Materializing the Web of Linked Data

NIKOLAOS KONSTANTINOU

DIMITRIOS-EMMANUEL SPANOS

Book

“Materializing the Web of Linked Data”

Nikolaos Konstantinou and Dimitrios-Emmanuel Spanos

© Springer, 2015

ISBN

- 978-3-319-16073-3 (Print)
- 978-3-319-16074-0 (Online)

DOI

- 10.1007/978-3-319-16074-0

Contents

1. Introduction: Linked Data and the Semantic Web
2. Technical Background
3. Deploying Linked Open Data: Methodologies and Software Tools
4. Creating Linked Data from Relational Databases
5. Generating Linked Data in Real-time from Sensor Data Streams
6. Conclusions: Summary and Outlook

Chapter 1

Introduction

Linked Data and the Semantic Web

NIKOLAOS KONSTANTINOU

DIMITRIOS-EMMANUEL SPANOS

Outline

Introduction

Preliminaries

The Linked Open Data Cloud

The Origin of the Semantic Web

Semantic Web

- Term primarily coined by Tim Berners-Lee
- For the Web to be understandable both by humans and software, it should incorporate its meaning
  - Its *semantics*

Linked Data implements the Semantic Web vision

Why a Semantic Web? (1)

Information management

- Sharing, accessing, retrieving, consuming information
  - Increasingly tedious task

Search engines

- Rely on Information Retrieval techniques
- Keyword-based searches
  - Do not release the information potential
    - Large quantity of existing data on the web is not stored in HTML
      - The Deep Web

Why a Semantic Web? (2)

Content meaning should be taken into account

- Structure to the meaningful Web page content
- Enable software agents to
  - Roam from page to page
  - Carry out sophisticated tasks for users

Semantic Web

- Complement, not replace current Web
- Transform content into an exploitable source of knowledge

Why a Semantic Web? (3)

Web of Data

- An emerging web of interconnected published datasets in the form of Linked Data
- Implements the Semantic Web vision

The Need for Adding Semantics (1)

Semantics

- From the Greek word *σημαντικός*
  - Pronounced *s̱imantikós*, means *significant*
- Term typically used to denote the study of meaning
- Using semantics
  - Capture the interpretation of a formal or natural language
  - Enables entailment, application of logical consequence
    - About relationships between the statements that are expressed in this language

The Need for Adding Semantics (2)

Syntax

- The study of the principles and processes by which sentences can be formed
- Also of Greek origin
  - συν and τάξις
  - Pronounced sin and taxis
  - Mean together and ordering, respectively

The Need for Adding Semantics (3)

Two statements can be syntactically different but semantically equivalent

- Their meaning (semantics) is the same, the syntax is different
  - More than one ways to state an assumption

The Need for Adding Semantics (4)

Search engines rely on keyword matches

- Keyword-based technologies
  - Have come a long way since the dawn of the internet
  - Will return accurate results
  - Based on keyword matches
    - Extraction and indexing of keywords contained in web pages
- But
  - More complex queries with semantics still partially covered
  - Most of the information will not be queried

Traditional search engines

Rely on keyword matches

Low precision

- Important results may not be fetched, or
- Ranked low
  - No exact keyword matches

Keyword-based searches

Do not return usually the desired results

Despite the amounts of information

Typical user behavior

- Change query rather than navigate beyond the first page of the search results

The Semantic Web (1)

Tackle these issues

- By offering ways to describe of Web resources

Enrich existing information

- Add semantics that specify the resources
- Understandable both by humans and computers

Information easier to discover, classify and sort

Enable semantic information integration

Increase serendipitous discovery of information

Allow inference

The Semantic Web (2)

Example

- A researcher would not have to visit every result page and distinguish the relevant from the irrelevant results
- Instead, retrieve a list of more related, machine-processable information
  - Ideally containing links to more relevant information

Outline

Introduction

Preliminaries

The Linked Open Data Cloud

Data-Information-Knowledge (1)

Data

> Information, especially facts or numbers, collected to be examined and considered and used to help decision-making, or information in an electronic form that can be stored and used by a computer

Information

- “facts about a situation, person, event, etc.”

Smallest information particle

- The bit
  - Replies with a yes or no (1 or 0)
  - Can carry data, but in the same time, it can carry the result of a process
  - For instance whether an experiment had a successful conclusion or not

Data-Information-Knowledge (2)

E.g. “temperature” in a relational database could be

- The information produced after statistical analysis over numerous measurements
  - By a researcher wishing to extract the average value in a region over the years
- A sensor measurement
  - Part of the data that will contribute in drawing conclusions regarding climate change

Data-Information-Knowledge (3)

Knowledge

Understanding of or information about a subject that you get by experience or study, either known by one person or by people generally

- Key term: "by experience or study"
  - Implies processing of the underlying information

Data-Information-Knowledge (4)

No clear lines between them

- Term that is used depends on the respective point of view

When we process collected data in order to extract meaning, we create new information

Addition of semantics

- Indispensable in generating new knowledge
- Semantic enrichment of the information
  - Makes it unambiguously understood by any interested party (by people generally)

Data → Information → Knowledge

Heterogeneity

Distributed data sources

- Interconnected, or not
- Different models and schemas
- Differences in the vocabulary
- Syntactic mismatch
- Semantic mismatch

Interoperability (1)

Interoperable

- Two systems in position to successfully exchange information

3 non-mutually exclusive approaches

- Mapping among the concepts of each source
- Intermediation in order to translate queries
- Query-based

Protocols and standards (recommendations) are crucial

- E.g. SOAP, WSDL, microformats

Interoperability (2)

Key concept: *Schema*

- Word comes from the Greek word σχήμα
  - Pronounced schíma
  - Means the shape, the outline
  - Can be regarded as a common agreement regarding the interchanged data
- Data schema
  - Defines how the data is to be structured

Information Integration (1)

Combine information from heterogeneous systems, sources of storage and processing

- Ability to process and handle it as a whole

Global-As-View

- Every element of the global schema is expressed as a query/view over the schemas of the sources
- Preferable when the source schemas are not subject to frequent changes

Information Integration (2)

Local-As-View

- Every element of the local schemas is expressed as a query/view over the global schema

P2P

- Mappings among the sources exist but no common schema

Information Integration Architecture (1)

A source Π_1 with schema S_1

A source Π_2 with a schema S_2

…

A source Π_v with source S_v

A global schema S in which the higher level queries are posed

Information Integration Architecture (2)

The goal in the information integration problem

- Submit queries to the global schema *S*
- Receive answers from S_1, S_2, …, S_v
- Without having to deal with or even being aware of the heterogeneity in the information sources

Data Integration (1)

A data integration system

- A triple $I = \langle G; S; M \rangle$
  - G is the global schema,
  - S is the source schema and
  - M is a set of mappings between G and S

Data Integration (2)

Local-As-View

- Each declaration in *M* maps an element from the source schema *S* to a query (a view) over the global schema *G*

Global-As-View

- Each declaration in *M* maps an element of the global schema *G* to a query (a view) over the source schema *S*

The global schema *G* is a unified view over the heterogeneous set of data sources

Data Integration (3)

Same goal

- Unifying the data sources under the same common schema

Semantic information integration

- Addition of its semantics in the resulting integration scheme

Mapping

Using the definition of data integration systems, we can define the concept of mapping

- A mapping m (member of M) from a schema S to a schema T
  - A declaration of the form $Q^S \rightsquigarrow Q^T$
  - Q^S is a query over S
  - Q^T a query over T

Mapping vs. Merging

(Data) mapping

- A mapping is the specification of a mechanism
  - The members of a model are transformed to members of another model
- The meta-model that can be the same, or different
- A mapping can be declared as a set of relationships, constraints, rules, templates or parameters
  - Defined during the mapping process, or through other forms that have not yet been defined

Merging

- Implies unifying the information at the implementation/storage layer

Annotation (1)

Addition of metadata to the data

Data can be encoded in any standard

Especially important in cases when data is not human-understandable in its primary form

- E.g. multimedia

Annotation (2)

(Simple) annotation

- Uses keywords or other ad hoc serialization
  - Impedes further processing of the metadata

Semantic annotation

- Describe the data using a common, established way
- Addition of the semantics in the annotation (i.e. the metadata)
- In a way that the annotation will be machine-processable
  - In order to be able to infer additional knowledge

Problems with (Semantic) Annotation

Time-consuming

- Users simply do not have enough time or do not consider it important enough in order to invest time to annotate their content

Familiarity required with both the conceptual and technical parts of the annotation

- User performing the annotation must be familiar with both the technical as well as the conceptual part

Outdated annotations

- A risk, especially in rapidly changing environments with lack of automation in the annotation process

Automated Annotation

Incomplete annotation is preferable to absent

- E.g. in video streams

Limitations of systems performing automated annotation

- Limited recall and precision (lost or inaccurate annotations)

Metadata (1)

Data about data

Efficient materialization with the use of ontologies

Ontologies can be used to describe Web resources

- Adding descriptions about their semantics and their relations
- Aims to make resources machine-understandable
- Each (semantic) annotation can correspond to a piece of information
- It is important to follow a common standard

Metadata (2)

Annotation is commonly referred to as “metadata”

Usually in a semi-structured form

- Semi-structured data sources
  - Structure accompanies the metadata
  - E.g. XML, JSON
- Structured data sources
  - Structure is stored separately
  - E.g. relational databases

Metadata (3)

Powerful languages

- Practically unlimited hierarchical structure
- Covers most of the description requirements that may occur
- Storage in separate files allows collaboration with communication protocols
- Files are independent from the environment in which they reside
  - Resilience to technological evolutions

Negatives

- Expressivity of the description model
- Limited way of structuring the information

Metadata (4)

Inclusion of semantics

- Need for more expressive capabilities and terminology

Ontologies

- Offer a richer way of describing information
  - E.g. defining relationships among concepts such as subclass/superclass, mutually disjoint concepts, inverse concepts, etc.
- RDF can be regarded as the evolution of XML
- OWL enables more comprehensive, precise and consistent description of Web Resources

Ontologies (1)

Additional description to an unclear model that aims to further clarify it

Conceptual description model of a domain

- Describe its related concepts and their relationships

Can be understood both by human and computer

- A shared conceptualization of a given specific domain

Aim at bridging and integrating multiple and heterogeneous digital content on a semantic level

Ontologies (2)

In Philosophy, a systematic recording of “Existence”

For a software system, something that “exists” is something that can be represented

The specification of a conceptualization (Gruber 1995)

- A set of definitions that associate the names of entities in the universe of discourse with human-readable text describing the meaning of the names
- A set of formal axioms that constrain the interpretation and well-formed use of these terms

Ontologies (3)

Can be used to model concepts regardless to how general or specific these concepts are

According to the degree of generalization

- Top-level ontology
- Domain ontology
- Task ontology
- Application ontology

Content is made suitable for machine consumption

- Automated increase of the system knowledge
- Logical rules to infer implicitly declared facts

Reasoners (1)

Software components

Validate consistency of an ontology

- Perform consistency checks
- Concept satisfiability and classification

Infer implicitly declared knowledge

- Apply simple rules of deductive reasoning

Reasoners (2)

Basic reasoning procedures

- Consistency checking
- Concept satisfiablility
- Concept subsumption
- Instance checking (Realization)

Properties of special interest

- Termination
- Soundness
- Completeness

Reasoners (3)

Research in Reasoning

- Tradeoff between
  - Expressiveness of ontology definition languages
  - Computational complexity of the reasoning procedure
- Discovery of efficient reasoning algorithms

Commercial

- E.g. RacerPro

Free of charge

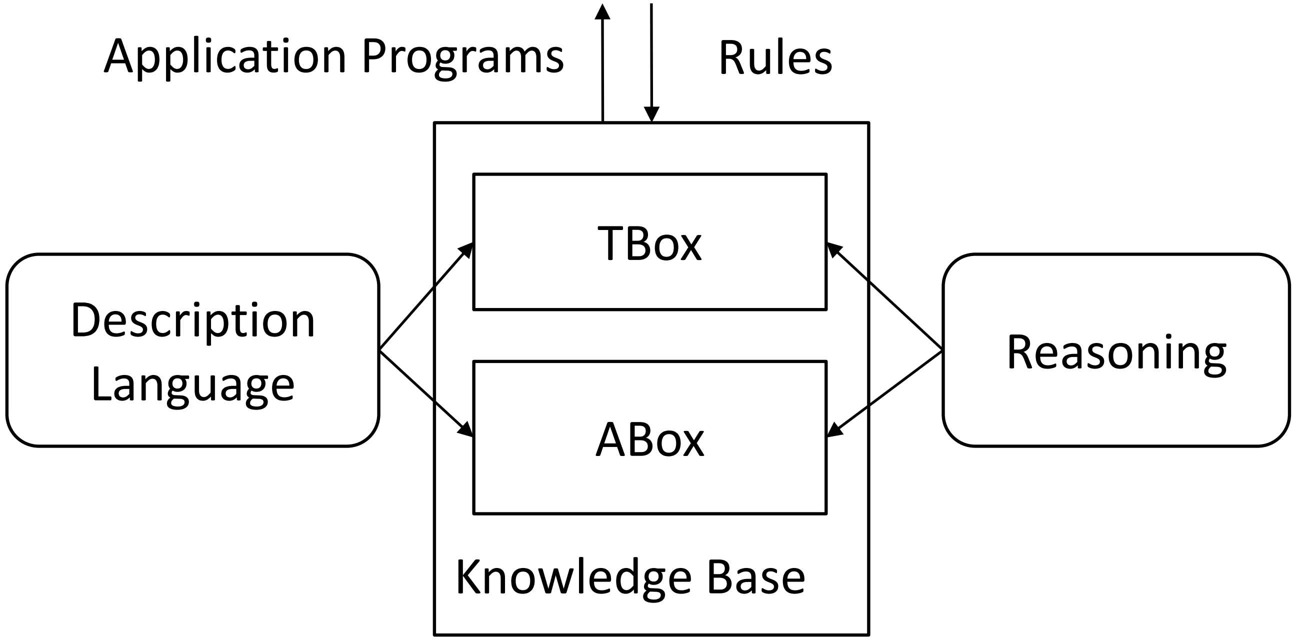
- E.g. Pellet, FaCT++

Knowledge Bases

Terminological Box (TBox)

- Concept descriptions
- Intensional knowledge
- Terminology and vocabulary

- Assertional Box (ABox)
  - The real data
  - Extensional knowledge
  - Assertions about named individuals in terms of the TBox vocabulary



Knowledge Bases vs. Databases (1)

A naïve approach:

- TBox ≡ Schema of the relational database
- ABox ≡ Schema instance

However, things are more complex than that

Knowledge Bases vs. Databases (2)

Relational model

- Supports only untyped relationships among relations
- Does not provide enough features to assert complex relationships among data
- Used in order to manipulate large and persistent models of relatively simple data

Ontological scheme

- Allows more complex relationships
- Can provide answers about the model that have not been explicitly stated to it, with the use of a reasoner
- Contains fewer but more complex data

Closed vs. Open World Assumption (1)

Closed world assumption

- Relational databases

- Everything that has not been stated as true is false
  - What is not currently known to be true is false
- A null value about a subject’s property denotes the non-existence
  - A NULL value in the isCapital field of a table Cities claims that the city is not a capital
- The database answers with certainty
  - A query “select cities that are capitals” will not return a city with a null value at a supposed boolean isCapital field

Closed vs. Open World Assumption (2)

Open world assumption

- Knowledge Bases
- A query can return three types of answers
  - True, false, cannot tell
- Information that is not explicitly declared as true is not necessarily false
  - It can also be unknown
  - Lack of knowledge does not imply falsity
- A question “Is Athens a capital city?” in an appropriate schema will return “cannot tell” if the schema is not informed
  - A database schema would return false, in the case of a null value

Monotonicity

A feature present in Knowledge Bases

A system is considered monotonic when new facts do not discard existing ones

First version of SPARQL did not include update/delete functionality

- Included in SPARQL 1.1
  - Recommendation describes that these functions should be supported
    - Does not describe the exact behavior

Outline

Introduction

Preliminaries

The Linked Open Data Cloud

The LOD Cloud (1)

Structured data

Open format

Available for everyone to use it

Published on the Web and connected using Web technologies

Related data that was not previously linked

- Or was linked using other methods

The LOD Cloud (2)

Using URIs and RDF for this is goal is very convenient

- Data can be interlinked
- Create a large pool of data
- Ability to search, combine and exploit
- Navigate between different data sources, following RDF links
  - Browse a potentially endless Web of connected data sources

The LOD Cloud (3)

Applications in many cases out of the academia

Technology maturity

Open state/government data

- data.gov (US)
- data.gov.uk (UK)
- data.gov.au (Australia)
- opengov.se (Sweden)

Open does not necessarily mean Linked

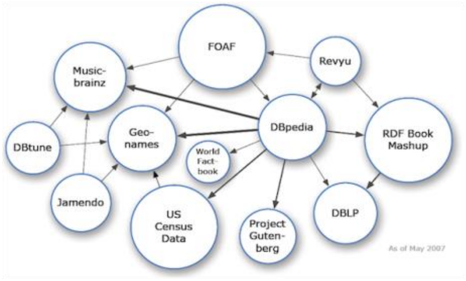
The LOD Cloud (4)

Published datasets span several domains of human activities

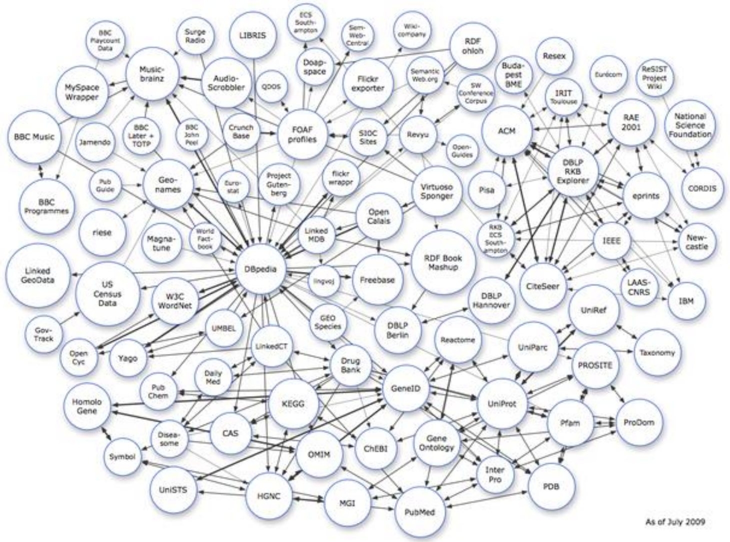
- Much more beyond government data
- Form the LOD cloud
  - Constantly increasing in terms of volume

The LOD Cloud (5)

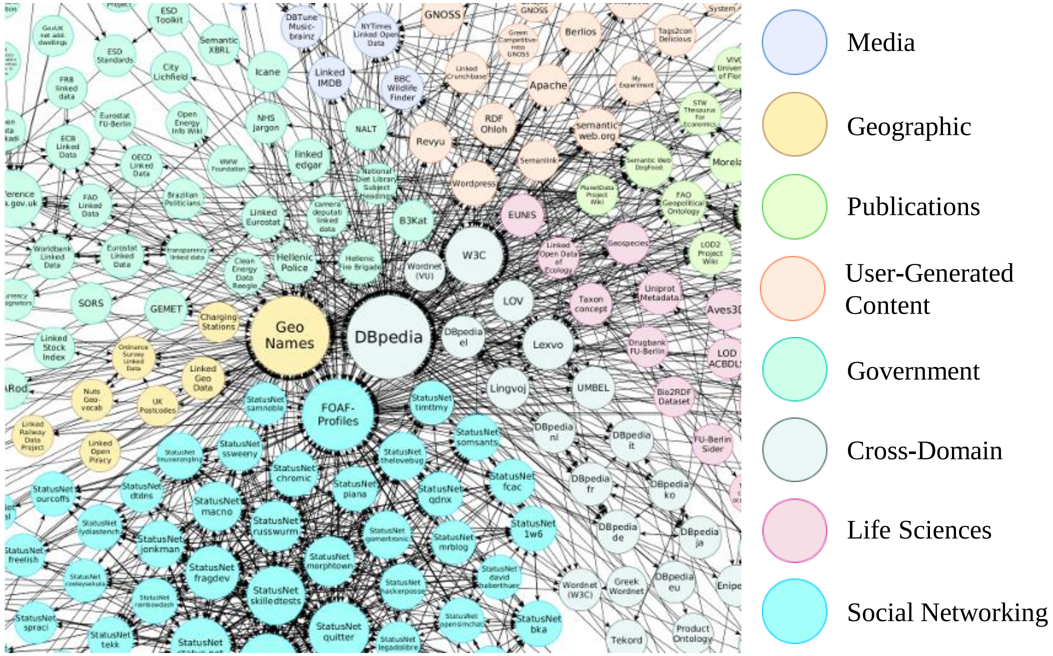
Evolution



2007



2009



2014

Chapter 2
Technical Background

NIKOLAOS KONSTANTINOU

DIMITRIOS-EMMANUEL SPANOS

Outline

Introduction

RDF and RDF Schema

Description Logics

Querying RDF data with SPARQL

Mapping relational data with R2RML

Other technologies

Ontologies

Datasets

Introduction

Linked Data

- A set of technologies
- Focused on the web
- Use the Web as a storage and communication layer
- Provide meaning to web content

Semantics

- Added value when part of a larger context
  - Data modeled as a graph

Technologies fundamental to the web

HTTP – HyperText Transfer Protocol

An application protocol for the management and transfer of hypermedia documents in decentralized information systems

Defines a number of request types and the expected actions that a server should carry out when receiving such requests

Serves as a mechanism to

- Serialize resources as a stream of bytes
  - E.g. a photo of a person
- Retrieve descriptions about resources that cannot be sent over network
  - E.g. the person itself

URI – Uniform Resource Identifier

URL

- Identifies a document location
- Address of a document or other entity that can be found online

URI

- Provides a more generic means to identify anything that exists in the world

IRI

- Internationalized URI

IRI ⊇ URI ⊇ URL

HTML – HyperText Markup Language

A markup language for the composition and presentation of various types of content into web pages

- E.g. text, images, multimedia

Documents delivered through HTTP are usually expressed in HTML

- HTML5
  - Current recommendation
  - Additional markup tags
  - Extends support for multimedia and mathematical content

XML – eXtensible Markup Language

Allows for strict definition of the structure of information

- Markup tags

The RDF model also follows an XML syntax

Outline

Introduction

RDF and RDF Schema

Description Logics

Querying RDF data with SPARQL

Mapping relational data with R2RML

Other technologies

Ontologies

Datasets

Modeling Data Using RDF Graphs

Ontologies in the Semantic Web

- Model a system’s knowledge
- Based on RDF
  - Model the perception of the world as a graph
  - OWL builds on top of RDF

RDF

- A data model for the Web
- A framework that allows representing knowledge
- Model knowledge as a directed and labeled graph

RDF (1)

The cornerstone of the Semantic Web

A common representation of web resources

First publication in 1999

Can represent data from other data models

- Makes it easy to integrate data from multiple heterogeneous sources

Main idea:

- Model every resource with respect to its relations (properties) to other web resources

RDF (2)

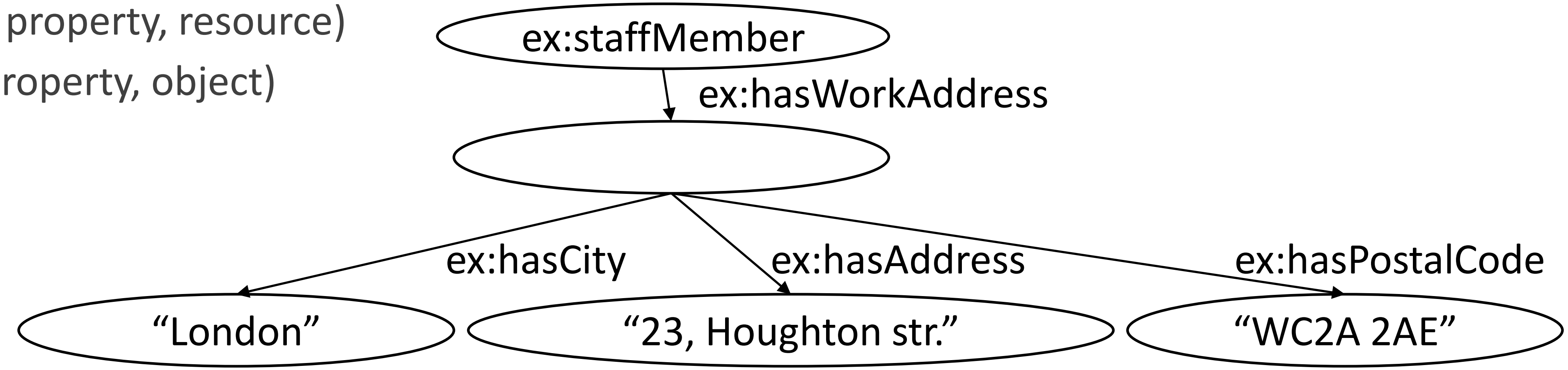
Relations form triples

- First term: subject
- Second term: property
- Third term: object

RDF statements contain triples, in the form:

