```
Refactor REST services

• we can also create and store an empty recipe in the rule store

$ curl -X POST -d recipe="recipe-ID" -d description="Description of the recipe." http://localhost:8080/rules/recipe

• and add the rules one-by-one

$ curl -X POST -d recipe="recipe-ID" -d rule="skos = <http://www.w3.org/2004/02/skos/core#> . foaf = <http://xmlns.com/foaf/0.1/> . conceptToPerson[ is(skos:Concept, ?x) -> is(foaf:Person, ?x) ]" -d description "Description of the rule." http://localhost:8080/rules
Exercise

• Download Semion and launch it as follows:
  – (Mac) java -jar -Xmx512m -XstartOnFirstThread /LocalPathname/it.cnr.istc.semion.tool-0.6-SNAPSHOT.one-jar.jar
  – (Win) java -jar -Xmx512m \LocalPathname\it.cnr.istc.semion.tool-0.6-SNAPSHOT.one-jar.jar

• Connect to the indicated database and perform reengineering first, and alignment (refactoring) second
Appendix: SPARQL Primer
SPARQL

• SPARQL (pronounced "sparkle") is an RDF query language
• it stands for Simple Protocol And RDF Query Language
• allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns
Anatomy of a Query

Declare prefix shortcuts (optional)

Define the dataset (optional)

Query modifiers (optional)

Query pattern

Query result clause

PREFIX foo: <...>
PREFIX bar: <...>

SELECT ...
FROM <...>
FROM NAMED <...>
WHERE {
...
}
GROUP BY ...
HAVING ...
ORDER BY ...
LIMIT ...
OFFSET ...
BINDINGS ...

courtesy of Lee Feigenbaum
Types of SPARQL Queries

**SELECT** queries

- **Project out specific variables and expressions:**
  
  ```sparql
  SELECT ?c ?cap (1000 * ?people AS ?pop)
  ```

- **Project out all variables:**
  ```sparql
  SELECT *
  ```

- **Project out distinct combinations only:**
  ```sparql
  SELECT DISTINCT ?country
  ```

**RESULTS**

<table>
<thead>
<tr>
<th>?c</th>
<th>?cap</th>
<th>?pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:France</td>
<td>ex:Paris</td>
<td>63,500,000</td>
</tr>
<tr>
<td>ex:Canada</td>
<td>ex:Ottawa</td>
<td>32,900,000</td>
</tr>
<tr>
<td>ex:Italy</td>
<td>ex:Rome</td>
<td>58,900,000</td>
</tr>
</tbody>
</table>

**CONSTRUCT** queries

- **Construct RDF triples/graphs:**
  ```sparql
  CONSTRUCT {
    ?country a ex:HolidayDestination;
    ex:arrive_at ?capital;
    ex:population ?population .
  }
  ```

**RESULTS**

```sparql
ex:France a ex:HolidayDestination;
  ex:arrive_at ex:Paris;
  ex:population 635000000 .
ex:Canada a ex:HolidayDestination;
  ex:arrive_at ex:Ottawa;
  ex:population 329000000 .
```

courtesy of Lee Feigenbaum

**ASK** queries

- **Ask whether or not there are any matches:**
  ```sparql
  ASK
  ```

**RESULTS**

Result is either “true” or “false” (in XML or JSON):

```
true, false
```

courtesy of Lee Feigenbaum
Write full URIs:

```xml
<http://this.is.a/full/URI/written#out>
```

Abbreviate URIs with prefixes:

```xml
PREFIX foo: <http://this.is.a/URI/prefix#>  
  ... foo:bar ...  
  ⇒ http://this.is.a/URI/prefix#bar
```

Shortcuts:

```xml
a  ⇒ rdf:type
```

Plain literals:

```
"a plain literal"
```

Plain literal with language tag:

```
"bonjour"@fr
```

Typed literal:

```
"13"^^xsd:integer
```

Shortcuts:

```
true  ⇒ "true"^^xsd:boolean
3  ⇒ "3"^^xsd:integer
4.2  ⇒ "4.2"^^xsd:decimal
```

Variables:

```
?var1, ?anotherVar, ?and_one_more
```

Comments:

```
# Comments start with a '#'  
# continue to the end of the line
```

Match an exact RDF triple:

```
ex:myWidget ex:partNumber "XY24Z1" .
```

Match one variable:

```
?person foaf:name "Lee Feigenbaum" .
```

Match multiple variables:

```
```

courtesy of Lee Feigenbaum
## Some Public SPARQL Endpoints

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>What’s there?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARQLer</td>
<td><a href="http://sparql.org/sparql.html">http://sparql.org/sparql.html</a></td>
<td>General-purpose query endpoint for Web-accessible data</td>
</tr>
<tr>
<td>DBPedia</td>
<td><a href="http://dbpedia.org/sparql">http://dbpedia.org/sparql</a></td>
<td>Extensive RDF data from Wikipedia</td>
</tr>
<tr>
<td>DBLP</td>
<td><a href="http://www4.wiwiss.fu-berlin.de/dblp/snorql/">http://www4.wiwiss.fu-berlin.de/dblp/snorql/</a></td>
<td>Bibliographic data from computer science journals and conferences</td>
</tr>
<tr>
<td>LinkedMDB</td>
<td><a href="http://data.linkedmdb.org/sparql">http://data.linkedmdb.org/sparql</a></td>
<td>Films, actors, directors, writers, producers, etc.</td>
</tr>
<tr>
<td>bio2rdf</td>
<td><a href="http://bio2rdf.org/sparql">http://bio2rdf.org/sparql</a></td>
<td>Bioinformatics data from around 40 public databases</td>
</tr>
</tbody>
</table>
SPARQL Over HTTP (the SPARQL Protocol)

http://host.domain.com/sparql/endpoint?<parameters>

where <parameters> can include:

query=<encoded query string>
  e.g. SELECT+*%0DWHERE+{...

default-graph-uri=<encoded graph URI>
  e.g. http%3A%2F%2Fexample.com%2Ffoo...
  n.b. zero of more occurrences of default-graph-uri

named-graph-uri=<encoded graph URI>
  e.g. http%3A%2F%2Fexample.com%2Fbar...
  n.b. zero of more occurrences of named-graph-uri

HTTP GET or POST. Graphs given in the protocol override graphs given in the query.