(resource, property, resource)

or (subject, property, object)



Namespaces (1)

Initially designed to prevent confusion in XML names

Allow document elements to be uniquely identified

Declared in the beginning of XML documents

```
xmlns:prefix="location".
```

Namespaces (2)

Prefix	Namespace URI	Description
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	The built-in RDF vocabulary
rdfs	http://www.w3.org/2000/01/rdf-schema	RDFS puts an order to RDF
owl	http://www.w3.org/2002/07/owl	OWL terms
xsd	http://www.w3.org/2001/XMLSchema#	The RDF-compatible XML Schema datatypes
dc	http://purl.org/dc/elements/1.1/	The Dublin Core standard for digital object description
foaf	http://xmlns.com/foaf/0.1	The FOAF network
skos	http://www.w3.org/2004/02/skos/core#	Simple Knowledge Organization System
void	http://rdfs.org/ns/void#	Vocabulary of Interlinked Datasets
sioc	http://rdfs.org/sioc/ns#	Semantically-Interlinked Online Communities
cc	http://creativecommons.org/ns	Creative commons helps expressing licensing information
rdfa	http://www.w3.org/ns/rdfa	RDFa

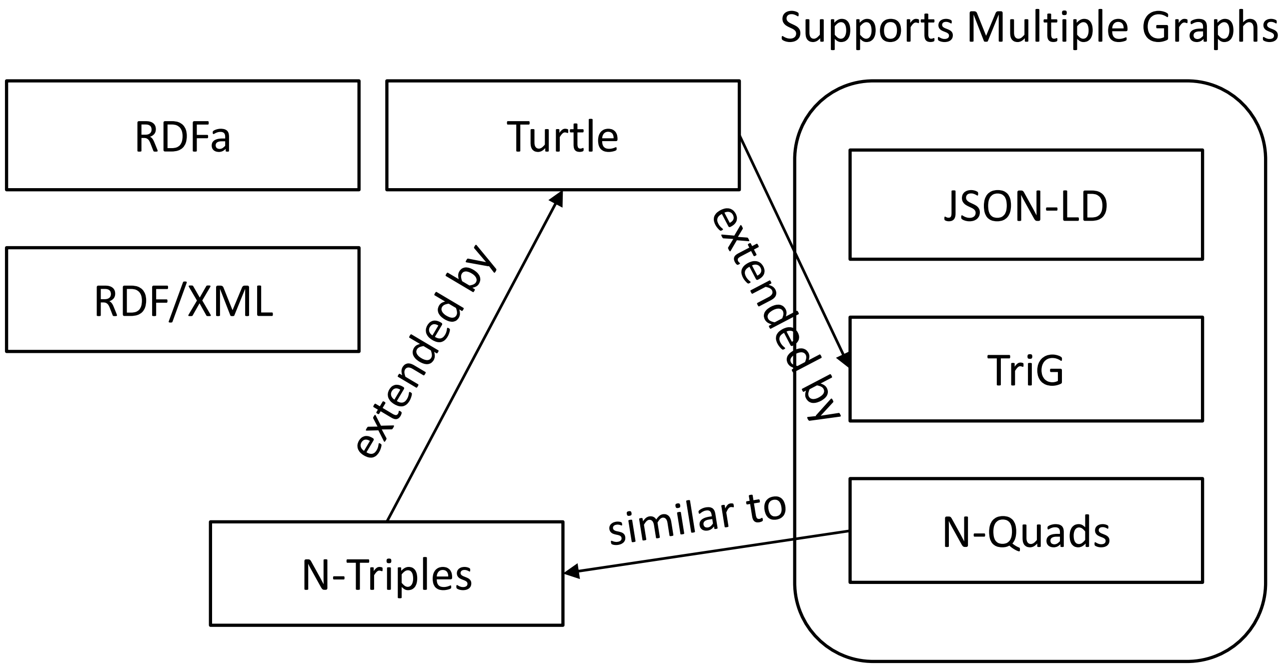
RDF Serialization

RDF is mainly destined for machine consumption

Several ways to express RDF graphs in machine-readable (serialization) formats

- N-Triples
- Turtle
- N-Quads
- TriG
- RDF/XML
- JSON-LD
- RDFa

Turtle family



N-Triples Serialization

```
<http://www.example.org/bradPitt> <http://www.example.org/isFatherOf>
<http://www.example.org/maddoxJoliePitt>.

<http://www.example.org/bradPitt> <http://xmlns.com/foaf/0.1/name> "Brad Pitt".

<http://www.example.org/bradPitt> <http://xmlns.com/foaf/0.1/based_near> :_x.

:_x <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "34.1000".

:_x <http://www.w3.org/2003/01/geo/wgs84_pos#long> "118.3333".
```

Turtle Serialization

```
PREFIX ex: <http://www.example.org/>.
PREFIX foaf: <http://xmlns.com/foaf/0.1/>.
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>.

ex:bradPitt ex:isFatherOf ex:maddoxJoliePitt;
            foaf:name "Brad Pitt";
            foaf:based_near :_x.
:_x geo:lat "34.1000";
    geo:long "118.3333".
```

TriG Serialization

```
PREFIX ex: <http://www.example.org/>.
PREFIX foaf: <http://xmlns.com/foaf/0.1/>.
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>.

GRAPH <http://www.example.org/graphs/brad> {
ex:bradPitt ex:isFatherOf ex:maddoxJoliePitt;
            foaf:name "Brad Pitt";
            foaf:based_near :_x.
:_x geo:lat "34.1000";
    geo:long "118.3333".
}
```

XML/RDF Serialization

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:ex="http://www.example.org/"
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
   <rdf:Description rdf:about=" http://www.example.org/bradPitt">
      <ex:isFatherOf rdf:resource=" http://www.example.org/maddoxJoliePitt"/>
      <foaf:name>Brad Pitt</foaf:name>
      <foaf:based_near rdf:nodeID="A0"/>
   </rdf:Description>
   <rdf:Description rdf:nodeID="A0">
      <geo:lat>34.1000</geo:lat>
      <geo:long>118.3333</geo:long>
   </rdf:Description>
</rdf:RDF>
```

JSON-LD Serialization

```
{
 "@context":{
   "foaf": "http://xmlns.com/foaf/0.1/",
   "child": {
      "@id": "http://www.example.org/isFatherOf",
      "@type": "@id"
   },
   "name": "foaf:name",
   "location": "foaf:based_near",
   "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#",
   "lat": "geo:lat",
   "long": "geo:long"
 },
 "@id": "http://www.example.org/bradPitt",
 "child": "http://www.example.org/maddoxJoliePitt",
 "name": "Brad Pitt",
 "location": {
   "lat": "34.1000",
   "long": "118.3333"
 }
}
```

RDFa Serialization

```
<html>
<head>
…
</head>
<body vocab="http://xmlns.com/foaf/0.1/">
   <div resource="http://www.example.org/bradPitt">
      <p>Famous American actor <span property="name">Brad Pitt</span>
eldest son is
 <a property="http://www.example.org/isFatheOf"
href="http://www.example.org/maddoxJoliePitt">Maddox Jolie-Pitt</a>.
      </p>
   </div>
</body>
</html>
```

The RDF Schema (1)

RDF

- A graph model
- Provides basic constructs for defining a graph structure

RDFS

- A semantic extension of RDF
- Provides mechanisms for assigning meaning to the RDF nodes and edges
- RDF Schema *is not* to RDF what XML Schema is to XML!

The RDF Schema (2)

Classes

- A definition of groups of resources
- Members of a class are called *instances* of this class

Class hierarchy

Property hierarchy

Property domain and range

RDFS specification also refers to the RDF namespace

Prefix	Namespace
rdfs	http://www.w3.org/2000/01/rdf-schema#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#

The RDF Schema (3)

A Class named Actor

```
ex:Actor rdf:type rdfs:Class.
```

- Where ex: http://www.example.org/

Class membership example

```
ex:bradPitt rdf:type ex:Actor.
```

Containment relationship

```
ex:Actor rdfs:subClassOf ex:MovieStaff.
```

Using reasoning, we can infer

```
ex:bradPitt rdf:type ex:MovieStaff.
```

- In practice, we could store inferred triples

The RDF Schema (4)

No restrictions on the form of the class hierarchy that can be defined in an ontology

- No strict tree-like hierarchy as regular taxonomies
- More complex graph-like structures are allowed
  - Classes may have more than one superclass

Allows hierarchies of properties to be defined

- Just like class hierarchies
- Introduced via the rdfs:subPropertyOf property

```
ex:participatesIn rdf:type rdf:Property.
ex:starsIn rdf:type rdf:Property.
ex:starsIn rdfs:subPropertyOf ex:participatesIn.
```

The RDF Schema (5)

```
ex:bradPitt ex:starsIn ex:worldWarZ.
```

- Inferred triple:

```
ex:bradPitt ex:participatesIn ex:worldWarZ.
```

Define the classes to which RDF properties can be applied

- rdfs:domain and rdfs:range

```
ex:starsIn rdfs:domain ex:Actor.
```

Then, the assertion

```
ex:georgeClooney ex:starsIn ex:idesOfMarch.
```

Entails

```
ex:georgeClooney rdf:type ex:Actor.
```

The RDF Schema (6)

Similarly

```
   ex:starsIn rdfs:range ex:Movie.
```

Produces

```
   ex:starsIn rdfs:range ex:Movie.
   ex:worldWarZ rdf:type ex:Movie.
```

Domain and range axioms

- Are not constraints that data need to follow
- Function as rules that lead to the production of new triples and knowledge

The RDF Schema (7)

Example

```
    ex:georgeClooney ex:starsIn ex:bradPitt.
```

- Infers:

```
    ex:bradPitt rdf:type ex:Movie.
```

An individual entity can be a member of both ex:Actor and ex:Movie classes

Such restrictions are met in more expressive ontology languages, such as OWL

Reification (1)

Statements about other RDF statements

Ability to treat an RDF statement as an RDF resource

- Make assertions about that statement

```
<http://www.example.org/person/1> foaf:name "Brad Pitt " .
```

Becomes:

```
<http://www.example.org/statement/5> a rdf:Statement;
       rdf:subject <http://www.example.org/person/1>;
       rdf:predicate foaf:name;
       rdf:object "Brad Pitt".
```

Reification (2)

Useful when referring to an RDF triple in order to

- Describe properties that apply to it
  - e.g. provenance or trust
- E.g. assign a trust level of 0.8

```
<http://www.example.org/statement/5> ex:hasTrust "0.8"^^xsd:float.
```

Classes (1)

rdfs:Resource

- All things described by RDF are instances of the class rdfs:Resource
- All classes are subclasses of this class, which is an instance of rdfs:Class

rdfs:Literal

- The class of all literals
- An instance of rdfs:Class
- Literals are represented as strings but they can be of any XSD datatype

Classes (2)

rdfs:langString

- The class of language-tagged string values
- It is a subclass of rdfs:Literal and an instance of rdfs:Datatype
- Example: "foo"@en

rdfs:Class

- The class of all classes, i.e. the class of all resources that are RDF classes

Classes (3)

rdfs:Datatype

- The class of all the data types
- An instance but also a subclass of rdfs:Class
- Every instance of rdfs:Datatype is also a subclass of rdfs:Literal

rdf:HTML

- The class of HTML literal values
- An instance of rdfs:Datatype
- A subclass of rdfs:Literal

Classes (4)

rdf:XMLLiteral

- The class of XML literal values
- An instance of rdfs:Datatype
- A subclass of rdfs:Literal

rdf:Property

- The class of all RDF properties

Properties (1)

rdfs:domain

- Declares the domain of a property *P*
- The class of all the resources that can appear as *S* in a triple (*S*, *P*, *O*)

rdfs:range

- Declares the range of a property *P*
- The class of all the resources that can appear as *O* in a triple (*S*, *P*, *O*)

Properties (2)

rdf:type

- A property that is used to state that a resource is an instance of a class

rdfs:label

- A property that provides a human-readable version of a resource’s name

Properties (3)

rdfs:comment

- A property that provides a human-readable description of a resource

rdfs:subClassOf

- Corresponds a class to one of its superclasses
- A class can have more than one superclasses

rdfs:subPropertyOf

- Corresponds a property to one of its superproperties
- A property can have more than one superproperties

Container Classes and Properties (1)

rdfs:Container

- Superclass of all classes that can contain instances such as rdf:Bag, rdf:Seq and rdf:Alt

rdfs:member

- Superproperty to all the properties that declare that a resource belongs to a class that can contain instances (container)

rdfs:ContainerMembershipProperty

- A property that is used in declaring that a resource is a member of a container
- Every instance is a subproperty of rdfs:member

Container Classes and Properties (2)

rdf:Bag

- The class of unordered containers
- A subclass of rdfs:Container

rdf:Seq

- The class of ordered containers
- A subclass of rdfs:Container

rdf:Alt

- The class of containers of alternatives
- A subclass of rdfs:Container

Collections (1)

rdf:List

- The class of RDF Lists
- An instance of rdfs:Class
- Can be used to build descriptions of lists and other list-like structures

rdf:nil

- An instance of rdf:List that is an empty rdf:List

Collections (2)

rdf:first

- The first item in the subject RDF list
- An instance of rdf:Property

rdf:rest

- The rest of the subject RDF list after the first item
- An instance of rdf:Property

rdf:_1, rdf:_2, rdf:_3, etc.

- A sub-property of rdfs:member
- An instance of the class rdfs:ContainerMembershipProperty

Reification (1)

rdf:Statement

- The class of RDF statements
- An instance of rdfs:Class

rdf:subject

- The subject of the subject RDF statement
- An instance of rdf:Property

Reification (2)

rdf:predicate

- The predicate of the subject RDF statement
- An instance of rdf:Property

rdf:object

- The object of the subject RDF statement
- An instance of rdf:Property

Utility Properties

rdf:value

- Idiomatic property used for describing structured values
- An instance of rdf:Property

rdfs:seeAlso

- A property that indicates a resource that might provide additional information about the subject resource

rdfs:isDefinedBy

- A property that indicates a resource defining the subject resource
- The defining resource may be an RDF vocabulary in which the subject resource is described

Outline

Introduction

RDF and RDF Schema

Description Logics

Querying RDF data with SPARQL

Mapping relational data with R2RML

Other technologies

Ontologies

Datasets

Ontologies Based on Description Logics

Language roots are in Description Logics (DL)

DAML, OIL, and DAML + OIL

OWL and OWL 2 latest results in this direction

Formal semantics

Syntactically compatible to the RDF serializations

- Ontologies in OWL can be queried in the same approach as in RDF graphs

Description Logics (1)

Offers the language for the description and manipulation of independent individuals, roles, and concepts

There can be many DL languages

- Describe the world using formulas

Formulas constructed using

- Sets of concepts, roles, individuals
- Constructors, e.g. intersection ∧, union ∨, exists ∃, for each ∀

Description Logics (2)

Variables in DL can represent arbitrary world objects

E.g. a DL formula can declare that a number *x*, greater than zero exists:

- $\exists x{:}\mathit{greaterThan}(x; 0)$

Adding more constructors to the basic DL language increases expressiveness

- Possible to describe more complex concepts

Description Logics (3)

E.g. concept conjunction ($C \cup D$)

- $Parent = Father \cup Mother$

Expressiveness used in describing the world varies according to the subset of the language that is used

Differentiation affects

- World description capabilities
- Behavior and performance of processing algorithms

The Web Ontology Language (1)

OWL language is directly related to DL

- Different “flavors” correspond to different DL language subsets

Successor of DAML+OIL

- Creation of the OWL language officially begins with the initiation of the DAML project
- DAML, combined to OIL led to the creation of DAML + OIL
  - An extension of RDFS
- OWL is the successor of DAML + OIL

W3C recommendation, currently in version 2

The Web Ontology Language (2)

Designed in order to allow applications to process the information content itself instead of simply presenting the information

The goal is to provide a schema that will be compatible both to the Semantic Web and the World Wide Web architecture

Makes information more machine- and human- processable

The Web Ontology Language (3)

First version comprised three flavors

- Lite, DL, Full

OWL Lite

- Designed keeping in mind that it had to resemble RDFS

OWL DL

- Guaranteed that all reasoning procedures are finite and return a result

OWL Full

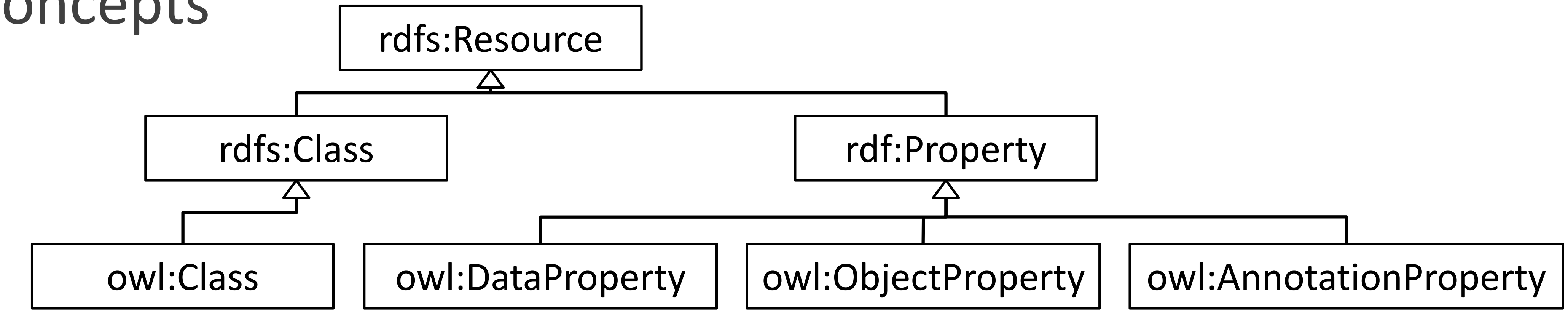
- The whole wealth and expressiveness of the language
- Reasoning is not guaranteed to be finite
  - Even for small declaration sets

The Web Ontology Language (4)

An OWL ontology may also comprise declarations from RDF and RDFS

- E.g. rdfs:subClassOf, rdfs:range, rdf:resource

OWL uses them and relies on them in order to model its concepts



OWL 2

Very similar overall structure to the first version of OWL

Backwards compatible

- Every OWL 1 ontology is also an OWL 2 ontology

OWL 2 Additional Features (1)

Additional property and qualified cardinality constructors

- Minimum, maximum or exact qualified cardinality restrictions, object and data properties
  - E.g. ObjectMinCardinality, DataMaxCardinality

Property chains

- Properties as a composition of other properties
  - E.g. ObjectPropertyChain in SubObjectPropertyOf

OWL 2 Additional Features (2)

Extended datatype support

- Support the definition of subsets of datatypes (e.g. integers and strings)
  - E.g. state that every person has an age, which is of type integer, and restrict the range of that datatype value

Simple metamodeling

- Relaxed separation between the names of, e.g. classes and individuals
  - Allow different uses of the same term

OWL 2 Additional Features (3)

Extended annotations

- Allow annotations of axioms
    - E.g. who asserted an axiom or when

Extra syntactic sugar

- Easier to write common patterns
    - Without changing language expressiveness, semantics, or complexity

OWL 2 Profiles (1)

OWL 2 EL

- Polynomial time algorithms for all standard reasoning tasks
- Suitable for very large ontologies
- Trades expressive power for performance guarantees

OWL 2 Profiles (2)

OWL 2 QL

- Conjunctive queries are answered in LOGSPACE using standard relational database technology
- Suitable for
  - Relatively lightweight ontologies with large numbers of individuals
  - Useful or necessary to access the data via relational queries (e.g. SQL)

OWL 2 Profiles (3)

OWL 2 RL

- Enables polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples
- Suitable for
  - Relatively lightweight ontologies with large numbers of individuals
  - Necessary to operate directly on data in the form of RDF triples

Manchester OWL syntax

A user-friendly syntax for OWL 2 descriptions

```
Class: VegetarianPizza
EquivalentTo:
Pizza and
not (hasTopping some FishTopping) and
not (hasTopping some MeatTopping)

DisjointWith:
NonVegetarianPizza
```

Outline

Introduction

RDF and RDF Schema

Description Logics

Querying RDF data with SPARQL

Mapping relational data with R2RML

Other technologies

Ontologies

Datasets

Querying the Semantic Web with SPARQL

SPARQL

- Is for RDF what SQL is for relational databases

SPARQL family of recommendations

- SPARQL Update
- SPARQL 1.1 Protocol
- Graph Store HTTP Protocol
- SPARQL 1.1 entailment regimes
- Specifications about the serialization of query results
  - In JSON, CSV and TSV, and XML

SELECT Queries (1)

Return a set of bindings of variables to RDF terms

Binding

- A mapping from a variable to a URI or an RDF literal

Triple pattern

- Resembles an RDF triple
- May also contain variables in one or more positions

SELECT Queries (2)

An RDF triple *matches* a triple pattern

- There is an appropriate substitution of variables with RDF terms that makes the triple pattern and the RDF triple equivalent
- Example:

```
   ex:bradPitt rdf:type ex:Actor.
```

matches

```
   ?x rdf:type ex:Actor
```

SELECT Queries (3)

A set of triple patterns build a *basic graph pattern*

- The heart of a SPARQL SELECT query

```
SELECT ?x ?date
WHERE {
   ?x rdf:type ex:Actor .
   ?x ex:bornIn ?date
}
```

SELECT Queries (4)

Graph example

```
ex:bradPitt rdf:type ex:Actor;
         ex:bornIn "1963"^^xsd:gYear.
ex:angelinaJolie rdf:type ex:Actor;
              ex:bornIn "1975"^^xsd:gYear.
ex:jenniferAniston rdf:type ex:Actor.
```

SPARQL query solution

?x	**?date**
ex:bradPitt	"1963"^^xsd:gYear
ex:angelinaJolie	"1975"^^xsd:gYear

The OPTIONAL keyword

Resource ex:jenniferAniston not included in the results

- No RDF triple matches the second triple pattern

Use of the OPTIONAL keyword

- Retrieves all actors and actresses and get their birthdate only if it is specified in the RDF graph

```
SELECT ?x ?date
WHERE {
     ?x rdf:type ex:Actor .
     OPTIONAL {?x ex:bornIn ?date}
}
```

?x	?date
ex:bradPitt	"1963"^^xsd:gYear
ex:angelinaJolie	"1975"^^xsd:gYear
ex:jenniferAniston	

The UNION Keyword

Specify alternative graph patterns

```
SELECT ?x
WHERE {
   {?x rdf:type ex:Actor}
   UNION
   {?x rdf:type ex:Director}
}
```

Search for resources that are of type ex:Actor or ex:Director, including cases where a resource may belong to both classes

The FILTER Keyword

Set a condition that limits the result of a SPARQL query

```
SELECT ?x ?date
WHERE{
   ?x rdf:type ex:Actor .
   ?x ex:bornIn ?date .
   FILTER(?date < "1975"^^xsd:gYear)
}
```

Return all actors that were known to be born before 1975 and their corresponding birth date

ORDER BY and LIMIT Keywords

```
SELECT ?x ?date
WHERE {
   ?x rdf:type ex:Actor .
   ?x ex:bornIn ?date .
}
ORDER BY DESC(?date)
LIMIT 10
```

CONSTRUCT Queries

Generate an RDF graph based on a graph template using the solutions of a SELECT query

```
CONSTRUCT {?x rdf:type ex:HighlyPaidActor}
WHERE {
   ?x rdf:type ex:Actor .
   ?x ex:hasSalary ?salary .
   FILTER (?salary > 1000000)
}
```

ASK Queries

Respond with a boolean

Whether a graph pattern is satisfied by a subgraph of the considered RDF graph

```
ASK {
   ?x rdf:type ex:Actor .
   ?x foaf:name "Cate Blanchett" .
}
```

Return true if the RDF graph contains an actor named "Cate Blanchett"

DESCRIBE Queries

Return a set of RDF statements that contain data about a given resource

Exact structure of the returned RDF graph depends on the specific SPARQL engine

```
DESCRIBE <http://www.example.org/bradPitt> .
```

Outline

Introduction

RDF and RDF Schema

Description Logics

Querying RDF data with SPARQL

Mapping relational data with R2RML

Other technologies

Ontologies

Datasets

Mapping Relational Data to RDF (1)

Lack of RDF data volumes ⟶ Slow uptake of the Semantic Web vision

Large part of available data is stored in relational databases

Several methods and tools were developed for the translation of relational data to RDF

- Each one with its own set of features and, unfortunately, its own mapping language

The need for the reuse of RDB-to-RDF mappings led to the creation of R2RML

- A standard formalism by W3C for expressing such mappings

Mapping Relational Data to RDF (2)

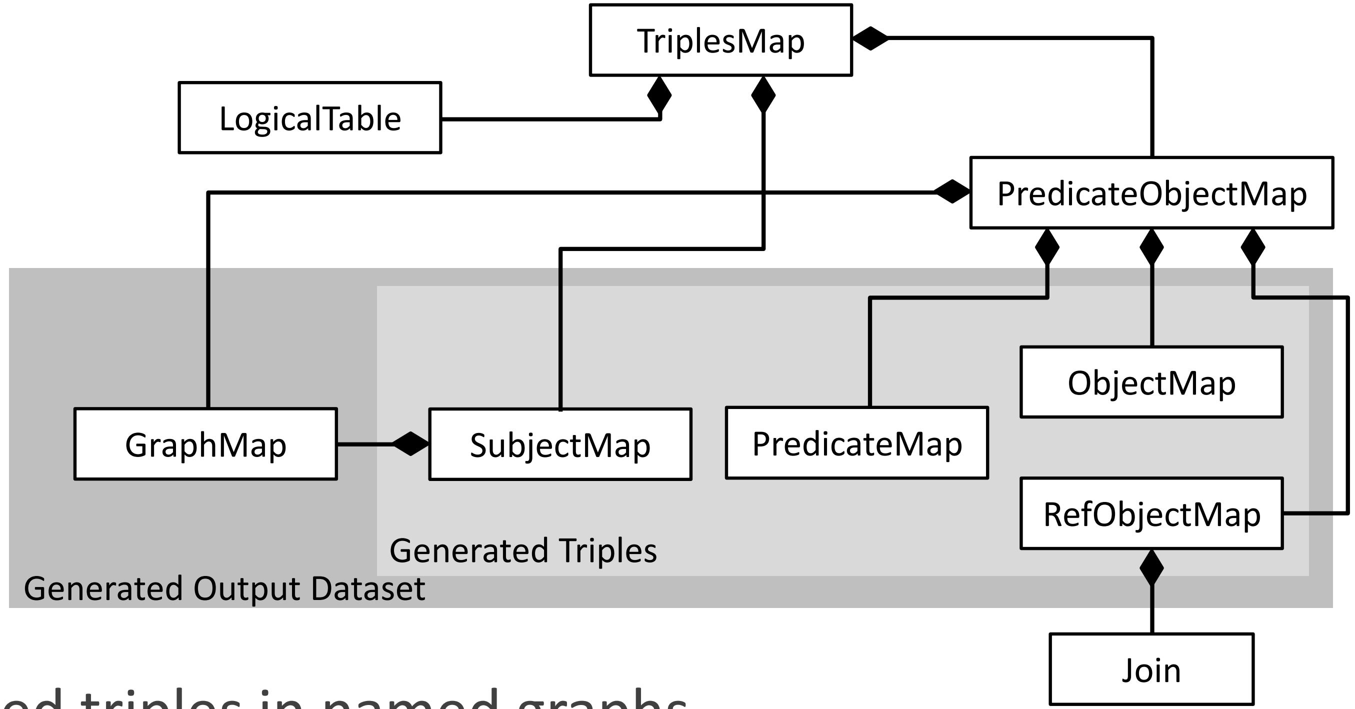
R2RML: RDB to RDF Mapping Language

- Provides a vocabulary for the definition of RDF views over relational schemas
- Is database vendor-agnostic

An R2RML mapping is an RDF graph: an *R2RML mapping graph*

- In Turtle syntax

R2RML Overview



More features

- Organization of generated triples in named graphs
- Definition of blank nodes
- Specification of a generated literal’s language

R2RML (1)

An R2RML mapping

- Associates relational views with functions that generate RDF terms

Logical tables

- Relations or (custom) views defined in the relational schema

Term maps

- RDF generating functions
- Distinguished according to the position of the RDF term in the generated triple
  - Subject, predicate, and object maps

R2RML (2)

Triples maps

- Functions that map relational data to a set of RDF triples
- Groups of term maps
- R2RML mappings contain one or more triples maps

Graph maps

- Generated RDF triples can be organized into named graphs

R2RML Example (1)

A relational instance

FILM

ID	*Title*	*Year*	*Director*
1	The Hunger Games	2012	1

DIRECTOR

ID	*Name*	*BirthYear*
1	Gary Ross	1956

ACTOR

ID	*Name*	*BirthYear*	*BirthLocation*
1	Jennifer Lawrence	1990	Louisville, KY
2	Josh Hutcherson	1992	Union, KY

FILM2ACTOR

FilmID	*ActorID*
1	1
1	2

R2RML Example (2)

An R2RML triples map

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://www.example.org/>.
@prefix dc: <http://purl.org/dc/terms/>.

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "FILM" ];
    rr:subjectMap [
        rr:template "http://data.example.org/film/{ID}";
        rr:class ex:Movie;
    ];
    rr:predicateObjectMap [
        rr:predicate dc:title;
        rr:objectMap [ rr:column "Title" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:releasedIn;
        rr:objectMap [ rr:column "Year";
                       rr:datatype xsd:gYear;];
].
```

R2RML Example (3)

The result (in Turtle)

```
<http://data.example.org/film/1> a ex:Movie;
              dc:title "The Hunger Games";
              ex:releasedIn "2012"^^xsd:gYear.
```

Note the reuse of terms from external ontologies

- E.g. dc:title from Dublin Core

R2RML Example (4)

The R2RML mapping graph of the example

- Specifies the generation of a set of RDF triples for every row of the FILM relation
- Is the simplest possible
  - Contains only one triples map

Every triples map must have exactly

- One logical table
- One subject map
- One or more predicate-object maps

R2RML Example (5)

A logical table represents an SQL result set

- Each row gives rise to an RDF triple
  - Or quad in the general case, when named graphs are used

A subject map is simply a term map that produces the subject of an RDF triple

R2RML Example (6)

Term maps

- Template-valued (rr:template)
  - The subject map of the example
- Constant-valued (rr:predicate)
  - The predicate map of the example
- Column-valued (rr:column)
  - The object map of the example

Every generated resource will be an instance of ex:Movie

- Use of rr:class

R2RML Example (7)

Subject maps

- Generate subjects

Predicate-object maps

- Generate pairs of predicates and objects
- Contain at least one predicate map
- Contain at least one object map

Typed Literals

- Use of rr:datatype
- The second object map of the example specifies that the datatype of the RDF literal that will be generated will be xsd:gYear

Foreign Keys (1)

Add another predicate-object map that operates on the DIRECTOR relation

Specify the generation of three RDF triples per row of the DIRECTOR relation

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://www.example.org/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

<#TriplesMap2>
    rr:logicalTable [ rr:tableName "DIRECTOR" ];
    rr:subjectMap [
        rr:template "http://data.example.org/director/{ID}";
        rr:class ex:Director;
    ];
    rr:predicateObjectMap [
        rr:predicate foaf:name;
        rr:objectMap [ rr:column "Name" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:bornIn;
        rr:objectMap [ rr:column "BirthYear";
                       rr:datatype xsd:gYear;];
    ].
```

Foreign Keys (2)

Add another predicate-object map, the referencing object map

Link entities described in separate tables

Exploit the foreign key relationship of FILM.Director and DIRECTOR.ID

```
<#TriplesMap1>
    rr:logicalTable [ rr:tableName "FILM" ];
    rr:subjectMap [
        rr:template "http://data.example.org/film/{ID}";
        rr:class ex:Movie;
    ];
    rr:predicateObjectMap [
        rr:predicate dc:title;
        rr:objectMap [ rr:column "Title" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:releasedIn;
        rr:objectMap [ rr:column "Year";
                       rr:datatype xsd:gYear;];

    rr:predicateObjectMap[
        rr:predicate ex:directedBy;
        rr:objectMap[
                 rr:parentTriplesMap <#TriplesMap2>;
                 rr:joinCondition[
                             rr:child "Director";
                             rr:parent "ID";
                                  ];
                     ];
    ].
```

Foreign Keys (3)

TriplesMap1 now contains a *referencing object map*

- Responsible for the generation of the object of a triple
- Referencing object maps are a special case of object maps
- Follows the generation rules of the subject map of another triples map
  - Called *parent triples* map
- Join conditions are also specified in order to select the appropriate row of the logical table of the *parent triples* map

Foreign Keys (4)

Not necessary for this foreign key relationship to be explicitly defined as a constraint in the relational schema

R2RML is flexible enough to allow the linkage of any column set among different relations

TriplesMap1 result

```
<http://data.example.org/film/1> ex:directedBy <http://data.example.org/director/1>.
```

TriplesMap2 result

```
<http://data.example.org/director/1> a ex:Director;
    foaf:name "Gary Ross";
    ex:bornIn "1956"^^xsd:gYear.
```

Custom Views (1)

R2RML view defined via the rr:sqlQuery property

Used just as the native logical tables

Could also have been defined as an SQL view in the underlying database system

- Not always feasible/desirable

Produced result set must not contain columns with the same name

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ex: <http://www.example.org/>.
@prefix dc: <http://purl.org/dc/terms/>.

<#TriplesMap3>

    rr:logicalTable [ rr:sqlQuery """
                      SELECT ACTOR.ID AS ActorId, ACTOR.Name AS
ActorName, ACTOR.BirthYear AS ActorBirth, FILM.ID AS FilmId FROM
ACTOR, FILM, FILM2ACTOR WHERE FILM.ID=FILM2ACTOR.FilmID AND
ACTOR.ID=FILM2ACTOR.ActorID; """];

    rr:subjectMap [

        rr:template "http://data.example.org/actor/{ActorId}";

        rr:class ex:Actor;
    ];
    rr:predicateObjectMap [
        rr:predicate foaf:name;
        rr:objectMap [ rr:column "ActorName" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:bornIn;
        rr:objectMap [ rr:column "ActorBirth";
                       rr:datatype xsd:gYear;];
    ].
    rr:predicateObjectMap [
        rr:predicate ex:starsIn;
        rr:objectMap [ rr:template
"http://data.example.org/film/{ID}";
                       ];
    ].
```

Custom Views (2)

TriplesMap3 generates the following triples

```
<http://data.example.org/actor/1> a ex:Actor;
    foaf:name "Jennifer Lawrence";
    ex:bornIn "1990"^^xsd:gYear;
  ex:starsIn<http://data.example.org/film/1>.
<http://data.example.org/actor/2> a ex:Actor;
    foaf:name "Josh Hutcherson";
    ex:bornIn "1992"^^xsd:gYear;
    ex:starsIn <http://data.example.org/film/1>.
```

Direct Mapping (1)

A default strategy for translating a relational instance to an RDF graph

A trivial transformation strategy

A recommendation by W3C since 2012

Defines a standard URI generation policy for all kinds of relations

Direct Mapping (2)

Maps every relation to a new ontology class and every relation row to an instance of the respective class

The generated RDF graph essentially mirrors the relational structure

Can be useful as a starting point for mapping authors who can then augment and customize it accordingly using R2RML

Outline

Introduction

RDF and RDF Schema

Description Logics

Querying RDF data with SPARQL

Mapping relational data with R2RML

Other technologies

Ontologies

Datasets

POWDER – Protocol for Web Description Resources (1)

XML-based protocol

Allows provision of information about RDF resources

A POWDER document contains

- Its own provenance information
  - Information about its creator, when it was created, etc.
- A list of description resources

POWDER – Protocol for Web Description Resources (2)

Description resource

- Contains a set of property-value pairs in RDF
- For a given resource or group of resources

Allows for quick annotation of large amounts of content with metadata

Ideal for scenarios involving personalized delivery of content, trust control and semantic annotation

RIF – Rule Interchange Format

An extensible framework for the representation of rule dialects

A W3C recommendation

A family of specifications

- Three rule dialects
  - RIF-BLD (Basic Logic Dialect)
  - RIF-Core and
  - RIF-PRD (Production Rule Dialect)
- RIF-FLD (Framework for Logic Dialects)
  - Allows the definition of other rule dialects
  - Specifies the interface of rules expressed in a logic-based RIF dialect with RDF and OWL

SPIN – SPARQL Inferencing Notation

RIF uptake not as massive as expected

- Complexity, lack of supporting tools

SPIN

- SPARQL-based
- Production rules

Links RDFS and OWL classes with constraint checks and inference rules

- Similar to class behavior in object-oriented languages

GRDDL – Gleaning Resource Descriptions from Dialects of Languages

A W3C recommendation

Mechanisms that specify how to construct RDF representations

- Can be applied to XHTML and XML documents
- Defines XSLT transformations that generate RDF/XML

Is the standard way of converting an XML document to RDF

Familiarity with XSLT required

LDP – Linked Data Platform

Best practices and guidelines for organizing and accessing LDP resources

- Via HTTP RESTful services
- Resources can be RDF or non-RDF

A W3C recommendation

- Specifies the expected behavior of a Linked Data server when it receives HTTP requests
  - Creation, deletion, update
- Defines the concept of a Linked Data Platform Container
  - A collection of resources (usually homogeneous)

LDP specification expected to establish a standard behavior for Linked Data systems

Example: Apache Marmotta

Outline

Introduction

RDF and RDF Schema

Description Logics

Querying RDF data with SPARQL

Mapping relational data with R2RML

Other technologies

Ontologies

Datasets

Ontologies and Datasets (1)

Standards and technologies (e.g. RDF and OWL)

- Machine-consumable serializations
- Syntactical groundwork
- Assignment of meaning to resources and their relationships

Vocabularies

- (Re)used in order for the described information to be commonly, unambiguously interpreted and understood
- Reused by various data producers
- Make data semantically interoperable

Ontologies and Datasets (2)

Datasets

- Collections of facts involving specific individual entities and ontologies, which provide an abstract, high-level, terminological description of a domain, containing axioms that hold for every individual

A great number of ontologies exist nowadays

- Subject domains: e.g. geographical, business, life sciences, literature, media

Ontology search engines and specialized directories

Schemapedia

Watson

Swoogle

Linked Open Vocabularies (LOV)

- The most accurate and comprehensive source of ontologies used in the Linked Data cloud
- Vocabularies described by appropriate metadata and classified to domain spaces
- Full-text search enabled at vocabulary and element level

DC – Dublin Core (1)

A minimal metadata element set

Mainly used for the description of web resources

DC vocabulary

- Available as an RDFS ontology
- Contains classes and properties
  - E.g. agent, bibliographic resource, creator, title, publisher, description

DC – Dublin Core (2)

DC ontology partitioned in two sets

- Simple DC
  - Contains 15 core properties
- Qualified DC
  - Contains all the classes and the rest of the properties (specializations of those in simple DC)

FOAF – Friend-Of-A-Friend

One of the first lightweight ontologies being developed since the first years of the Semantic Web

Describes human social networks

Classes such as Person, Agent or Organization

Properties denoting personal details and relationships with other persons and entities

SKOS – Simple Knowledge Organization System (1)

A fundamental vocabulary in the Linked Data ecosystem

A W3C recommendation since 2009

Contains terms for organizing knowledge

- In the form of taxonomies, thesauri, and concept hierarchies

Defines concept schemes as sets of concepts

- Concept schemes can relate to each other through “broader/narrower than” or equivalence relationships

SKOS – Simple Knowledge Organization System (2)

Defined as an OWL Full ontology

- Has its own semantics
  - Its own set of entailment rules distinct from those of RDFS and OWL

Widely used in the library and information science domain

VoID – Vocabulary of Interlinked Datasets

Metadata for RDF datasets

- General metadata
  - E.g. Name, description, creator
- Information on the ways that this dataset can be accessed
  - E.g. as a data dump, via a SPARQL endpoint
- Structural information
  - E.g. the set of vocabularies that are used or the pattern of a typical URI
- Information on the links
  - E.g. the number and target datasets

Aid users and machines to decide whether a dataset is suitable for their needs

SIOC – Semantically-Interlinked Online Communities

An OWL ontology

Describes online communities

- E.g. forums, blogs, mailing lists

Main classes

- E.g. forum, post, event, group, user

Properties

- Attributes of those classes
- E.g. the topic or the number of views of a post

Good Relations

Describes online commercial offerings

- E.g. Product and business descriptions, pricing and delivery methods

Great impact in real-life applications

- Adoption by several online retailers and search engines

Search engines able to interpret product metadata expressed in the Good Relations vocabulary

- Offer results better tailored to the needs of end users

Outline

Introduction

RDF and RDF Schema

Description Logics

Querying RDF data with SPARQL

Mapping relational data with R2RML

Other technologies

Ontologies

Datasets

Datasets

Several “thematic neighborhoods” in the Linked Data cloud

- E.g. government, media, geography, life sciences

Single-domain and cross-domain datasets

DBpedia (1)

The RDF version of Wikipedia

- Perhaps the most popular RDF dataset in the LOD cloud
- A large number of incoming links

A cross-domain dataset

- Assigns a URI to every resource described by a Wikipedia article
- Produces structured information by mining information from Wikipedia infoboxes

DBpedia (2)

A multilingual dataset

- Transforms various language editions of Wikipedia

The DBpedia dataset offered as data dump and through a SPARQL endpoint

A “live” version available, updated whenever a Wikipedia page is updated

Freebase

Openly-licensed, structured dataset, also available as an RDF graph

Tens of millions of concepts, types and properties

Can be edited by anyone

Gathers information from several structured sources and free text

Offers an API for developers

Linked to DBpedia through incoming and outgoing links

GeoNames

An open geographical database

Millions of geographical places

Can be edited by anyone

Geographical information as the context of descriptions and facts

Omnipresent in the Linked Data cloud

Several incoming links from other datasets

Lexvo

Central dataset for the so-called linguistic Linked Data Cloud

Provides identifiers for thousands of languages

URIs for terms in every language

Identifiers for language characters

Chapter 3

Deploying Linked Open Data

Methodologies and Software Tools

NIKOLAOS KONSTANTINOU

DIMITRIOS-EMMANUEL SPANOS

Outline

Introduction

Modeling Data

Software for Working with Linked Data

Software Tools for Storing and Processing Linked Data

Tools for Linking and Aligning Linked Data

Software Libraries for working with RDF

Introduction

Today’s Web: Anyone can say anything about any topic

- Information on the Web cannot always be trusted

Linked Open Data (LOD) approach

- Materializes the Semantic Web vision
- A focal point is provided for any given web resource
  - Referencing (referring to)
  - De-referencing (retrieving data about)

Not All Data Can Be Published Online

Data has to be

- Stand-alone
  - Strictly separated from business logic, formatting, presentation processing
- Adequately described
  - Use well-known vocabularies to describe it, or
  - Provide de-referenceable URIs with vocabulary term definitions
- Linked to other datasets
- Accessed simply
  - HTTP and RDF instead of Web APIs

Linked Data-driven Applications (1)

Content reuse

- E.g. BBC’s Music Store
  - Uses DBpedia and MusicBrainz

Semantic tagging and rating

- E.g. Faviki
  - Uses DBpedia

Linked Data-driven Applications (2)

Integrated question-answering

- E.g. DBpedia mobile
  - Indicate locations in the user's vicinity

Event data management

- E.g. Virtuoso's calendar module
  - Can organize events, tasks, and notes

Linked Data-driven Applications (3)

Linked Data-driven data webs are expected to evolve in numerous domains

- E.g. Biology, software engineering

The bulk of Linked Data processing is not done online

Traditional applications use other technologies

- E.g. relational databases, spreadsheets, XML files
- Data must be transformed in order to be published on the web

The O in LOD: Open Data

Open ≠ Linked

- Open data is data that is publicly accessible via internet
  - No physical or virtual barriers to accessing them
- Linked Data allows relationships to be expressed among these data

RDF is ideal for representing Linked Data

- This contributes to the misconception that LOD can only be published in RDF

Definition of openness by www.opendefinition.org

> Open means anyone can freely access, use, modify, and share for any purpose (subject, at most, to requirements that preserve provenance and openness)

Why should anyone open their data?

Reluctance by data owners

- Fear of becoming useless by giving away their core value

In practice the opposite happens

- Allowing access to content leverages its value
  - Added-value services and products by third parties and interested audience
  - Discovery of mistakes and inconsistencies
    - People can verify convent freshness, completeness, accuracy, integrity, overall value
  - In specific domains, data have to be open for strategic reasons
    - E.g. transparency in government data

Steps in Publishing Linked Open Data (1)

Data should be kept simple

- Start small and fast
- Not all data is required to be opened at once
- Start by opening up just one dataset, or part of a larger dataset
- Open up more datasets
  - Experience and momentum may be gained
  - Risk of unnecessary spending of resources
    - Not every dataset is useful

Steps in Publishing Linked Open Data (2)

Engage early and engage often

- Know your audience
- Take its feedback into account
- Ensure that next iteration of the service will be as relevant as it can be
- End users will not always be direct consumers of the data
    - It is likely that intermediaries will come between data providers and end users
        - E.g. an end user will not find use in an array of geographical coordinates but a company offering maps will
    - Engage with the intermediaries
        - They will reuse and repurpose the data

Steps in Publishing Linked Open Data (3)

Deal in advance with common fears and misunderstandings

- Opening data is not always looked upon favorably
- Especially in large institutions, it will entail a series of consequences and, respectively, opposition
- Identify, explain, and deal with the most important fears and probable misconceptions from an early stage

Steps in Publishing Linked Open Data (4)

It is fine to charge for access to the data via an API

- As long as the data itself is provided in bulk for free
- Data can be considered as open
  - The API is considered as an added-value service on top of the data
  - Fees are charged for the use of the API, not of the data
- This opens business opportunities in the data-value chain around open data

Steps in Publishing Linked Open Data (5)

Data openness ≠ data freshness

- Opened data does not have to be a real-time snapshot of the system data
  - Consolidate data into bulks asynchronously
  - E.g. every hour or every day
  - You could offer bulk access to the data dump and access through an API to the real-time data

Dataset Metadata (1)

Provenance

- Information about entities, activities and people involved in the creation of a dataset, a piece of software, a tangible object, a thing in general
- Can be used in order to assess the thing’s quality, reliability, trustworthiness, etc.
- Two related recommendations by W3C
  - The PROV Data Model, in OWL 2
  - The PROV ontology

Dataset Metadata (2)

Description about the dataset

W3C recommendation

- DCAT
  - Describes an RDF vocabulary
  - Specifically designed to facilitate interoperability between data catalogs published on the Web

Dataset Metadata (3)

Licensing

- A short description regarding the terms of use of the dataset
- E.g. for the Open Data Commons Attribution License

This {DATA(BASE)-NAME} is made available under the Open Data Commons Attribution License: http://opendatacommons.org/licenses/by/{version }.—See more at: http://opendatacommons.org/licenses/by/#sthash.9HadQzSW.dpuf

Bulk Access vs. API (1)

Offering bulk access is a requirement

- Offering an API is not

Bulk Access vs. API (2)

Bulk access

- Can be cheaper than providing an API
  - Even an elementary API entails development and maintenance costs
- Allows building an API on top of the offered data
  - Offering an API does not allow clients to retrieve the whole amount of data
- Guarantees full access to the data
  - An API does not

Bulk Access vs. API (3)

API

- More suitable for large volumes of data
  - No need to download the whole dataset when a small subset is needed

The 5-Star Deployment Scheme

★	Data is made available on the Web (whatever format) but with an open license to be Open Data
★★	Available as machine-readable structured data: e.g. an Excel spreadsheet instead of image scan of a table
★★★	As the 2-star approach, in a non-proprietary format: e.g. CSV instead of Excel
★★★★	All the above plus the use of open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff
★★★★★	All the above, plus: Links from the data to other people’s data in order to provide context

Outline

Introduction

Modeling Data

Software Tools for Storing and Processing Linked Data

Tools for Linking and Aligning Linked Data

Software Libraries for working with RDF

The D in LOD: Modeling Content

Content has to comply with a specific model

- A model can be used
  - As a mediator among multiple viewpoints
  - As an interfacing mechanism between humans or computers
  - To offer analytics and predictions
- Expressed in RDF(S), OWL
- Custom or reusing existing vocabularies
- Decide on the ontology that will serve as a model
  - Among the first decisions when publishing a dataset as LOD
- Complexity of the model has to be taken into account, based on the desired properties
  - Decide whether RDFS or one of the OWL profiles (flavors) is needed

Reusing Existing Works (1)

Vocabularies and ontologies have existed long before the emergence of the Web

- Widespread vocabularies and ontologies in several domains encode the accumulated knowledge and experience

Highly probable that a vocabulary has already been created in order to describe the involved concepts

- Any domain of interest

Reusing Existing Works (2)

Increased interoperability

- Use of standards can help content aggregators to parse and process the information
  - Without much extra effort per data source
- E.g. an aggregator that parses and processes dates from several sources
  - More likely to support the standard date formats
  - Less likely to convert the formatting from each source to a uniform syntax
    - Much extra effort per data source
  - E.g. DCMI Metadata Terms
    - Field dcterms:created, value "2014-11-07"^^xsd:date

Reusing Existing Works (3)

Credibility

- Shows that the published dataset
  - Has been well thought of
  - Is curated
- A state-of-the-art survey has been performed prior to publishing the data

Ease of use

- Reusing is easier than rethinking and implementing again or replicating existing solutions
  - Even more, when vocabularies are published by multidisciplinary consortia with potentially more spherical view on the domain than yours

Reusing Existing Works (4)

In conclusion

- Before adding terms in our vocabulary, make sure they do not already exist
  - In such case, reuse them by reference
- When we need to be more specific, we can create a subclass or a subproperty of the existing
- New terms can be generated, when the existing ones do not suffice

Semantic Web for Content Modeling

Powerful means for system description

- Concept hierarchy, property hierarchy, set of individuals, etc.

Beyond description

- Model checking
    - Use of a reasoner assures creation of coherent, consistent models
- Semantic interoperability
- Inference
- Formally defined semantics
- Support of rules
- Support of logic programming

Assigning URIs to Entities

Descriptions can be provided for

- Things that exist online
- Items/persons/ideas/things (in general) that exist outside of the Web

Example: Two URIs to describe a company

- The company’s website
- A description of the company itself
  - May well be in an RDF document

A strategy has to be devised in assigning URIs to entities

- No deterministic approaches

Assigning URIs to Entities: Challenges

Dealing with ungrounded data

Lack of reconciliation options

Lack of identifier scheme documentation

Proprietary identifier schemes

Multiple identifiers for the same concepts/entities

Inability to resolve identifiers

Fragile identifiers

Assigning URIs to Entities: Benefits

Semantic annotation

Data is discoverable and citable

The value of the data increases as the usage of its identifiers increases

URI Design Patterns (1)

Conventions for how URIs will be assigned to resources

Also widely used in modern web frameworks

In general applicable to web applications

Can be combined

Can evolve and be extended over time

Their use is not restrictive

- Each dataset has its own characteristics

Some upfront thought about identifiers is always beneficial

URI Design Patterns (2)

Hierarchical URIs

- URIs assigned to a group of resources that form a natural hierarchy
  - E.g. :collection/:item/:sub-collection/:item

Natural keys

- URIs created from data that already has unique identifiers
  - E.g. identify books using their ISBN

URI Design Patterns (3)

Literal keys

- URIs created from existing, non-global identifiers
  - E.g. the dc:identifier property of the described resource

Patterned URIs

- More predictable, human-readable URIs
- E.g. /books/12345
  - /books is the base part of the URI indicating “the collection of books”
  - 12345 is an identifier for an individual book

URI Design Patterns (4)

Proxy URIs

- Used in order to deal with the lack of standard identifiers for third-party resources
- If for these resources, identifiers do exist, then these should be reused
  - If not, use locally minted Proxy URIs

Rebased URIs

- URIs constructed based on other URIs
  - E.g. URIs rewritten using regular expressions
  - From http://graph1.example.org/document/1 to http://graph2.example.org/document/1

URI Design Patterns (5)

Shared keys

- URIs specifically designed to simplify the linking task between datasets
- Achieved by a creating *Patterned URIs* while applying the *Natural Keys* pattern
- Public, standard identifiers are preferable to internal, system-specific

URL slugs

- URIs created from arbitrary text or keywords, following a certain algorithm
- E.g. lowercasing the text, removing special characters and replacing spaces with a dash
  - A URI for the name “Brad Pitt” could be http://www.example.org/brad-pitt

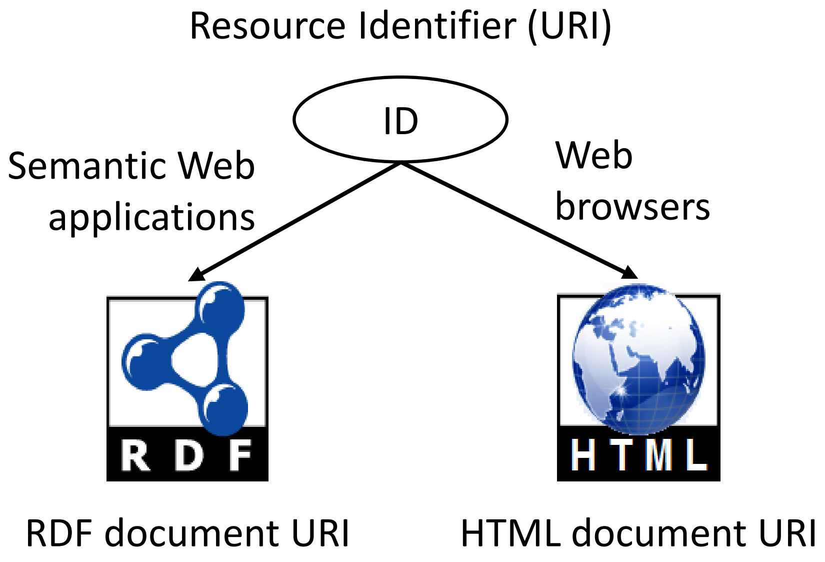
Assigning URIs to Entities

Desired functionality

- Semantic Web applications retrieve the RDF description of things
- Web browsers are directed to the (HTML) documents describing the same resource

Two categories of technical approaches for providing URIs for dataset entities

- Hash URIs
- 303 URIs



Hash URIs

URIs contain a fragment separated from the rest of the URI using ‘#’

- E.g. URIs for the descriptions of two companies
  - http://www.example.org/info#alpha
  - http://www.example.org/info#beta
- The RDF document containing descriptions about both companies
  - http://www.example.org/info
- The original URIs will be used in this RDF document to uniquely identify the resources
  - Companies Alpha, Beta and anything else

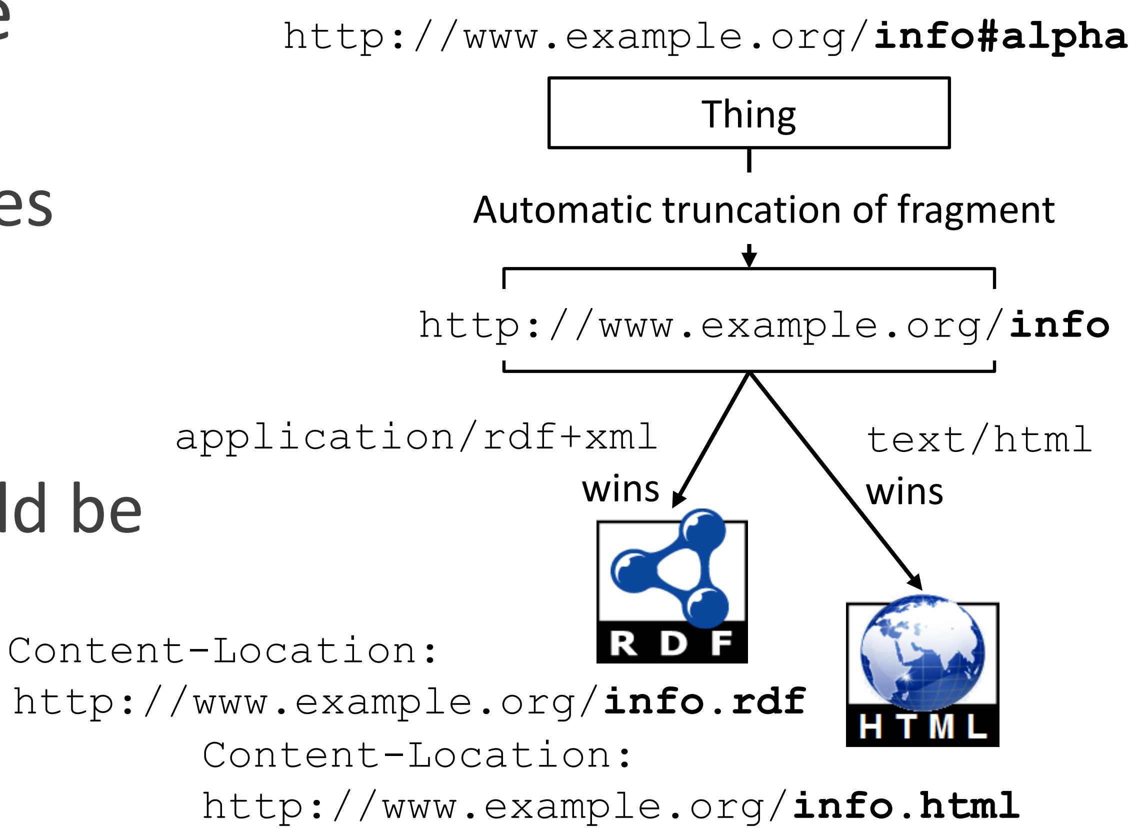
Hash URIs with Content Negotiation

Redirect either to the RDF or the HTML representation

- Decision based on client preferences and server configuration

Technically

- The Content-Location header should be set to indicate where the hash URI refers to
  - Part of the RDF document (info.rdf)
  - Part of the HTML document (info.html)


http://www.example.org/info#alpha
Thing
Automatic truncation of fragment
http://www.example.org/info
application/rdf+xml
wins
text/html
wins
RDF
HTML
Content-Location:
http://www.example.org/info.rdf
Content-Location:
http://www.example.org/info.html

Hash URIs without Content Negotiation

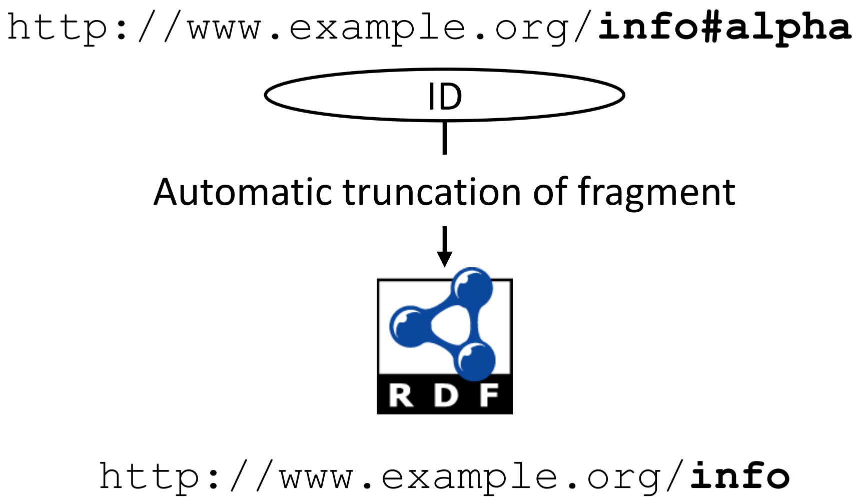
Can be implemented by simply uploading static RDF files to a Web server

- No special server configuration is needed
- Not as technically challenging as the previous one

Popular for quick-and-dirty RDF publication

Major problem: clients will be obliged to load (download) the whole RDF file

- Even if they are interested in only one of the resources


http://www.example.org/**info#alpha**
ID
Automatic truncation of fragment
R D F
http://www.example.org/**info**

303 URIs (1)

Approach relies on the “303 See Other” HTTP status code

- Indication that the requested resource is not a regular Web document

Regular HTTP response (200) cannot be returned

- Requested resource does not have a suitable representation

However, we still can retrieve description about this resource

- Distinguishing between the real-world resource and its description (representation) on the Web

303 URIs (2)

HTTP 303 is a redirect status code

- Server provides the location of a document that represents the resource

E.g. companies Alpha and Beta can be described using the following URIs

- http://www.example.org/id/alpha
- http://www.example.org/id/beta

Server can be configured to answer requests to these URIs with a 303 (redirect) HTTP status code

303 URIs (3)

Location can contain an HTML, an RDF, or any alternative form, e.g.

- http://www.example.org/doc/alpha
- http://www.example.org/doc/beta

This setup allows to maintain bookmarkable, de-referenceable URIs

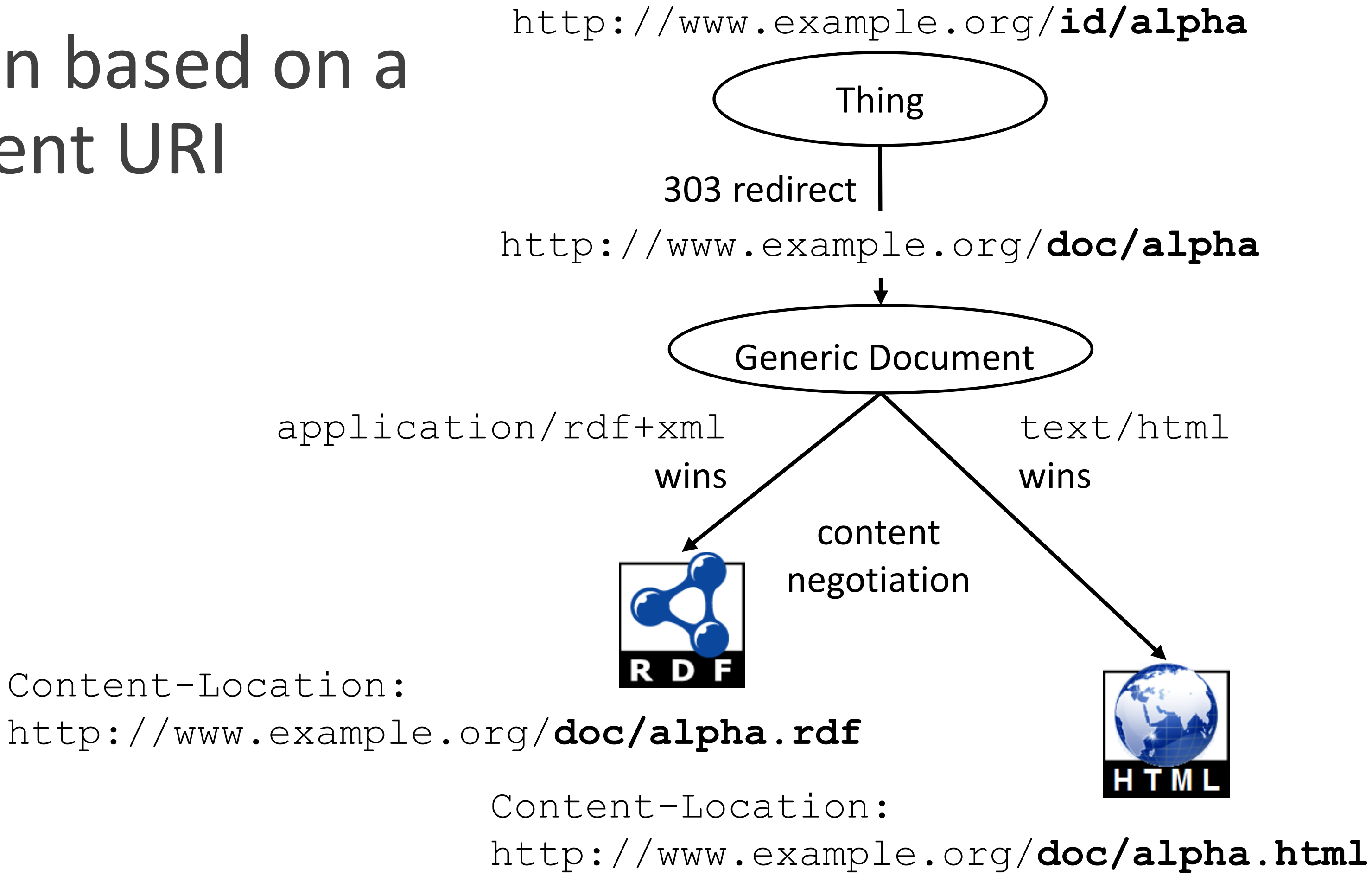
- For both the RDF and HTML views of the same resource

A very flexible approach

- Redirection target can be configured separately per resource
- There could be a document for each resource, or one (large) document with descriptions of all the resources

303 URIs (4)

303 URI solution based on a generic document URI



http://www.example.org/**id/alpha**

Thing

303 redirect

http://www.example.org/**doc/alpha**

Generic Document

application/rdf+xml wins

text/html wins

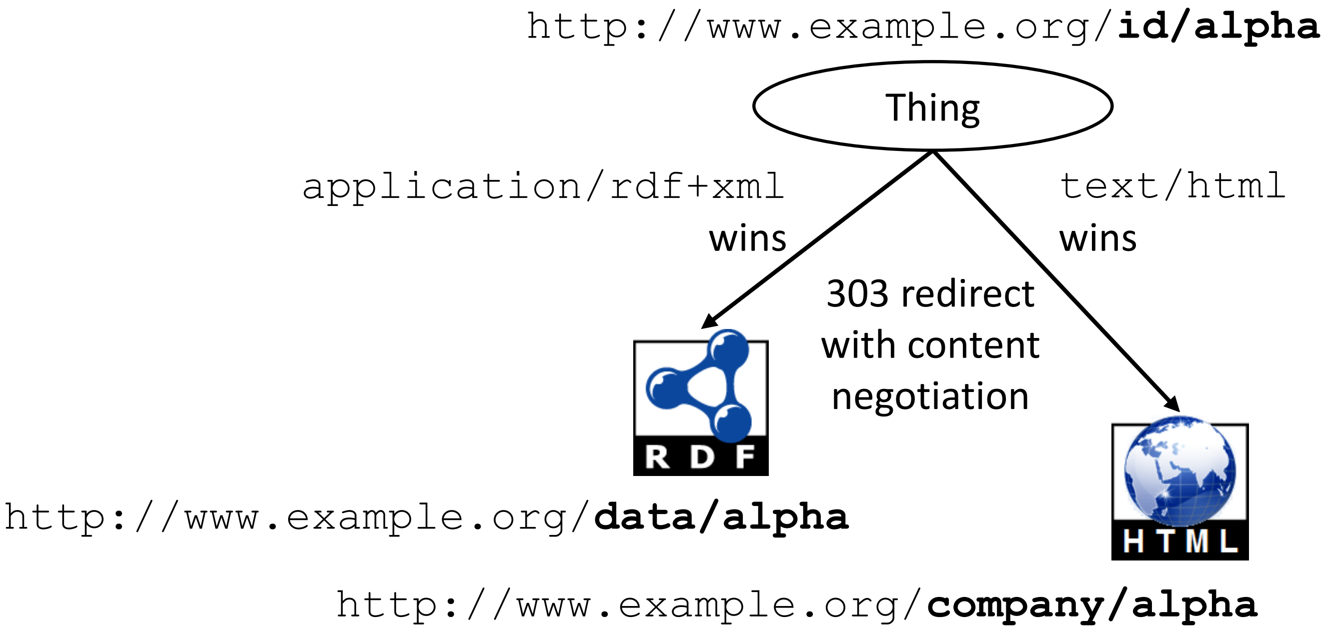
content negotiation

Content-Location: http://www.example.org/**doc/alpha.rdf**

Content-Location: http://www.example.org/**doc/alpha.html**

303 URIs (5)

303 URI solution without the generic document URI



http://www.example.org/**id/alpha**

Thing

application/rdf+xml
wins

text/html
wins

303 redirect
with content
negotiation

http://www.example.org/**data/alpha**

http://www.example.org/**company/alpha**

303 URIs (6)

Problems of the 303 approach

- Latency caused by client redirects
- A client looking up a set of terms may use many HTTP requests
  - While everything that could be loaded in the first request is there and is ready to be downloaded
- Clients of large datasets may be tempted to download the full data via HTTP, using many requests
  - In these cases, SPARQL endpoints or comparable services should be provided in order to answer complex queries directly on the server

303 URIs (7)

303 and Hash approaches are not mutually exclusive

- Contrarily, combining them could be ideal
  - Allow large datasets to be separated into multiple parts and have identifiers for non-document resources

Outline

Introduction

Modeling Data

Software for Working with Linked Data

Software Tools for Storing and Processing Linked Data

Tools for Linking and Aligning Linked Data

Software Libraries for working with RDF

Software for Working with Linked Data (1)

Working on small datasets is a task that can be tackled by manually authoring an ontology, however

Publishing LOD means the data has to be programmatically manipulated

Many tools exist that facilitate the effort

Software for Working with Linked Data (2)

The most prominent tools listed next

- Ontology authoring environments
- Cleaning data
- Software tools and libraries for working with Linked Data
- No clear lines can be drawn among software categories
  - E.g. graphical tools offering programmatic access, or software libraries offering a GUI

Ontology Authoring (1)

Not a linear process

- It is not possible to line up the steps needed in order to complete its authoring

An iterative procedure

- The core ontology structure can be enriched with more specialized, peripheral concepts
  - Further complicating concept relations
  - The more the ontology authoring effort advances, the more complicated the ontology becomes

Ontology Authoring (2)

Various approaches

- Start from the more general and continue with the more specific concepts
- Reversely, write down the more specific concepts and group them

Can uncover existing problems

- E.g. concept clarification
- Understanding of the domain in order to create its model
- Probable reuse and connect to other ontologies

Ontology Editors (1)

Offer a graphical interface

- Through which the user can interact
- Textual representation of ontologies can be prohibitively obscure

Assure syntactic validity of the ontology

Consistency checks

Ontology Editors (2)

Freedom to define concepts and their relations

- Constrained to assure semantic consistency

Allow revisions

Several ontology editors have been built

- Only a few are practically used
- Among them the ones presented next

Protégé (1)

Open-source

Maintained by the Stanford Center for Biomedical Informatics Research

Among the most long-lived, complete and capable solutions for ontology authoring and managing

A rich set of plugins and capabilities

Protégé (2)

Customizable user interface

Multiple ontologies can be developed in a single frame workspace

Several Protégé frames roughly correspond to OWL components

- Classes
- Properties
  - Object Properties, Data Properties, and Annotation Properties
- Individuals

A set of tools for visualization, querying, and refactoring

Protégé (3)

Reasoning support

- Connection to DL reasoners like HermiT (included) or Pellet

OWL 2 support

Allows SPARQL queries

WebProtégé

- A much younger offspring of the Desktop version
- Allows collaborative viewing and editing
- Less feature-rich, more buggy user interface

TopBraid Composer

RDF and OWL authoring and editing environment

Based on the Eclipse development platform

A series of adapters for the conversion of data to RDF

- E.g. from XML, spreadsheets and relational databases

Supports persistence of RDF graphs in external triple stores

Ability to define SPIN rules and constraints and associate them with OWL classes

Maestro, Commercial, Free edition

- Free edition offers merely a graphical interface for the definition of RDF graphs and OWL ontologies and the execution of SPARQL queries on them

The NeOn Toolkit

Open-source ontology authoring environment

- Also based on the Eclipse platform

Mainly implemented in the course of the EC-funded NeOn project

- Main goal: the support for all tasks in the ontology engineering life-cycle

Contains a number of plugins

- Multi-user collaborative ontology development
- Ontology evolution through time
- Ontology annotation
- Querying and reasoning
- Mappings between relational databases and ontologies
  - ODEMapster plugin

Platforms and Environments

Data that published as Linked Data is not always produced primarily in this form

- Files in hard drives, relational databases, legacy systems etc.

Many options regarding how the information is to be transformed into RDF

Many software tools and libraries available in the Linked Data ecosystem

- E.g. for converting, cleaning up, storing, visualizing, linking etc.
- Creating Linked Data from relational databases is a special case, discussed in detail in the next Chapter

Cleaning-Up Data: OpenRefine (1)

Data quality may be lower than expected

- In terms of homogeneity, completeness, validity, consistency, etc.

Prior processing has to take place before publishing

It is not enough to provide data as Linked Data

- Published data must meet certain quality standards

Cleaning-Up Data: OpenRefine (2)

Initially developed as “Freebase Gridworks”, renamed “Google Refine” in 2010, “OpenRefine” after its transition to a community-supported project in 2012

Created specifically to help working with messy data

Used to improve data consistency and quality

Used in cases where the primary data source are files

- In tabular form (e.g. TSV, CSV, Excel spreadsheets) or
- Structured as XML, JSON, or even RDF.

Cleaning-Up Data: OpenRefine (3)

Allows importing data into the tool and connect them to other sources

It is a web application, intended to run locally, in order to allow processing sensitive data

Cleaning data

- Removing duplicate records
- Separating multiple values that may reside in the same field
- Splitting multi-valued fields
- Identifying errors (isolated or systematic)
- Applying ad-hoc transformations using regular expressions

OpenRefine: The RDF Refine Extension (1)

Allows conversion from other sources to RDF

- RDF export
- RDF reconciliation

RDF export part

- Describe the shape of the generated RDF graph through a template
- Template uses values from the input spreadsheet
- User can specify the structure of an RDF graph
  - The relationships that hold among resources
  - The form of the URI scheme that will be followed, etc.

OpenRefine: The RDF Refine Extension (2)

RDF reconciliation part

- Offers a series of alternatives for discovering related Linked Data entities
- A reconciliation service
- Allows reconciliation of resources
    - Against an arbitrary SPARQL endpoint
        - With or without full-text search functionality
        - A predefined SPARQL query that contains the request label (i.e. the label of the resource to be reconciled) is sent to a specific SPARQL endpoint
    - Via the Sindice API
        - A call to the Sindice API is directly made using the request label as input to the service

Outline

Introduction

Modeling Data

Software Tools for Storing and Processing Linked Data

Tools for Linking and Aligning Linked Data

Software Libraries for working with RDF

Tools for Storing and Processing Linked Data

Storing and processing solutions

- Usage not restricted to these capabilities

A mature ecosystem of technologies and solutions

- Cover practical problems such as programmatic access, storage, visualization, querying via SPARQL endpoints, etc.

Sesame

An open source, fully extensible and configurable with respect to storage mechanisms, Java framework for processing RDF data

Transaction support

RDF 1.1 support

Storing and querying APIs

A RESTful HTTP interface supporting SPARQL

The Storage And Inference Layer (Sail) API

- A low level system API for RDF stores and inferences, allowing for various types of storage and inference to be used

OpenLink Virtuoso (1)

RDF data management and Linked Data server solution

- Also a web application/web services/relational database/file server

Offers a free and a commercial edition

Implements a quad store

- (graph, subject, predicate, object)

OpenLink Virtuoso (2)

Graphs can be

- Directly uploaded to Virtuoso
- Transient (not materialized) RDF views on top of its relational database backend
- Crawled from third party RDF (or non-RDF, using Sponger) sources

Offers several plugins

- Full-text search, faceted browsing, etc.

Apache Marmotta

The LDClient library

- A Linked Data Client
- A modular tool that can convert data from other formats into RDF
- Can be used by any Linked Data project, independent of the Apache Marmotta platform

Can retrieve resources from remote data sources and map their data to appropriate RDF structures

A number of different backends is included

- Provide access to online resources
  - E.g. Freebase, Facebook graph API, RDFa-augmented HTML pages

Callimachus

An open source platform

- Also available in an enterprise closed-source edition

Development of web applications based on RDF and Linked Data

A Linked Data Management System

- Creating, managing, navigating and visualizing Linked Data through appropriate front-end components

Relies on XHTML and RDFa templates

- Populated by the results of SPARQL queries executed against an RDF triple store
  - Constitute the human-readable web pages

Visualization software

Users may have limited or non-existent knowledge of Linked Data and the related ecosystem

LodLive

- Provides a navigator that uses RDF resources, relying on SPARQL endpoints

CubeViz

- A faceted browser for statistical data
- Relies on the RDF Data Cube vocabulary for representing statistical data in RDF

Gephi

GraphViz

Open-source, generic graph visualization platforms

Apache Stanbol

A semantic content management system

Aims at extending traditional CMS's with semantic services

Reusable components

- Via a RESTful web service API that returns JSON, RDF and supports JSON-LD

Ontology manipulation

Content enhancement

- Semantic annotation

Reasoning

Persistence

Stardog

RDF database, geared towards scalability

Reasoning

OWL 2

SWRL

Implemented in Java

Exposes APIs for Jena and Sesame

Offers bindings for its HTTP protocol in numerous languages

- Javascript, .Net, Ruby, Clojure, Python

Commercial and free community edition

Outline

Introduction

Modeling Data

Software Tools for Storing and Processing Linked Data

Tools for Linking and Aligning Linked Data

Software Libraries for working with RDF

The L in LOD (1)

Web of Documents

- HTML Links
  - Navigate among (HTML) pages

Web of (Linked) Data

- RDF links
  - Navigate among (RDF) data
  - Relationships between Web resources
  - Triples (resource, property, resource)
  - Main difference from simple hyperlinks: they possess some meaning

The L in LOD (2)

Links to external datasets of the LOD cloud

Integration of the new dataset in the Web of data

Without links, all published RDF datasets would essentially be isolated islands in the “ocean” of Linked Data

The L in LOD (3)

Establishing links can be done

- Manually
  - I.e. the knowledge engineer identifies the most appropriate datasets and external resources
  - More suitable for small and static datasets
- Semi-automatically
  - Harness existing open-source tools developed for such a purpose
  - More suitable for large datasets

Silk (1)

An open-source framework for the discovery of links among RDF resources of different datasets

Available

- As a command line tool
- With a graphical user interface
- Cluster edition, for the discovery of links among large datasets

Silk (2)

Link specification language

- Details and criteria of the matching process, including
  - The source and target RDF datasets
  - The conditions that resources should fulfill in order to be interlinked
  - The RDF predicate that will be used for the link
  - The pre-matching transformation functions
  - Similarity metrics to be applied

LIMES

A link discovery framework among RDF datasets

Extracts instances and properties from both source and target datasets, stores them in a cache storage or memory and computes the actual matches

- Based on restrictions specified in a configuration file

Offers a web interface for authoring the configuration file

Sindice

A service that can be used for the manual discovery of related identifiers

An index of RDF datasets that have been crawled and/or extracted from semantically marked up Web pages

Offers both free-text search and SPARQL query execution functionalities

Exposes several APIs that enable the development of Linked Data applications that can exploit Sindice’s crawled content

DBpedia Spotlight

Designed for annotating mentions of DBpedia resources in text

Provides an approach for linking information from unstructured sources to the LOD cloud through DBpedia

Tool architecture comprises:

- A web application
- A web service
- An annotation and an indexing API in Java/Scala
- An evaluation module

Sameas.org

An online service

Retrieves related LOD entities from some of the most popular datasets

Serves more than 150 million URIs

Provides a REST interface that retrieves related URIs for a given input URI or label

Accepts URIs as inputs from the user and returns URIs that may well be co-referent

Outline

Introduction

Modeling Data

Tools for Linking and Aligning Linked Data

Software Libraries for working with RDF

Jena (1)

Open-source Java API

Allows building of Semantic Web and Linked Data applications

The most popular Java framework for ontology manipulation

First developed and brought to maturity by HP Labs

Now developed and maintained by the Apache Software Foundation

First version in 2000

Comprises a set of tools for programmatic access and management of Semantic Web resources

Jena (2)

ARQ, a SPARQL implementation

Fuseki

- SPARQL Server, part of the Jena framework, that offers access to the data over HTTP using RESTful services. Fuseki can be downloaded and extracted locally, and run as a server offering a SPARQL endpoint plus some REST commands to update the dataset.

OWL support

- Coverage of the OWL language

Inference API

- Using an internal rule engine, Jena can be used as a reasoner

Jena TDB (1)

High-performance triple store solution

Based on a custom implementation of threaded B+ Trees

- Only provides for fixed length key and fixed length value
- No use of the value part in triple indexes

Efficient storage and querying of large volumes of graphs

- Performs faster, scales much more than a relational database backend

Stores the dataset in directory

Jena TDB (2)

A TDB instance consists of

- A Node table that stores the representation of RDF terms
- Triple and Quad indexes
  - Triples are used for the default graph, quads for the named graphs
- The Prefixes table
  - Provides support for Jena’s prefix mappings and serves mainly presentation and serialization of triples issues, and does not take part in query processing

Apache Any23 (1)

A programming library

A web service

A command line tool

Ability to extract structured data in RDF from Web documents

Input formats

- RDF (RDF/XML, Turtle, Notation 3)
- RDFa, microformats (hCalendar, hCard, hListing, etc.)
- HTML 5 Microdata (such as schema.org)
- JSON-LD (Linked Data in JSON format)

Apache Any23 (2)

Support for content extraction following several vocabularies

- CSV
- Dublin Core Terms
- Description of a Career
- Description Of A Project
- Friend Of A Friend
- GeoNames
- ICAL
- Open Graph Protocol
- schema.org
- VCard, etc.

Redland

Support for programmatic management and storage of RDF graphs

A set of libraries written in C, including:

- Raptor
  - Provides a set of parsers and serializers of RDF
  - Including RDF/XML, Turtle, RDFa, N-Quads, etc.
- Rasqal
  - Supports SPARQL query processing of RDF graphs
- Redland RDF Library
  - Handles RDF manipulation and storage

Also allows function invocation through Perl, PHP, Ruby, Python, etc.

EasyRDF

A PHP library for the consumption and production of RDF

Offers parsers and serializers for most RDF serializations

Querying using SPARQL

Type mapping from RDF resources to PHP objects

RDFLib

A Python library for working with RDF

Serialization formats

Microformats

RDFa

OWL 2 RL

Using relational databases as a backend

Wrappers for remote SPARQL endpoints

The Ruby RDF Project

RDF Processing using the Ruby language

Reading/writing in different RDF formats

Microdata support

Querying using SPARQL

Using relational databases as a storage backend

Storage adaptors

- Sesame
- MongoDB

dotNetRDF (1)

Open-source library for RDF

Written in C#.Net, also offering ASP.NET integration

Developer API

dotNetRDF (2)

Supports

- SPARQL
- Reasoning
- Integration with third party triple stores
  - Jena, Sesame, Stardog, Virtuoso, etc.

Suite of command line and graphical tools for

- Conversions between RDF formats
- Running a SPARQL server and submitting queries
- Managing any supported triple stores

Chapter 4
Creating Linked Data from Relational Databases

NIKOLAOS KONSTANTINOU

DIMITRIOS-EMMANUEL SPANOS

Outline

Introduction

Motivation-Benefits

Classification of approaches

Creating ontology and triples from a relational database

Complete example

Future outlook

Introduction (1)

Relational databases vs. Semantic Web standards

- Active research topic since more than a decade ago
- Not just a theoretical exercise, but also practical value
  - Bootstrap the Semantic Web with a sufficiently large mass of data
  - Facilitate database integration
  - Ontology-based data access
  - Semantic annotation of dynamic Web pages

Introduction (2)

Database-to-ontology mapping

- The investigation of the similarities and differences among relational databases and Semantic Web knowledge models
- Broad term encompassing several distinct problems
- Classification of approaches needed

Outline

Introduction

Motivation-Benefits

Classification of approaches

Creating ontology and triples from a relational database

Complete example

Future outlook

Semantic Annotation of Dynamic Web Pages (1)

Goal of the Semantic Web: emergence of a Web of Data, from the current Web of Documents

HTML documents are mainly for human consumption

How to achieve this?

- Add *semantic* information to HTML documents
  - I.e. setup correspondences with terms from ontologies

RDFa

- Embedding references to ontology terms in XHTML tags, *but*…

Semantic Annotation of Dynamic Web Pages (2)

What about dynamic documents?

- Content retrieved from relational databases
- The biggest part of the World Wide Web
  - Aka. Deep Web
- CMS's, forums, wikis, etc.
- Manual annotation of every single dynamic web page is infeasible

Semantic Annotation of Dynamic Web Pages (3)

Directly “annotate” the database schema!

- Establish correspondences between the elements of the database schema and a suitable existing domain ontology

Use these correspondences to generate automatically semantically annotated dynamic pages

Heterogeneous Database Integration (1)

Longstanding issue in database research

- Due to differences in:
  - Software infrastructure
  - Syntax
  - Representation models
  - Interpretation of the same data

Remains unresolved to a large degree

Heterogeneous Database Integration (2)

Typical database integration architecture

- One or more conceptual models for the description of the contents of each source database
- Queries against a global conceptual schema
- Wrappers on top of every source database for the reformulation of queries and data retrieval

Heterogeneous Database Integration (3)

Ontology-based database integration

- Ontologies instead of conceptual schemas
- Definition of correspondences between source databases and one or more ontologies
- LAV, GAV or GLAV approach (target schema = ontology)
  - Database term ↔ Query over the ontology (LAV)
  - Ontology term ↔ Query over the database (GAV)
  - Query over the database ↔ Query over the ontology (GLAV)
- Mappings between relational database schemas and ontologies need to be discovered!

Ontology-Based Data Access (1)

Objective:

- Offer high-level services on top of an information system without knowledge of the underlying database schema

Ontology as an intermediate layer between the end user and the storage layer

- Ontology provides an abstraction of the database contents
- Users formulate queries using terms from the ontology

Ontology-Based Data Access (2)

Similar to a database integration architecture

- OBDA engine ≈ wrapper
  - Transforms queries against the ontology to queries against the local data source

OBDA engine

- Performs query rewriting
- Uses mappings between a database and a relevant domain ontology

Advantages

- Semantic queries posed directly to the database
- No need to replicate database contents in RDF

Semantic Rewriting of SQL Queries

Objective:

- Reformulate an SQL query to another one that better captures the intention of the user

Substitution of terms in the original SQL query with synonyms and related terms from an ontology

Also related:

- Query relational data using external ontologies as context
  - SQL queries with their WHERE conditions containing terms from an ontology

Feature implemented in some DBMSes

- E.g. OpenLink Virtuoso, Oracle

Mass Data Generation for the Semantic Web

Reasons for slow uptake of the Semantic Web

- Few successful paradigms of tools and “killer” applications
- Few data
- “Chicken-and-egg” problem

Relational databases hold the majority of data on the World Wide Web

Automated extraction of RDB contents in RDF

Generation of a critical mass of Semantic Web data

Increased production of SW applications and tools anticipated

Ontology Learning (1)

Manual development of ontologies is difficult, time-consuming and error-prone

Ontology learning

- Semi-automatic extraction of ontologies from free texts, semi-structured documents, controlled vocabularies, thesauri etc.
- Relational databases can be sources of domain knowledge as well
- Information gathered from database schema, contents, queries and stored procedures
- Supervision from domain expert is necessary

Ontology Learning (2)

Useful in domains where there is no suitable ontology

- Typical in the earlier Semantic Web years

Nowadays, ontology learning for the creation of a "wrapping" ontology for an RDB in:

- OBDA
- Database integration

Intended Meaning of a Relational Schema (1)

Database schema design

- Conceptual model ⟶ relational model
- Subsequent changes often directly to the relational model
- Initial conceptual model lost
- Hard to re-engineer to another model (e.g. object-oriented)

Definition of correspondences between RDB and ontology

- Semantic grounding of the meaning of the former

Intended Meaning of a Relational Schema (2)

Facilitates:

- Database maintenance
- Integration with other data sources
- Mapping discovery between 2 or more database schemas

In the latter case, database-to-ontology mappings are used as a reference point for the construction of inter-database schema mappings

Database Integration with Other Data Sources

Mapping RDB to RDF enables integration with existing RDF content

- Content generated from either structured or unstructured sources

Linked Data paradigm

- Vocabulary reuse
- Inter-dataset links
- Identifier reuse
- Facilitates data source integration at global level
- Billions of RDF statements from several domains of interest

Integration of RDB content with Linked Data offers unlimited potential

Outline

Introduction

Motivation-Benefits

Classification of approaches

Creating ontology and triples from a relational database

Complete example

Future outlook

Existing Classifications (1)

Several classification schemes proposed for database-to-ontology mapping approaches

Classification criteria vs. descriptive measures

- Classification criteria
  - Finite number of values
  - Should separate approaches in non-overlapping sets
- Descriptive measures
  - Can also be qualitative

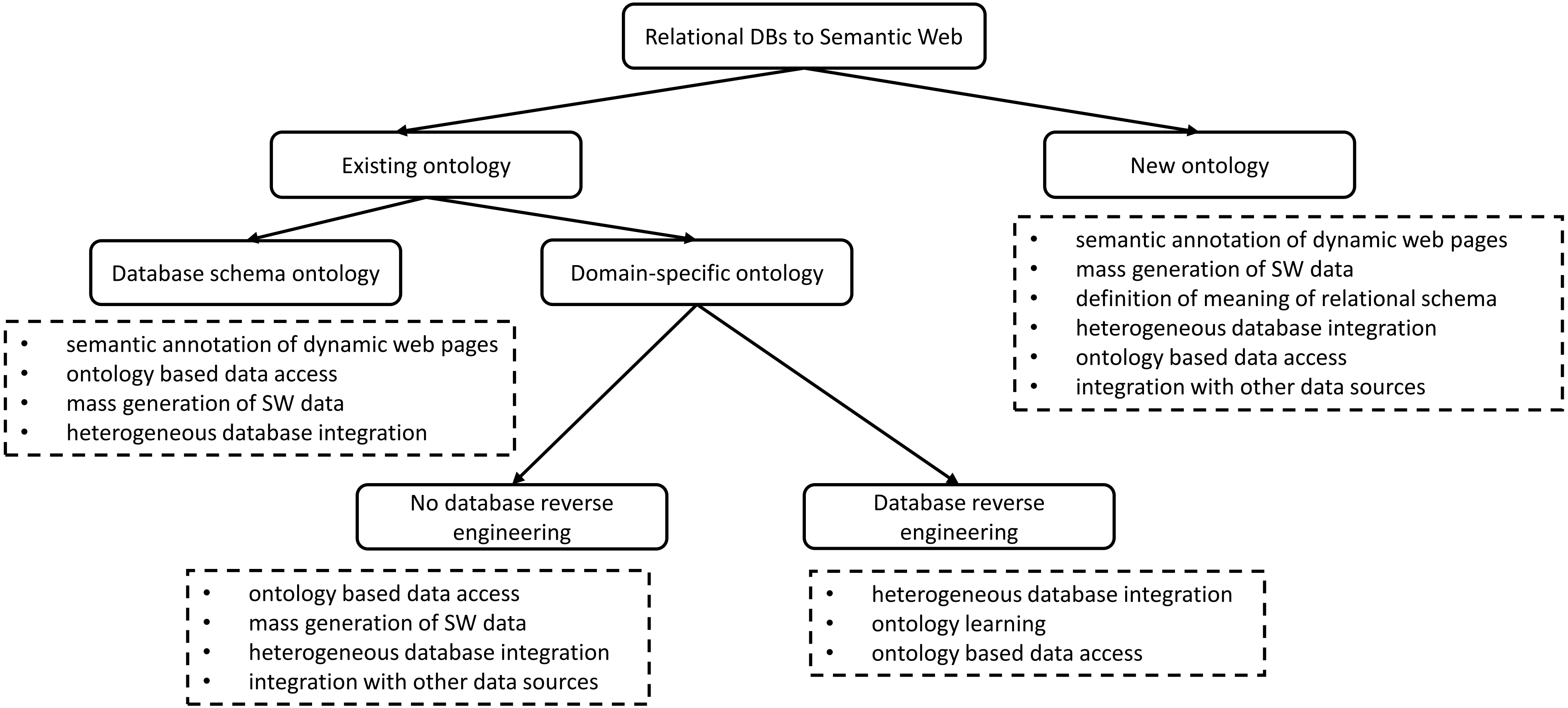
Existing Classifications (2)

Work	Classification criteria	Values	Descriptive parameters
(Auer et al. 2009)	a. Automation in the creation of mapping b. Source of semantics considered c. Access paradigm d. Domain reliance	a. Automatic/Semi-automatic/Manual b. Existing domain ontologies/Database/Database and User c. Extract-Transform-Load (ETL)/SPARQL/Linked Data d. General/Dependent	Mapping representation language
(Barrasa-Rodriguez and Gómez-Pérez 2006)	a. Existence of ontology b. Architecture c. Mapping exploitation	Yes (ontology reuse)/No (created ad-hoc) Wrapper/Generic engine and declarative definition Massive upgrade (batch)/Query driven (on demand)	-
(Ghawi and Cullot 2007)	a. Existence of ontology b. Complexity of mapping definition c. Ontology population process d. Automation in the creation of mapping	a. Yes/No b. Complex/Direct c. Massive dump/Query driven d. Automatic/Semi-automatic/Manual	Automation in the instance export process
(Hellmann et al. 2011)	-	-	Data source, Data exposition, Data synchronization, Mapping language, Vocabulary reuse, Mapping automation, Requirement of domain ontology, Existence of GUI

Existing Classifications (3)

Work	Classification criteria	Values	Descriptive parameters
(Konstantinou et al. 2008)	a. Existence of ontology b. Automation in the creation of mapping c. Ontology development	a. Yes/No b. Automatic/Semi-automatic/Manual c. Structure driven/Semantics driven	Ontology language, RDBMS supported, Semantic query language, Database components mapped, Availability of consistency checks, User interaction
(Sahoo et al. 2009)	Same as in (Auer et al. 2009) with the addition of:		
	a. Query implementation b. Data integration	a. SPARQL/SPARQL→SQL b. Yes/No	Mapping accessibility, Application domain
(Sequeda et al. 2009)	-	-	Correlation of primary and foreign keys, OWL and RDFS elements mapped
(Zhao and Chang 2007)	a. Database schema analysis	Yes/No	Purpose, Input, Output, Correlation analysis of database schema elements, Consideration of database instance, application source code and other sources

A Proposed Classification (1)



Relational DBs to Semantic Web

Existing ontology

New ontology

- semantic annotation of dynamic web pages
- mass generation of SW data
- definition of meaning of relational schema
- heterogeneous database integration
- ontology based data access
- integration with other data sources

Database schema ontology

- semantic annotation of dynamic web pages
- ontology based data access
- mass generation of SW data
- heterogeneous database integration

Domain-specific ontology

No database reverse engineering

- ontology based data access
- mass generation of SW data
- heterogeneous database integration
- integration with other data sources

Database reverse engineering

- heterogeneous database integration
- ontology learning
- ontology based data access

A Proposed Classification (2)

Total classification of all relevant solutions in mutually disjoint classes

Exceptions

- Customizable software tools with multiple possible workflows
- Each one belongs to multiple categories

Every class associated with a number of benefits/motivations

- Not significant correlation among taxonomy classes and motivations and benefits
- Categorization of approaches based on the *nature* of the mapping and the *techniques applied* to establish the mapping
- Benefits state the *applications* of the already established mappings

Classification Criteria (1)

Existence of ontology

- Is an ontology *required* for the application of the approach?
- Yes
  - Establishment of mappings between a given relational database and a given existing ontology
  - Domain of ontology compatible with database domain
  - Existing ontology selected by human user
- No
  - Creation of a new ontology from a given relational database
  - Useful when:
    - An ontology for the domain covered by the database is not available yet
    - The human user is not familiar with the domain of the database and relies on the mapping process to discover the semantics of the database contents

Classification Criteria (2)

Domain of the generated ontology

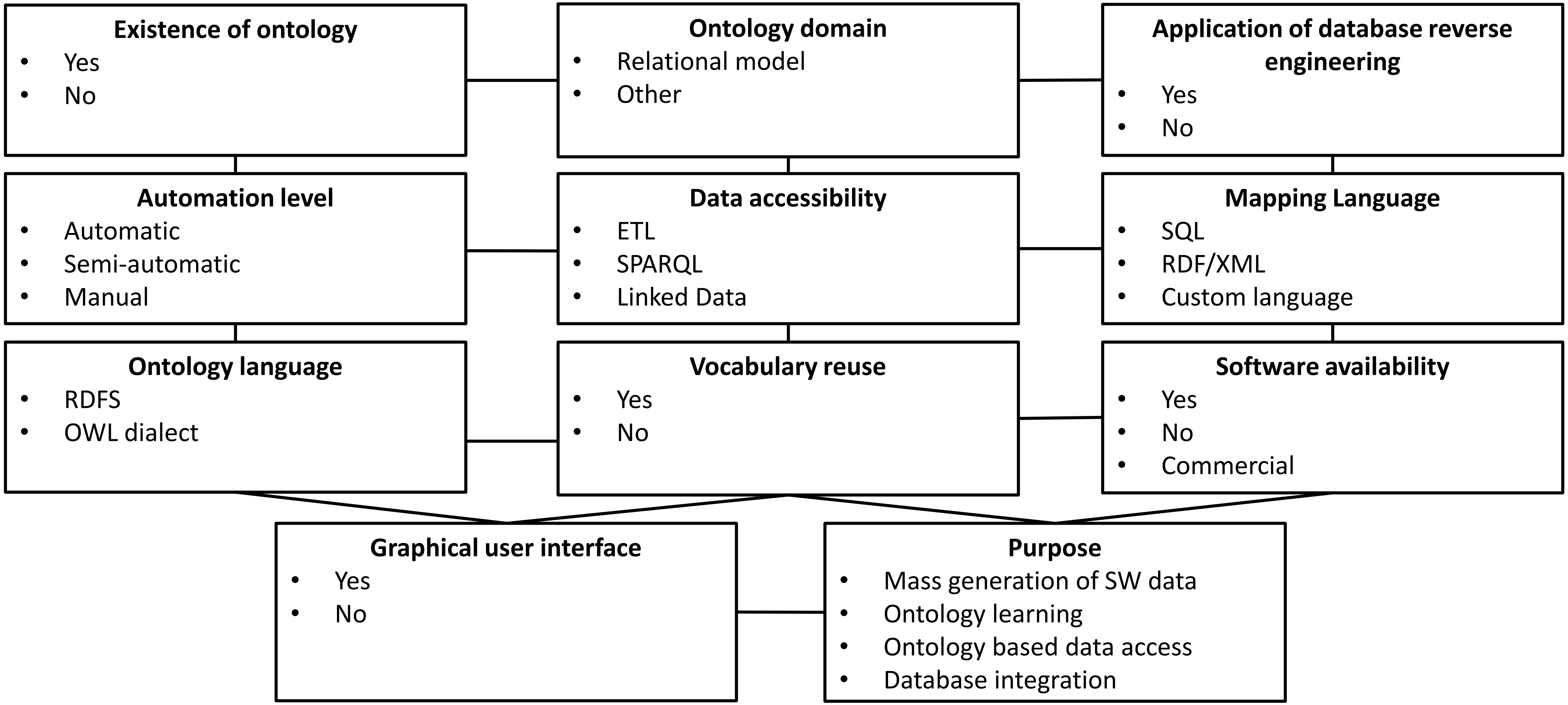
- What is the domain of the generated ontology?
- The relational model
    - Generated ontology consists of concepts and relationships that reflect the constructs of the relational model
    - Mirrors the structure of the input relational database
    - “*Database schema ontology*”
    - Mainly automatic class of approaches
- Another domain
    - Depending on the domain described by the contents of the input database

Classification Criteria (3)

Database reverse engineering

- Are any database reverse engineering techniques applied?
- Yes
  - Recover the initial conceptual schema from the relational schema
  - Translate re-engineered schema to an ontology expressed in a target language
- No
  - Few basic translation rules from the relational to the RDF model
  - Reliance on the human expert for the definition of complex mappings and the enrichment of the generated ontology

Classification criteria and descriptive features



Existence of ontology
- Yes
- No

Ontology domain
- Relational model
- Other

Application of database reverse engineering
- Yes
- No

Automation level
- Automatic
- Semi-automatic
- Manual

Data accessibility
- ETL
- SPARQL
- Linked Data

Mapping Language
- SQL
- RDF/XML
- Custom language

Ontology language
- RDFS
- OWL dialect

Vocabulary reuse
- Yes
- No

Software availability
- Yes
- No
- Commercial

Graphical user interface
- Yes
- No

Purpose
- Mass generation of SW data
- Ontology learning
- Ontology based data access
- Database integration

Descriptive Features (1)

Level of Automation

- How much is the user involved in the mapping process?
- Automatic
  - No input from human user
- Semi-automatic
  - Some input from human user
  - Sometimes necessary
  - Sometimes optional (e.g. validation or enrichment of results)
- Manual
  - Mapping defined entirely from human user
- Feature usually common among approaches of the same class

Descriptive Features (2)

Data Accessibility

- The way the mapping result is accessed
  - Aka. access paradigm / mapping implementation / data exposition
- ETL
  - Result of the mapping process generated and stored as a whole in an external storage medium (i.e. *materialized*)
  - Aka. batch transformation / massive dump

Descriptive Features (3)

Data Accessibility (cont’d)

- SPARQL
  - Only a part of the mapping result is accessed
  - No additional storage medium is required (i.e. no materialization)
  - Rewriting of a SPARQL query to an SQL one
  - SQL results transformed back to SPARQL results
  - Aka. query-driven access
- Linked Data
  - Mapping result published as Linked Data (i.e. all URIs use the HTTP scheme and, when dereferenced, provide useful information for the resource they identify)

Descriptive Features (4)

Data Synchronization

- Does the mapping result reflect the current database contents?
- Static
  - Mapping executed only once
  - Mapping result not tied with source database
- Dynamic
  - Mapping executed on every incoming query
  - Mapping result depends on current database state
- Strongly related to data accessibility, redundant feature
- ETL methods are static
- SPARQL (query-driven) and Linked Data methods are dynamic

Descriptive Features (5)

Mapping language

- The language in which the mapping is represented
- Large variance of values: a lot of proprietary formats
- …until the standardization of R2RML
- Feature only applicable to methods that need to reuse the mapping
    - E.g. not applicable to ontology generation methods

Descriptive Features (6)

Ontology language

- The language in which the involved ontology is expressed
- Either:
  - The language of the ontology generated by the approach
  - The language of the existing ontology required
- RDFS
- OWL (all flavours and dialects)

Descriptive Features (7)

Vocabulary reuse

- Does the mapping support more than one existing ontologies?
- Yes
  - Mainly manual approaches
  - Human user free to reuse terms from existing ontologies
  - Not obligatory to reuse terms
- No
  - E.g. methods generating a new “database schema ontology”

Descriptive Features (8)

Software availability

- Does the method have a free implementation?
- Theoretical methods
- Practical solutions
- Commercial software

Descriptive Features (9)

Graphical User Interface

- Can the user interact with the system via a GUI?
- Feature applicable to approaches with an accessible software implementation
- Guides user through steps of the mapping process
- Provides mapping suggestions
- Essential for inexperienced users / users not familiar with SW technologies

Outline

Introduction

Motivation-Benefits

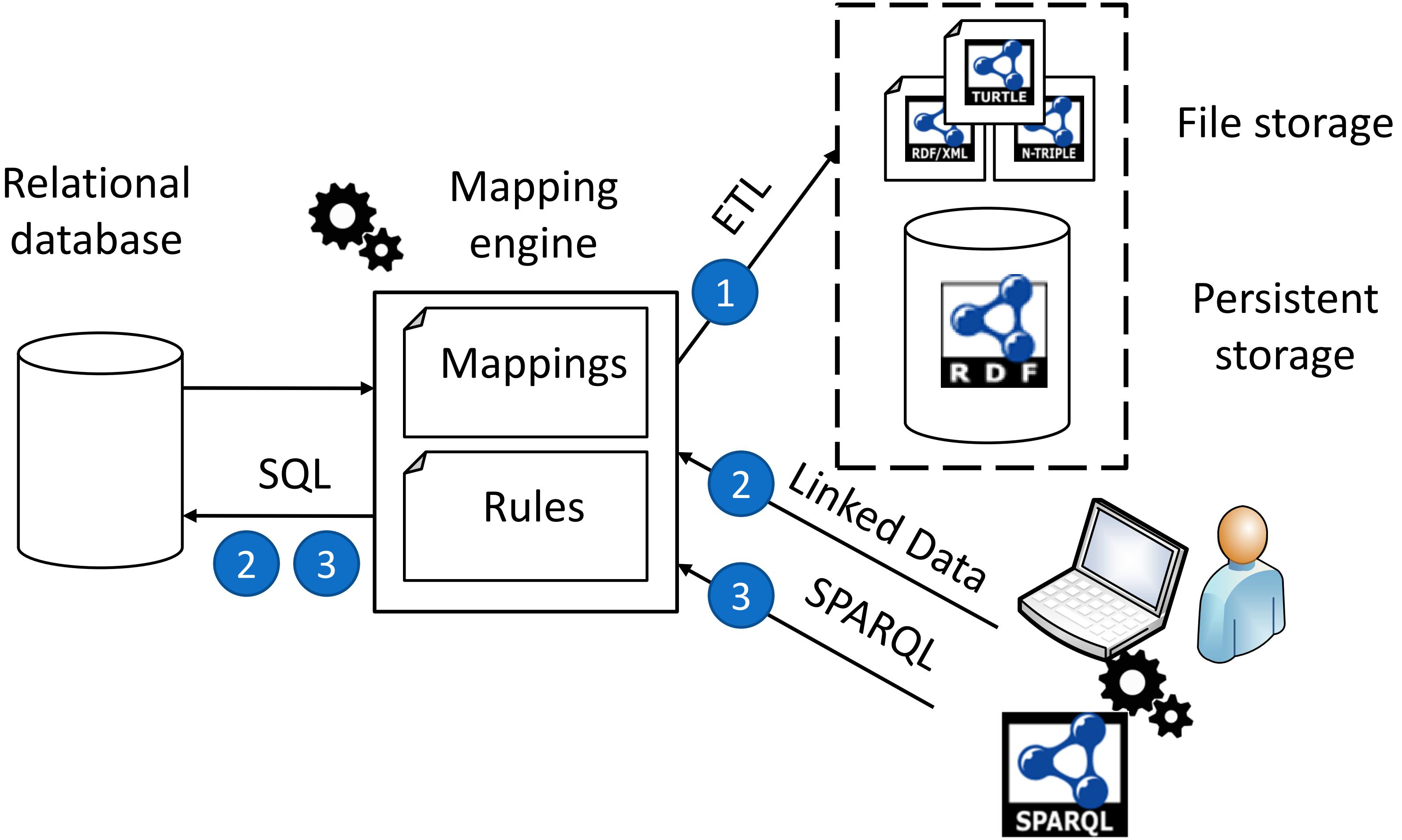
Classification of approaches

Creating ontology and triples from a relational database

Complete example

Future outlook

Creating Ontology and Triples from a Relational Database (1)



Creating Ontology and Triples from a Relational Database (2)

Generation of a new ontology

Population with RDF data originating from the database

Mapping engine

- Communicates with database
- Uses heuristic or manually defined rules

3 ways to access the generated RDF data

- ETL
- SPARQL
- Linked Data

The Basic Approach (1)

Method proposed by Tim Berners-Lee (1998)

Generic, applicable to every database

Automatic

“Table-to-class, column-to-predicate” method

A URI generation scheme also needed

- Should be reversible (i.e. recognize database element from URI)

The Basic Approach (2)

Rules:

(a) Every relation *R* maps to an RDFS class *C*(*R*)

(b) Every tuple of a relation *R* maps to an RDF node of type *C*(*R*)

(c) Every attribute *att* of a relation maps to an RDF property *P*(*att*)

(d) For every tuple *R*[*t*], the value of an attribute *att* maps to a value of the property *P*(*att*) for the node corresponding to the tuple *R*[*t*]

The Basic Approach (3)

Typical URI generation scheme

Database Element	URI Template	Example
Database	*{base_URI}/{db}*	http://www.example.org/company_db
Relation	*{base_URI}/{db}/{rel}*	http://www.example.org/company_db/emp
Attribute	*{base_URI}/{db}/{rel}#{attr}*	http://www.example.org/company_db/emp#name
Tuple	*{base_URI}/{db}/{rel}/{pk=pkval}*	http://www.example.org/company_db/emp/id=5

db: database name

rel: relation name

attr: attribute name

pk: name of a primary key

pkval: value of primary key for given tuple

The Basic Approach (4)

Very crude export

Simple generated ontology

- No complex constructs
- Looks like a copy of the relational schema

New URI for every tuple

- Even when there is an existing one for an entity

All database values mapped to literals

- “Flat” RDF graph

Nevertheless, serves as foundation for several approaches

Creation and Population of a Domain Ontology (1)

"Database schema ontologies" are hardly useful for Linked Data publication

Domain-specific ontologies reflect the domain of the database

Expressiveness of generated ontology depends on the amount of domain knowledge extracted from:

- Human user
- Relational instance

a) Approaches using database schema reverse engineering

b) Basic approach + enrichment from human user

More tools follow b)

- User has full control of the mapping

Creation and Population of a Domain Ontology (2)

Automation level

- Depends on the involvement of the human user

Data accessibility

- SPARQL-based access more popular

Mapping language

- Needed to express complex correspondences between database and ontology
- Until R2RML, every tool used its own language
- Mapping lock-in, low interoperability

Creation and Population of a Domain Ontology (3)

Ontology language

- RDFS, since majority of tools follows basic approach

Vocabulary reuse

- Possible when mappings are manually defined
- User should be familiar with SW vocabularies

Creation and Population of a Domain Ontology (4)

Main goal

- Generate lightweight ontologies reusing existing terms
- Increased semantic interoperability
- Focus not on ontology expressiveness

Motivation

- Mass generation of RDF data from existing large quantities of relational data
- Easier integration with other heterogeneous data

D2RQ / D2R Server (1)

One of the most popular tools in the field

Both automatic and user-assisted operation modes

- Automatic mode
  - Automatic mapping generation
  - Basic approach + rules for M:N relationships ⟶ RDFS ontology
- Semi-automatic mode
  - User modifies automatic mapping
- Manual mode
  - User builds mapping from scratch

D2RQ / D2R Server (2)

Custom mapping language

- Feature-rich
  - URI generation mechanism
  - Translation schemes for database values etc.

Both ETL and SPARQL-based access

Vocabulary reuse

- Refer to any ontology inside the mapping file

OpenLink Virtuoso Universal Server

Integration platform (both commercial and open-source versions)

RDF Views feature

- Similar functionality to D2RQ

Both automatic and manual modes

- Automatic mode relies on the basic approach

Virtuoso Meta-Schema language for the mapping definition

- Also very expressive
- One has to learn it in order to customize the mapping (same as in D2RQ)

ETL, SPARQL-based and Linked Data access

Triplify

RDF extraction tool from relational instances

Maps subsets of the database contents (i.e. SQL queries) to URIs of ontology terms

- No need for users to learn a new mapping language

Mappings as configuration files

- Can reuse terms from existing vocabularies (manual editing)

ETL (static) and Linked Data (dynamic) access

Predefined mappings for schemas used by popular Web applications

Supports update logs for RDF resources

- Useful for crawling engines

Ultrawrap

Wraps a database as a SPARQL endpoint

Commercial tool

Supports creation of new domain ontology

- Set of advanced heuristic rules

SPARQL-based access

- SPARQL query refers to terms from new ontology
- Mappings expressed as views defined on the relational schema
- Rewriting to SQL queries referring to above views

Support for manual mappings that reuse terms from existing vocabularies

Oracle DBMS

RDF Views feature (similar to Virtuoso)

Query relational data as RDF

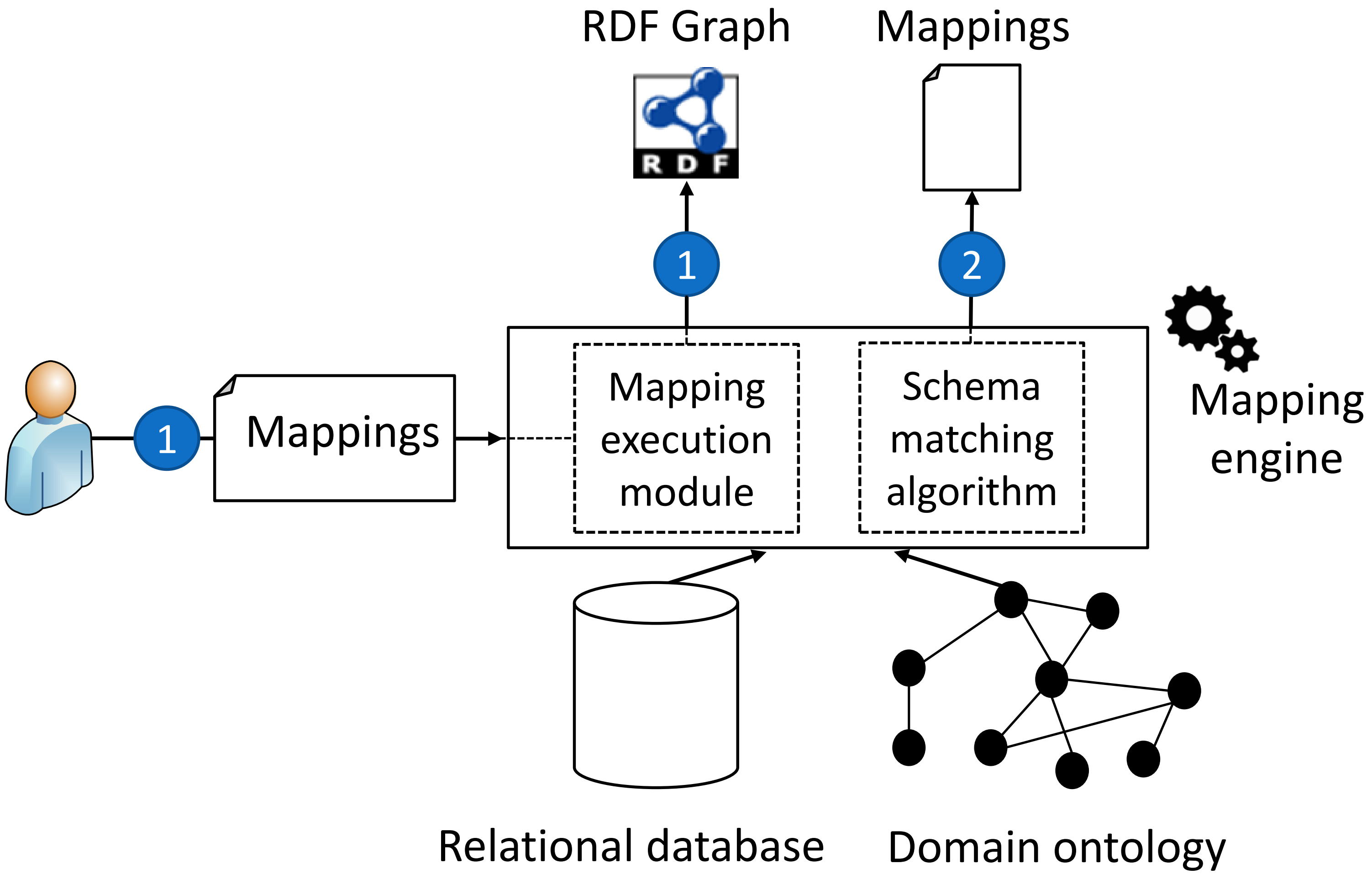
- No replication
- No physical storage for RDF graphs

Both automatic and manual mappings

- Automatic mode follows W3C's Direct Mapping

Supports combination of virtual and materialized RDF data in the same query

Mapping a Database to an Existing Ontology (1)



Mapping a Database to an Existing Ontology (2)

Existence of ontology is required

- Assumption: Ontology domain same as database domain

Discover mappings between a database and an ontology

- Schema matching algorithms
- Reverse engineering + linguistic similarity measures
- Reuse of such mappings in other applications (e.g. database integration)

Mapping a Database to an Existing Ontology (3)

Apply user-defined mappings to a database

- Mappings refer to one or more existing ontologies
- RDF graph contains instance data from the database
- Tools useful for Linked Data publication

Ontop (1)

Conversion of a relational instance to a SPARQL endpoint

User-defined mappings

Ontology-based data access (OBDA) framework

- Not just SPARQL
- RDFS and OWL 2 QL entailment regimes

Ontop (2)

No need to materialize inferences, calculated at query-time

SPARQL-to-SQL rewriting

- Datalog as intermediate representation language
- Several optimizations simplifying generated SQL queries

Plugin for ontology editor Protégé also available

R_2O / ODEMapster / Morph

Declarative XML-based mapping language

Support for complex mappings

- Conditional mappings
- Definition of URI generation scheme

ODEMapster engine

- R_2O mappings
- Materialized / query-driven access

Morph

- R2RML mappings
- SPARQL-based data access

R2RML Parser

Export of RDF graphs from a relational instance

R2RML mappings

Materialized RDF graph (ETL)

Supports faceted browsing of the generated RDF graph

Incremental dump feature

- Tackles the data synchronization issue
- Graph not generated from scratch
- Only the necessary updates are made to the extracted RDF graph

Outline

Introduction

Motivation-Benefits

Classification of approaches

Creating ontology and triples from a relational database

Complete example

Future outlook

Linked Data in Scholarly/Cultural Heritage Domain (1)

Rich experience

Software systems that demonstrate flawless performance

High level of accuracy

Why evolve?

- Data and knowledge description
- New technologies entail new benefits
- Solutions have to remain competitive

Linked Data in Scholarly/Cultural Heritage Domain (2)

Solutions by the LOD paradigm

- Integration
  - Typically materialized using OAI-PMH that does not ease integration with data from other domains
- Expressiveness in describing the information
  - OAI-PMH allows for a tree structure that extends to a depth-level of two
  - RDF allows for a graph-based description
- Query answering
  - Querying graphs using graph patterns allows for much more complex queries

Linked Data in Scholarly/Cultural Heritage Domain (3)

Benefits

- Query expressiveness
- Inherent semantics
- Integration with third party sources

Disadvantages

- Resources investment in creating and maintaining the data

Linked Data in Scholarly/Cultural Heritage Domain (4)

More and more institutions open their data

- Biblioteca Nacional De España
- Deutsche National Bibliothek
- British Library

Linked Data in Scholarly/Cultural Heritage Domain (5)

Is Linked Data the future?

- Content re-use
- Participation of individual collections
- Evolving global Linked Data cloud
- Users can discover new data sources following data-level links
- More complete answers can be delivered as new data sources appear

Ontologies Related to Scholarly Information (1)

Good practice

- Reuse existing vocabularies/ontologies
  - Easier for the outside world to integrate with already existing datasets and services
- Several vocabularies have been proposed

Ontologies Related to Scholarly Information (2)

Title	URL	Namespace	Namespace URL
The Bibliographic Ontology	bibliontology.com	bibo	http://purl.org/ontology/bibo/
Creative Commons Rights Ontology	creativecommons.org	cc	http://creativecommons.org/ns#
CiTo, the Citation Typing Ontology	purl.org/spar/cito	cito	http://purl.org/spar/cito/
Legacy Dublin Core element set	dublincore.org/documents/dces/	dc	http://purl.org/dc/elements/1.1/
DCMI Metadata Terms	dublincore.org/documents/dcmi-terms/	dcterms	http://purl.org/dc/terms/
FaBiO: FRBR-aligned bibliographic ontology	purl.org/spar/fabio	fabio	http://purl.org/spar/fabio/
FRBRcore	purl.org/vocab/frbr/core	frbr	http://purl.org/vocab/frbr/core#
FRBRextended	purl.org/vocab/frbr/extended#	frbre	http://purl.org/vocab/frbr/extended#
IFLA’s FRBRer Model	iflastandards.info/ns/fr/frbr/frbrer/	frbrer	http://iflastandards.info/ns/fr/frbr/frbrer/
International Standard Bibliographic Description (ISBD)	iflastandards.info/ns/isbd/elements/	isbd	http://iflastandards.info/ns/isbd/elements/
Lexvo.org Ontology	lexvo.org/ontology	lvont	http://lexvo.org/ontology#
MARC Code List for Relators	id.loc.gov/vocabulary/relators	mrel	http://id.loc.gov/vocabulary/relators/
Open Provenance Model Vocabulary	purl.org/net/opmv/ns	opmv	http://purl.org/net/opmv/ns#
PRISM: Publishing Requirements for Industry Standard Metadata	prismstandard.org	prism	http://prismstandard.org/namespaces/basic/2.0/
Provenance Vocabulary Core Ontology	purl.org/net/provenance/ns	prv	http://purl.org/net/provenance/ns#
RDA Relationships for Works, Expressions, Manifestations, Items	rdvocab.info/RDARelationshipsWEMI	rdarel	http://rdvocab.info/RDARelationshipsWEMI
Schema.org	schema.org	schema	http://schema.org/

Aggregators

International coverage and diverse scope

- European digital heritage gateway Europeana
- DRIVER
- OpenAIRE

Compatibility with aggregators

- Important for repositories
- Common requirement for repositories
- Metadata have to meet specific criteria and adopt specific vocabularies

LOD adoption is the prevailing approach

- Brings an order to the chaos of disparate solutions

Benefits by LOD Adoption

Avoid vendor lock-ins

Allow complex queries to be evaluated on the results

- Utilize the full capacities of SPARQL

Content can be harvested and integrated by third-parties

- Ability to create meta-search repositories
  - Researchers can browse, search and retrieve content from these repositories

Bring existing content into the Semantic Web

- New capabilities are opened

Synchronous Vs. Asynchronous Exports

SPARQL-to-SQL translation in the digital repositories

- Asynchronous approach seems more viable
  - Real-time results may not be as critical
  - RDF updates could take place in a manner similar to search indexes
- The trade-off in data freshness is largely remedied by the improvement in the query answering mechanism
  - Data freshness can be sacrificed in order to obtain much faster results
- Exposing data periodically comes at a low cost
  - Information does not change as frequently as e.g., in sensor data
  - Data is not updated to a significant amount daily
  - Selection queries over the contents are more frequent than the updates

From DSpace to Europeana (1)

DSpace cultural heritage repository

Data model

- Dublin Core
- Europeana Data Model (EDM)

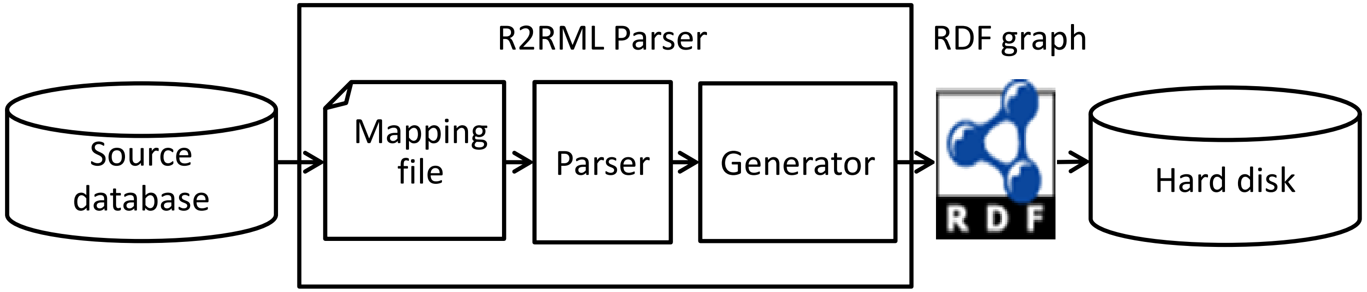
The problem

- How to transform item records as RDF using the EDM model

From DSpace to Europeana (2)

Components

- Source
  - The relational database
- Target
  - An RDF graph
- The R2RML Parser



Information flow

- Parse database contents into result sets
- Generate a Java object
- Instantiates the resulting RDF graph in-memory
- Persist the RDF graph

From DSpace to Europeana (3)

Bibliographic record example

Metadata field	Metadata value
dc.creator	G.C. Zalidis
	A. Mantzavelas
	E. Fitoka
dc.title	Wetland habitat mapping
dc.publisher	Greek Biotope-Wetland Centre
dc.date	1995
dc.coverage.spatial	Thermi
dc.type	Article
dc.rights	http://creativecommons.org/licenses/by/4.0/

From DSpace to Europeana (4)

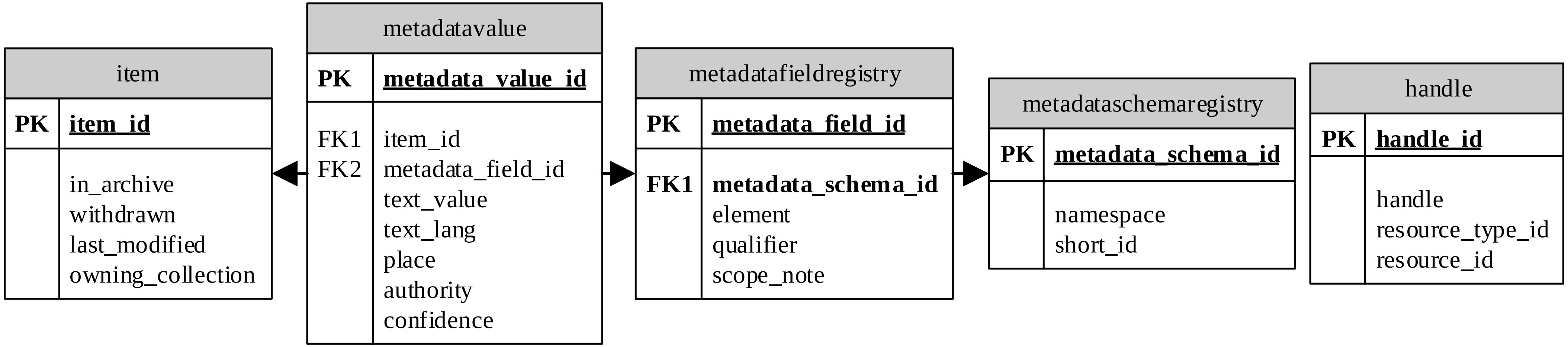
Output description (RDF/XML abbreviated)

```
<edm:ProvidedCHO rdf:about="http://www.example.org/handle/11340/615">
  <dc:creator rdf:resource="http://www.example.org/persons#G.C. Zalidis"/>
  <dc:creator rdf:resource="http://www.example.org/persons#A. Mantzavelas"/>
  <dc:creator rdf:resource="http://www.example.org/persons#E. Fitoka"/>
  <dc:title>
    Wetland habitat mapping
  </dc:title>
  <dc:publisher rdf:resource="http://www.example.org/publishers#Greek Biotope-
Wetland Centre"/>
  <dc:date>1995</dc:date>
  <dcterms:spatial rdf:resource="http://www.example.org/spatial_terms#Thermi"/>
  <dc:type rdf:resource="http://www.example.org/types#Article"/>
  <dc:rights>
    http://creativecommons.org/licenses/by/4.0/
  </dc:rights>
</edm:ProvidedCHO>
```

From DSpace to Europeana (5)

DSpace relational database schema

- Basic infrastructure
- Allows arbitrary schemas and vocabularies



item	
PK	**item_id**
	in_archive withdrawn last_modified owning_collection

metadatavalue	
PK	**metadata_value_id**
FK1 FK2	item_id metadata_field_id text_value text_lang place authority confidence

metadatafieldregistry	
PK	**metadata_field_id**
FK1	**metadata_schema_id** element qualifier scope_note

metadataschemaregistry	
PK	**metadata_schema_id**
	namespace short_id

handle	
PK	**handle_id**
	handle resource_type_id resource_id

From DSpace to Europeana (6)

Triples Maps definitions in R2RML

Create URIs based on metadata values from Dspace

- Example: dc.coverage.spatial
- Subject (rr:subjectMap template)
  - ' http://www.example.org/handle/{"handle"} '
- Predicate (rr:predicate value)
  - dcterms:spatial
- Object (rr:objectMap template)
  - ' http://www.example.org/spatial_terms#{"text_value"} '

From DSpace to Europeana (7)

R2RML mapping

```
map:dc-coverage-spatial
  rr:logicalTable <#dc-coverage-spatial-view>;
  rr:subjectMap [
    rr:template
'http://www.example.org/handle/{"handle"}';
  ];
  rr:predicateObjectMap [
    rr:predicate dcterms:spatial;
    rr:objectMap [
      rr:template

'http://www.example.org/spatial_terms#{"text_value"}';
      rr:termType rr:IRI
    ];
  ].
```

```
<#dc-coverage-spatial-view>
  rr:sqlQuery """
  SELECT h.handle AS handle, mv.text_value AS
text_value
  FROM handle AS h, item AS i, metadatavalue AS mv,
metadataschemaregistry AS msr, metadatafieldregistry
AS mfr WHERE
  i.in_archive=TRUE AND h.resource_id=i.item_id AND
  h.resource_type_id=2 AND
  msr.metadata_schema_id=mfr.metadata_schema_id AND
  mfr.metadata_field_id=mv.metadata_field_id AND
  mv.text_value is not null AND i.item_id=mv.item_id
AND
msr.namespace='http://dublincore.org/documents/dcmi-
terms/'
  AND mfr.element='coverage' AND
mfr.qualifier='spatial'
  """.
```

From DSpace to Europeana (8)

Technical vs. Bibliographic dimension

Widespread ontologies have to be used where applicable

Linking the data to third party datasets using other datasets' identifiers is also an aspect

Outline

Introduction

Motivation-Benefits

Classification of approaches

Creating ontology and triples from a relational database

Complete example

Future outlook

Challenges: Ontology-based Data Updates

SPARQL-based access to the contents of the database is unidirectional

Transform SPARQL Update requests to appropriate SQL statements and execute them on the underlying relational database

An issue similar to the classic database view update problem

Challenges: Mapping Updates

Database schemas and ontologies constantly evolve

- Established mappings should also evolve, not be redefined or rediscovered from scratch

An issue closely related to the previous one

Modifications in either participating model do not incur adaptations to the mapping but cause some necessary changes to the other model

Could prove useful in practice

- Database trigger functions
- The Link Maintenance Protocol (WOD-LMP) from the Silk framework

Challenges: Linking Data

Reusing popular Semantic Web is not sufficient for the generation of 5-star Linked Data

- Database values should not only be translated to RDF literals
- Real-world entities that database values represent should be identified and links between them should be established

Related tools

- RDF extension for Google Refine
- T2LD

Chapter 5

Generating Linked Data in Real-time from Sensor Data Streams

NIKOLAOS KONSTANTINOU

DIMITRIOS-EMMANUEL SPANOS

Outline

Introduction

Fusion

The Data layer

Rule-based Reasoning

Complete Example

Problem Framework

Rapid evolution in ubiquitous technologies

Pervasive computing is part of everyday experience

- User input
- Information sensed by the environment

Parallel decrease of the price of sensors

IoT and M2M

Streamed Data (1)

Need for real-time, large-scale stream processing application deployments

Data Stream Management Systems

- Managing dynamic knowledge
- Emerged from the database community
- Similar concern among the Semantic Web community

Streamed Data (2)

Numerous challenges

- Large scale
- Geographic dispersion
- Data volume
- Multiple distributed heterogeneous components
  - Sensor, sensor processing and signal processing (including a/v) components
- Vendor diversity
- Need for automation

Data Stream Management Systems

Fill the gap left by traditional DBMS's

- DBMS's are not geared towards dealing with continuous, real-time sequences of data

Novel rationale

- Not based on persistent storage of all available data and user-invoked queries

Another approach

- On-the-fly stream manipulation
- Permanent monitoring queries

Context-awareness, IoT and Linked Data (1)

Context

> Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves

Context-aware systems

- Knowledge of the environment in which they are acting
- Extract and use information from the environment in order to adjust their functionality according to the incoming information
- Behavior according to the existing conditions
- Tightly connected to the IoT vision

Context-awareness, IoT and Linked Data (2)

The IoT vision

- Aims at connecting (large numbers of) sensors deployed around the world
  - Focus shifts to automated configuration of filtering, fusion and reasoning mechanisms that can be applied to the collected sensor data streams

Ubiquitous, pervasive, context-aware

- The ability of a system to “read” its environment and take advantage of this information in its behaviour

Context-awareness, IoT and Linked Data (3)

Challenge

- Heterogeneity in systems
  - Capture, store, process, and represent information
- Information is generated in large volumes and at a high velocity

Common representation format

- Making systems interoperable
- Allows information to be shared, communicated and processed

Context-awareness, IoT and Linked Data (4)

Linked Data

- An efficient approach towards filling in this gap
- A common, reliable and flexible framework for information management
- Information homogeneity
- Facilitates information integration

Outline

Introduction: Problem Framework

Fusion

The Data layer

Rule-based Reasoning

Complete Example

Information Fusion (1)

Fusion

> The study of techniques that combine and merge information and data residing at disparate sources, in order to achieve improved accuracies and more specific inferences than could be achieved by the use of a single data source alone

- Leverages information meaning
- Partial loss of initial data may occur
- Several fusion levels

Information Fusion (2)

Algorithm

- Online (distributed)
  - Each node can take decisions based only on its perception of the world
  - Each algorithm execution is based on the knowledge of only a local node or a cluster of nodes
- Offline (centralized)
  - There is a need of a central entity maintaining system-wide information

Fusion nodes can

- Act as a server (push)
- Harvest information (pull)

Information Fusion (3)

Information fusion vs integration

- Fusion takes place in the processing steps
- Integration refers to the final step
  - The end user’s gateway to (integrated) access to the information

Fusion Levels

Early fusion

Signal level

- Signals are received simultaneously by a number of sensors
- Fusion of these signals may lead to a signal with a better signal-to-noise ratio

Feature level

- A perceptual component must first extract the desired low-level features from each modality
- Typically represents them in a multidimensional vector

Late fusion

Decision level

- Combines information from multiple algorithms in order to yield a final fused decision
- May be defined by specific decision rules

JDL Fusion Levels (1)

A process model for data fusion and a data fusion lexicon

Intended to be very general and useful across multiple application areas

Identifies the processes, functions, categories of techniques, and specific techniques applicable to data fusion

JDL Fusion Levels (2)

Process conceptualization

- Sensor inputs
- Source preprocessing
- Database management
- Human-computer interaction
- Four key subprocesses (following next)

JDL Fusion Levels (3)

Level 1 – Object Refinement

- Combines sensor data together to obtain a reliable estimation of an entity position, velocity, attributes, and identity

Level 2 – Situation Refinement

- Attempts to develop a description of current relationships among entities and events in the context of their environment

JDL Fusion Levels (4)

Level 3 – Threat Refinement

- Projects the current situation into the future to draw inferences
  - E.g. about friendly and enemy vulnerabilities, threats, and opportunities for operations

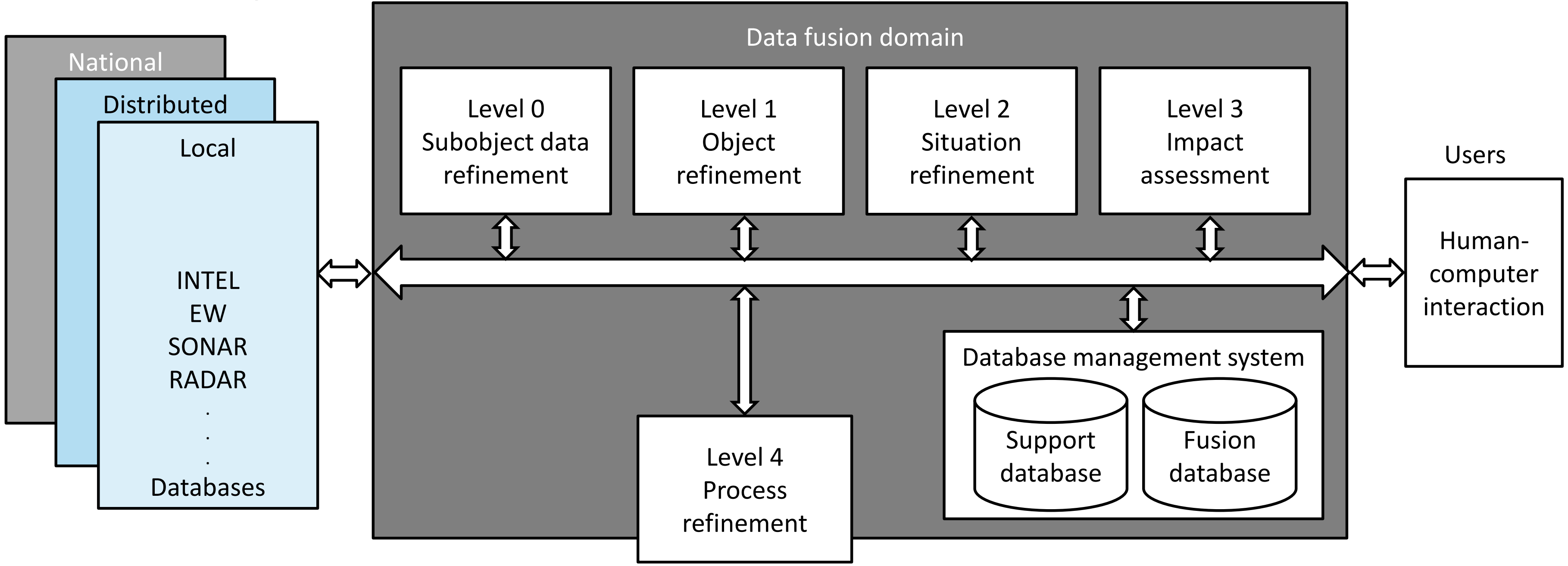
Level 4 – Process Refinement

- A meta-process which monitors the overall data fusion process
  - Assesses and improves the real-time system performance

JDL Fusion Levels (5)

Level 5 – Cognitive or User Refinement

- Added in revisions
- Introduce the human user in the fusion loop
  - The aim is to generate fusion information according to the needs of the system user



Outline

Introduction: Problem Framework

Fusion

The Data layer

Complete Example

The Data Layer (1)

Metadata of multimedia streams

Semi-structured vs structured

- No significant technological challenges into converting semi-structured documents into RDF

Data is produced in the form of streams

- Originating from a set of heterogeneous distributed sources
- No starting or ending point
- A strategy must be devised
    - Define how new facts will be pushed into the system and old facts will be pushed out of it

The Data Layer (2)

Data flow

- Use of a middleware
- Semantic annotation → Knowledge Base
- Ability to answer semantic queries

Modeling Context (1)

Challenges

- Complexity of capturing, representing and processing the concepts
- Expressivity of the description of the world
    - Expressive enough in order to enable specific behaviors
    - Not as complex as to render the collected information unmanageable

Common representation format and vocabulary

- Ensure syntactic and semantic interoperability in the system
- Enable integration with third party data sources

Modeling Context (2)

Uniform context representation and processing at the infrastructure level

- Better reuse of derived context by multiple data producers and consumers

Ontology-based descriptions

- Offer unambiguous definitions of the concepts and their relationships
- Allow further exploitation of the created Knowledge Base
  - Higher level, intelligent, semantic queries
- Formalize knowledge about the specific domain

Semantic Web Technologies in Sensor Networks

Research focuses on modelling

Goal: enable higher level processing for event/situation analysis

- Define ontologies for sensor measurement and sensor description
- Use a meta-ontology (e.g. SUMO)
  - Define a sensor ontology under it
  - Use it to annotate, query, and discover sensors
- Base concepts: Resource, Actor, Environment
  - Parts that describe the state of the resources
  - Entities operating in the resources
  - Surrounding environment conditions

Semantics for Situation Awareness (1)

Situation Theory Ontology (STO)

- A unified expression of the Situation Theory
- Models events/objects and their relationships in OWL
- Can be extended with classes and relations that correspond to the actual application scenario
- Two additional ontologies integrated, in order to be able to use it in a real sensor fusion environment
  - The Time ontology
  - The WGS84 Geo Positioning ontology

Semantics for Situation Awareness (2)

SSN ontology

- Describes sensor data in the Linked Sensor Data context

Semantic Sensor Web (SSW)

- Framework for providing meaning for sensor observations
- Bridges the gap between
  - XML-based metadata standards of the Sensor Web Enablement
  - RDF/OWL-based vocabularies driving the Semantic Web

Modeling Context using Ontologies

A Local-As-View setting

- Optimal when new types of sensors need to be integrated and there is a relatively high variety in structure
- Preferred approach
  - Better than translating user queries from the global schema to each one of the local ones in order to retrieve, aggregate and return results to the user

Data produced by sensors is eventually stored in its Knowledge Base

- Reasoning
  - Infer knowledge which corresponds to events
- Situation assessment
  - Cannot be performed at lower fusion levels

Sensor Data

Audiovisual

- Audio/video stream

Non-audiovisual

- Any kind of streamed sensor observations
  - Temperature, RFID tags, etc.

Incoming data is subject to filtering

- Not all information is stored and processed
- Only the subset that is meaningful and needed by the processing modules
  - E.g. discard out-of-range values

Challenges in Homogenizing Sensor Data

Numerous standards to annotate various areas of sensor data

- MPEG-7 for audiovisual
- Federal Geographic Data Committee (FGDC) for geographical information

Accuracy of the annotations

Convenience of updates and maintenance

Added value to the content itself

- Not always clear how the user can benefit from the existence of such annotations

Tracker errors

- False, missing, incorrect annotations

A Two-step Approach

Annotate sensor data according to the nature of its tracker

Homogenize the data under a common vocabulary

- Containing the captured values and semantics

Real-time vs. Near-real-time (1)

A real-time system

- Must satisfy explicit bounded response time constraints to avoid failure and present consistency regarding the results and the process time needed to produce them
- Emphasis in predicting the response time and the effort in reducing it
  - Response time: the time between the presentation of a set of inputs and the appearance of all the associated outputs

A real-time sensor fusion system

- Produces certain alerts (outputs)
- Connected to the appearance of certain inputs (events)

Real-time vs. Near-real-time (2)

Near real-time

- E.g. systems that schedule their operations at fixed time intervals
- Frequency of the updates depends on the application
    - E.g. in a surveillance scenario, more frequent updates would be needed than in an environmental monitoring scenario

Sensor inputs/outputs

- Real-time signals
    - E.g. audiovisual streams
- Near-real time/asynchronous messages
    - E.g. RFID readings

Data Synchronization and Timestamping (1)

Synchronization

- Rules of the form

if *event*$_1$ occurred before *event*$_2$ then…

Data Synchronization and Timestamping (2)

Two kinds of timestamps

- The time the event was recognized
    - Local time at the node that made the measurement
    - Problem: how to synchronize the sensor network to a common clock
- The time the event arrived in the fusion node
    - Fusion node
        - A node that fuses information incoming from other nodes
    - No need of a common clock
        - The fusion node will timestamp events upon their arrival
    - Followed when there are no great delays in communicating messages

Data Synchronization and Timestamping (3)

Timestamping can be

- Distributed
  - At each node
- Centralized
  - At a central node, maintaining a common clock

Windowing

A mechanism to assure continuous data processing

In order to process newly generated information properly, the system will not have to take into account all existing information

Maintain a working memory window

Streams are unbounded

- Cannot fit into memory in order to be processed as a whole

Windowing-related Decisions

The measurement unit

The size

The window behavior

- Sliding windows
- Tumbling windows
- Landmark windows
- Partitioned windows
- Predicate windows

Rules applied real-time are restricted to the current information window

The (Distributed) Data Storage Layer (1)

Generated information needs to be stored and processed before being communicated to the system

Each node maintains its perception of the real world

- Physically stored in a local database

Multi-sensor stream processing systems purposed to function under a heavy load of information

- Preferred database design geared towards scalability
- I.e. decentralized to the maximum extent possible
    - Not centralizing collected information

The (Distributed) Data Storage Layer (2)

An approach in order to maintain scalability

- Each node keeps the amount of information required for its local operation
  - Local database schema has to relate only to the hosted components
- Restrict information communicated throughout the system
- Communicates to the central node only higher level information
  - E.g. detected events or entities

Relational to RDF in Sensor Data Streams

Produced messages will have to be eventually converted to RDF and ultimately inserted in an ontology

A mapping layer

- Map the relational schema to the semantic schema

Mapping Layer (1)

Push strategy

- Forward data to the ontology using semantic notation as soon as they are generated
- Advantages
  - Transformations are executed fast
  - The ontology is always up-to-date
- Disadvantages
  - Each lower level node will have to implement its own push method
  - Risk that the ontology will be populated with data even when no query is sent to the semantic layer

Mapping Layer (2)

Pull strategy

- Transform relational data to semantic on request
  - I.e. during query time
- Similar to RDF Views
- Advantages
  - Actual mapping defined at semantic level
  - Data transformed on request
    - The ontology will accumulate instances needed for the actual query evaluation
- Disadvantages
  - Could lead to longer response times during queries

Outline

Introduction: Problem Framework

Fusion

The Data layer

Rule-based Reasoning

Complete Example

Rule-based Stream Reasoning in Sensor Environments (1)

Reasoning

- Infer implicit information
- Use the ontology structure and instances in order to draw conclusions about the ongoing situations
- Based on a set of provided rules, applied on the created knowledge base

Extend the knowledge base by using rules

- To describe complex situations/events
- To create alarm-type of objects if certain conditions are met
- To depict the desired behavior and intelligence

Rule-based Stream Reasoning in Sensor Environments (2)

Event-condition-action pattern

on *event* if *condition* then *action*

An event is a message arrival indicating a new available measurement

Two distinct sets of rules

- Mapping rules
- Semantic rules

Rule-based Stream Reasoning in Sensor Environments (3)

Mapping rules

- Specify how sensor measurements represented in an XML-based format will be mapped to a selected ontology
- Can fetch data from the XML-like message and store it into the ontology model in the form of class instances
- Can be perceived as the necessary step bridging the gap between semi-structured data and ontological models

Rule-based Stream Reasoning in Sensor Environments (4)

Semantic rules

    - Derive new facts, actions or alerts
        - Based on existing facts and knowledge
    - Perform modifications on the ontology model
- Depend on the specific domain or deployment scenario
- Involve high-level concepts , meaningful to humans
    - E.g., “when a certain area under observation is too hot, open the ventilating system”
    - The “too hot” conclusion will probably be inferred from a set of current observations coupled with the knowledge stored in the ontology

Rule-based Stream Reasoning in Sensor Environments (5)

Rule-based systems

- Combine real-time data with stored sensor data
- Fire rules based on data obtained from sensors in real-time and classified as ontology instances

Rule-based problem solving

- An active topic in AI and expert systems
- Classic approach comes from work on logical programming and deductive databases
- Conventional rule engine implementations based on the Rete algorithm

Rule-based Stream Reasoning in Sensor Environments (6)

Stream reasoning

Performing reasoning on a knowledge base comprising stable or occasionally changing terminological axioms and a stream of incoming assertions or facts

Will lead the way for smarter and more complex applications

- E.g. traffic management, fastest route planning, environmental monitoring, surveillance, object tracking, disease outburst detection, etc.

Rule-based Stream Reasoning in Sensor Environments (7)

RIF – Rule Interchange Format

- W3C recommendation
- A core rule language
- A set of extensions (dialects) that allow the serialization and interchange of different rule formats

Also RuleML and SWRL

Also using SPARQL CONSTRUCT

- SPIN being an extension to this approach

Rule-based Reasoning in Jena (1)

Jena Semantic Web Framework

- The most popular Java framework for ontology manipulation
- Includes an inference engine, can be used as a reasoner
- Includes a number of predefined reasoners
    - Transitive reasoner
    - RDFS rule reasoner
    - OWL, OWL mini, OWL micro
    - DAML micro reasoner
    - A generic rule reasoner that can be customized to meet specific ad hoc application demands

Rule-based Reasoning in Jena (2)

Forward chain rules (body ⟶ head)

- When the body is true, then the head is also true

Built-in rule files of the form:

```
[rdfs5a: (?a rdfs:subPropertyOf ?b), (?b rdfs:subPropertyOf ?c) -> (?a rdfs:subPropertyOf ?c)]
```

Symmetric and transitive properties in OWL:

```
[symmetricProperty1: (?P rdf:type owl:SymmetricProperty), (?X ?P ?Y) -> (?Y ?P ?X)]
```

```
[transitivePropery1: (?P rdf:type owl:TransitiveProperty), (?A ?P ?B), (?B ?P ?C) -> (?A ?P ?C)]
```

Rule-based Reasoning in Jena (3)

Builtin primitives

- Functions that can be used in the place of rule predicates
  - isLiteral(?x), notLiteral(?x),
  - isFunctor(?x), notFunctor(?x)
  - isBNode(?x), notBNode(?x), etc.
  - Custom builtin primitives can be developed

Rule-based Reasoning in Virtuoso (1)

Reasoning essentially a set of rules applied on the RDF graph

Relatively simple reasoning

- Scalability instead of rich inference capabilities

Rule sets

- Loaded using the rdfs_rule_set function
- User can specify a logical name for the rule set, plus a graph URI
- Are provided as a context to user queries
- Can be referenced by SPARQL queries or endpoints

Rule-based Reasoning in Virtuoso (2)

Queries return results as if the inferred triples were included in the graph

- Inferred triples generated by reasoning (the rule set) are generated at runtime, are not physically stored

Outline

Introduction: Problem Framework

Fusion

The Data layer

Rule-based Reasoning

Complete Example

Complete Example

Information originating from a distributed sensor network

Architecture of a Multi-Sensor Fusion System

- Based on GSN
- Operating at all JDL levels
- Create Linked Data

Fusion and its potential capabilities

Combine semantic web technologies with a sensor network middleware

Blend ontologies with low-level information databases

Proof-of-Concept Implementation

An LLF node

Two processing components

- A Smoke Detector
- A Body Tracker
- Each component hosted on a computer with a camera
  - RTP streams

An HLF node

A Central node

- Overall system supervision

The GSN Middleware (1)

Needed in order to perform LLF

Open-source, java-based

Allows processing data from a large number of sensors

Covers LLF functionality requirements in sensor data streams

The GSN Middleware (2)

Virtual sensor

- Any data provider (not only sensors)
- Configuration file in XML
  - Processing class
  - Windowing
    - Time- or tuple-based sliding window size
  - Data source
  - Output fields

The GSN Middleware (3)

GSN Servers

- Can communicate between them
- Support input from more than one data stream
- Can form a network
  - Allow information collection, communication, fusion, integration

Data acquisition

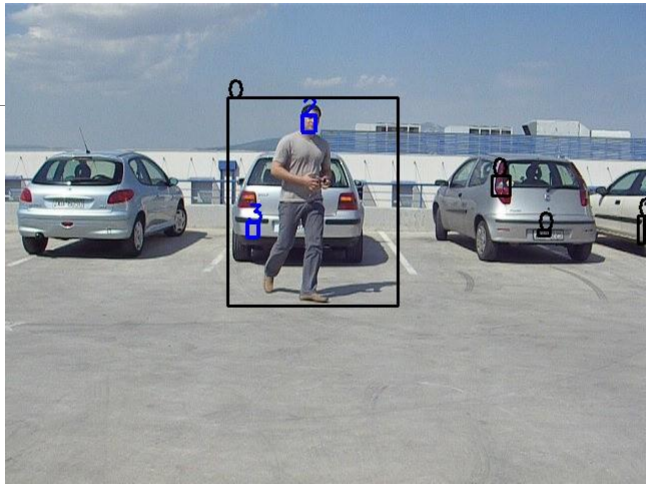
- An SQL-like procedural language
  - Combine and fuse the information
  - User-defined LLF functions
- GSN takes care behind the scenes about crucial issues
  - E.g. thread safety, synchronization, etc.

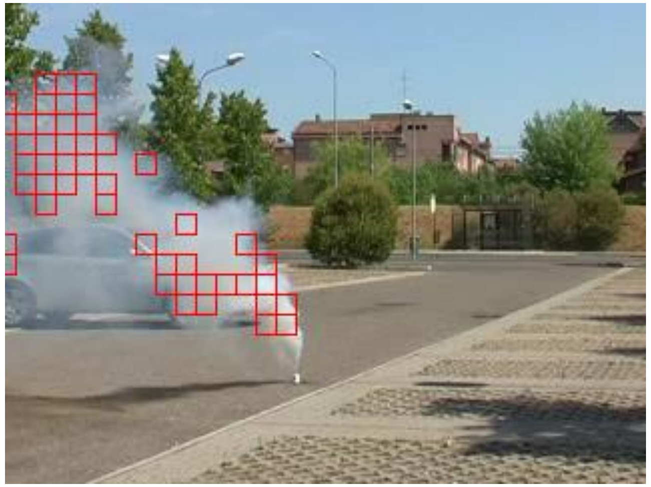
Low Level Fusion (1)

Camera generates an RTP feed with its perception of the world

Feed processed by signal processing components

- The Body tracker
  - Reports NumberOfPersons
- The Smoke detector
  - Reports NumberOfSmokeParticles
- Both components
  - Receive a POLL command at fixed time intervals
  - Return an XML file





Low Level Fusion (2)

Sampling the video source takes place asynchronously

- Camera streams at 25 fps
- 500 ms between two consecutive polls suffices for LLF

Tracking a person is more demanding than detecting it

- Implies comparing consecutive frames

Asynchronous processing

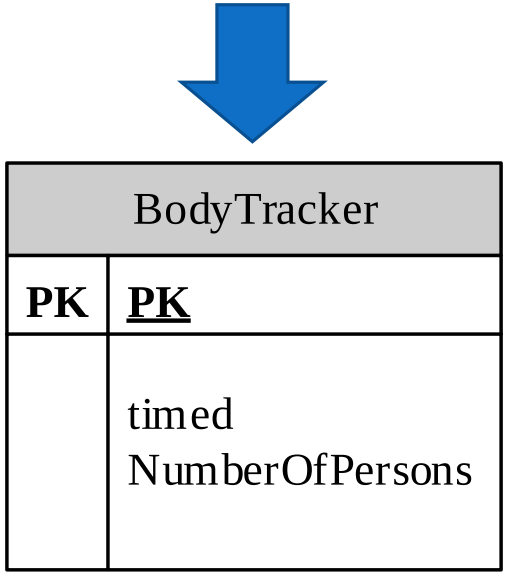
- The camera fps rate not aligned with the produced message rate

Low Level Fusion (3)

Virtual sensor XML configuration files

- Body tracker
  - PK: primary key (auto-incremented integer)
  - Timed (timestamp)
  - NumberOfPersons (integer)
- Similarly for the smoke detector
  - ExistenceOfSmoke (boolean)

```
<virtual-sensor name="BodyTracker">
  …
  <output-structure>
   …
   <field name="NumberOfPersons"
          type="int" />
  </output-structure>
  …
  <storage history-size="1m" />
</virtual-sensor>
```



BodyTracker	
PK	**PK**
	timed NumberOfPersons

Low Level Fusion (4)

LLF virtual sensor definition

- Two data providers (source streams)
- Produce events only when the fusion conditions are satisfied
  - A person and smoke are detected, concurrently

```
SELECT source1. NumberOfPersons AS v1,
       source2. ExistenceOfSmoke AS v2
FROM source1, source2
WHERE v1 > 0 AND v2 = "true"
```

Low Level Fusion (5)

Fusion

- Results provided by taking into account the inputs from both the sensors
- More processing components, sensors, sensor types and more complex fusion conditions can be integrated into the system

Low Level Fusion

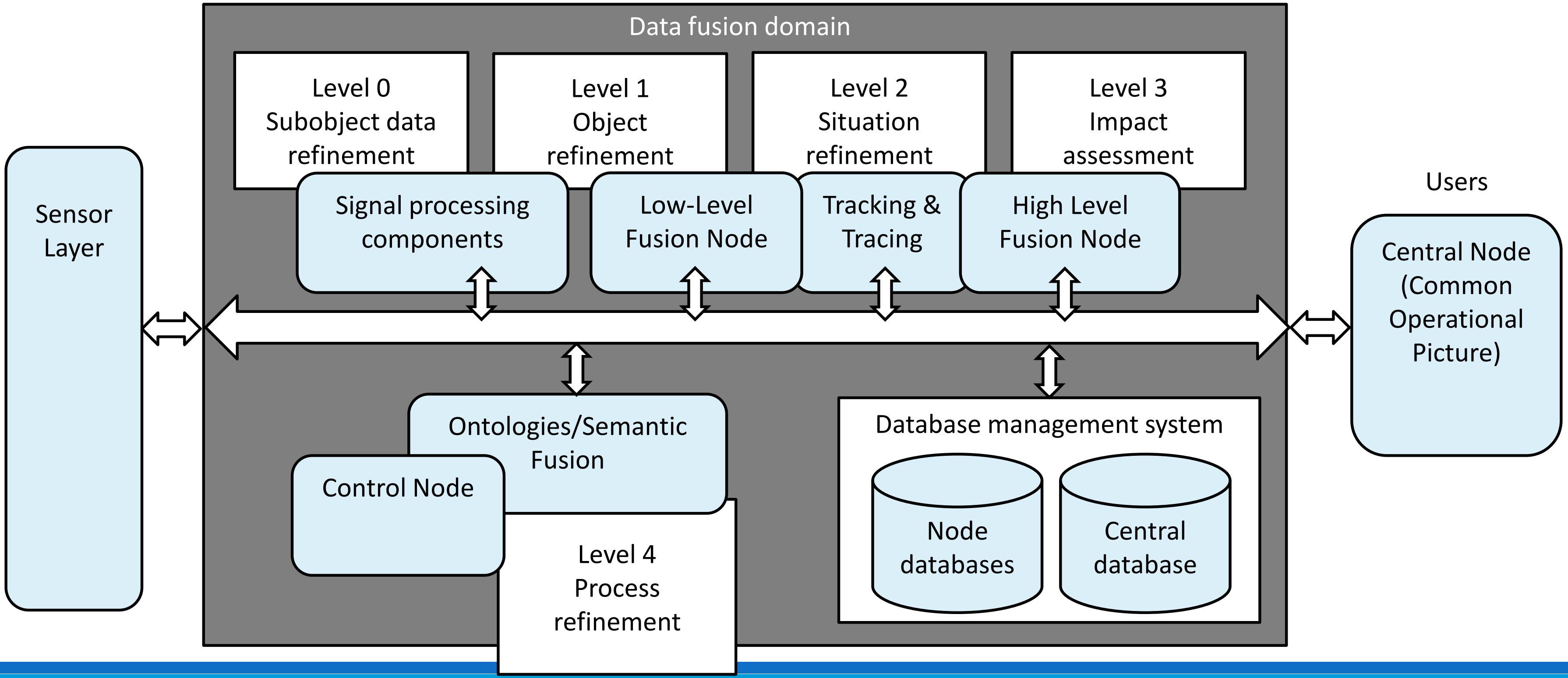
- Allow only important information to be forwarded to the upper layers

No semantic enrichment so far

A Sensor Fusion Architecture (1)

From theory to practice

Address matters that correspond to all JDL levels



A Sensor Fusion Architecture (2)

Collected data is leveraged into meaningful and semantically enriched information

- I.e. transformed from signals to higher level information

All levels of multi-sensor data fusion are applied to the data

A Sensor Fusion Architecture (3)

JDL Level 0 – Subobject data refinement

- Signal Processing components operate at this level
- System collects observation data
  - Sensory data input mostly from Electro-Optical sensors (e.g. cameras, microphones, etc.)
- Virtualizes the inputs of the sensors
- Performs spatial and temporal alignment

A Sensor Fusion Architecture (4)

JDL Level 1 – Object refinement

- System analyzes the collected data
- Extracts features, objects, and low level events
- Detected objects may include persons or vehicles
- Low level events may include movements
- Implementation can be based on e.g. GSN

A Sensor Fusion Architecture (5)

JDL Level 2 – Situation refinement

- System aggregates and fuses the objects, events and the general context in order to refine the common operational environment
- System is able to perform object fusion and tracking, and identify situations in the system, in a manner that is not feasible by a sensor alone

A Sensor Fusion Architecture (6)

JDL Level 3 – Impact assessment

- Events and objects refined at JDL Level 2 are subsequently analyzed, using semantically enriched information, resulting at the inferred system state
- High Level Fusion (HLF, e.g., using Virtuoso's rule engine) operates at this level
  - Fuses the information, enabling reasoning that can infer and assess potential impacts
    - E.g. situations and respective alerts

A Sensor Fusion Architecture (7)

JDL Level 4 – Process refinement

- System can refine its operation by providing feedback to the sensor layer
- Ontologies/Semantic fusion component
  - Perform analysis on the system-wide high-level collected information in order to infer events and potential risks and threats
  - Hosted at the Central Node
  - Monitor and curate the whole system

A Sensor Fusion Architecture (8)

Level 5 – Cognitive or User Refinement

- A higher level, introduced in revised versions of the JDL model
- Control Node component
  - Hosted at the Central Node
  - Responsible for issuing commands back to the sensors
- Common Operational Picture component
  - System front-end
  - Provide a visualization of the system state
    - Deployed sensors, detected objects, sensed events and inferred threats
  - Allows for human-computer interaction

A Sensor Fusion Architecture (9)

Two types of databases to support system operation

- Support databases
  - Materialized as a node database, kept locally at each node
  - Collected data are kept close to their source of origin, allowing for each node to be configured and maintained according to its environment and system's needs
  - Allow system to scale
- Central Node database
  - Plays the role of the fusion database, where higher level fused information is kept

High Level Fusion Example (1)

Introduction

- Ontologies
  - Powerful means to describing concepts and their relations
    - Entities, events, situations, etc.
    - JDL levels 2 and 3
  - Main information gathering point in the proposed architecture
- Reasoning
  - Infer new knowledge
  - Cannot be performed at the LLF level

High Level Fusion Example (2)

The scenario

- Demonstrate inference capabilities of the proposed architecture
- Full use of the data chain
  - From low level sensor data to high level complex event detection and situation awareness/threat assessment

- Detect threats to public safety and associate an action plan
  - I.e. detect persons and smoke in an area of interest, e.g. a petrol station

High Level Fusion Example (3)

Configure GSN to use Virtuoso as its backend at the HLF node

Develop RDF views in Virtuoso over the sensor data

Use the Situation Theory Ontology (STO)

- The STO:FocalSituation class marks significant situations that prompt action by security personnel

High Level Fusion Example (4)

Perform reasoning in Virtuoso

- Triple store populated with data from the Smoke detector and Body tracker components
- Reasoning associates the smoke detection event with a criticality factor
  - According to the events and geospatial information modeled in STO

The scheduler component in Virtuoso

- Update the triple store via RDF views from the sensor inference database
- Subsequently invoke on the reasoning

High Level Fusion Example (5)

Query to retrieve focal situations with their event related details

- Time, location, textual descriptions

```
SELECT ?focalsituation ?timeTxt ?locTxt
WHERE {
  ?focalsituation STO:focalRelation ?event .
  ?event STO:hasAttribute ?time .
  ?event STO:hasAttribute ?location .
  ?time rdf:type STO:Time .
  ?time time:inXSDDateTime ?timeTxt .
  ?location rdf:type STO:Location .
  ?location rdfs:label ?locTxt .
}
```

High Level Fusion Example (6)

Integration with emergency departments achieved by sending details on significant threats found in SPARQL result sets to a Web service endpoint

- The smoke event together with location data is forwarded to emergency personnel

More details in:

- Doulaverakis et al. An approach to intelligent information fusion in sensor saturated urban environments. EISIC 2011, Athens, Greece

Chapter 6

Conclusions

Summary and Outlook

NIKOLAOS KONSTANTINOU

DIMITRIOS-EMMANUEL SPANOS

Outline

Introduction

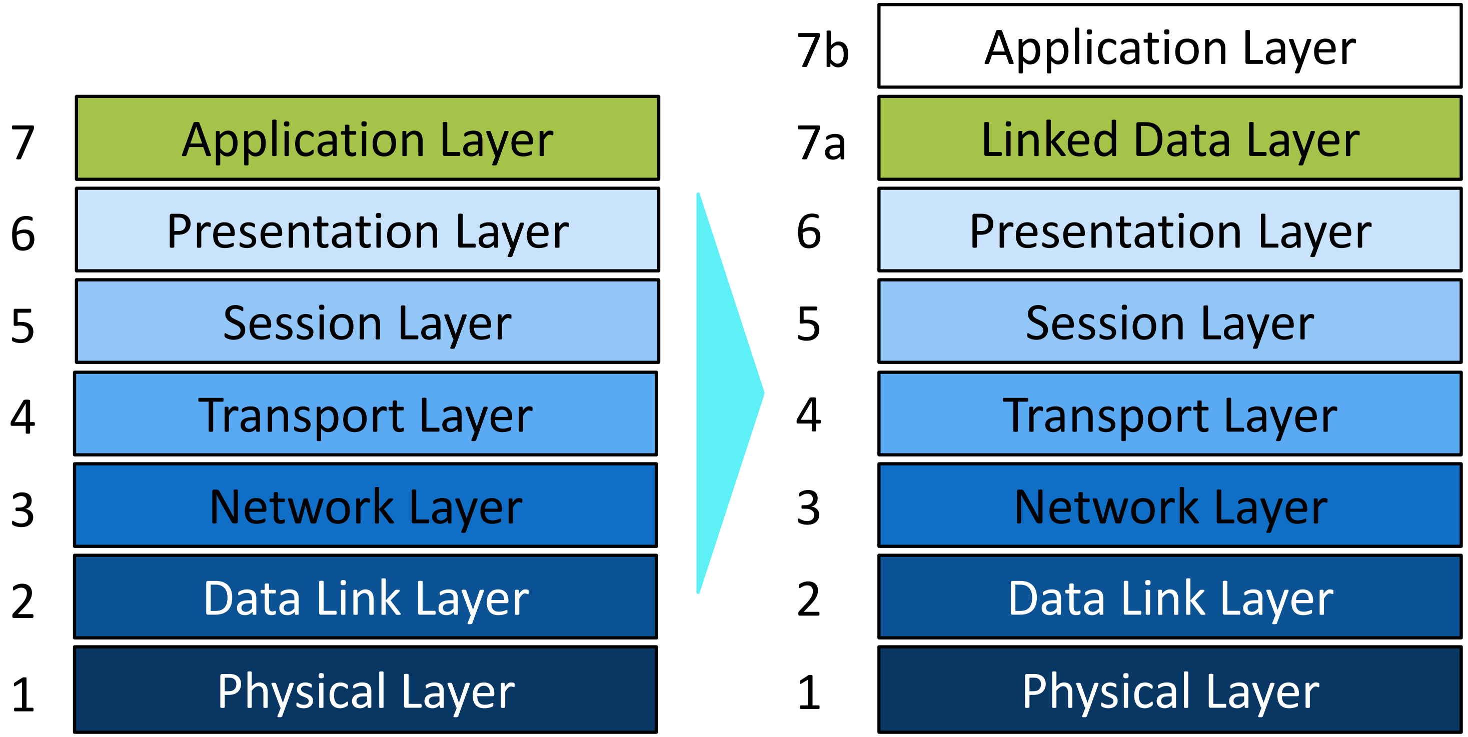
Recap

Discussion

Open Research Challenges

Introduction

An envisioned Linked Data Interoperability Layer for tomorrow’s Web



7 Application Layer
6 Presentation Layer
5 Session Layer
4 Transport Layer
3 Network Layer
2 Data Link Layer
1 Physical Layer

7b Application Layer
7a Linked Data Layer
6 Presentation Layer
5 Session Layer
4 Transport Layer
3 Network Layer
2 Data Link Layer
1 Physical Layer

Outline

Introduction

Recap

Discussion

Open Research Challenges

Chapter 1 – Introduction

Definitions – prospects – solutions

- Semantic Web
  - Main building blocks
  - Key terms
  - Issues
- Linked Data

Chapter 2 – Technical Background

From theory to practice

- Introduction of the technical background that materializes Chapter 1 concepts
- Fundamental technologies
  - From knowledge representation models to query languages and mappings
- Popular Linked Data vocabularies

Chapter 3 – Deploying Linked Data

A technical overview

Modeling data

Opening Data

Linking Data

Processing Data

- Available technical solutions and tools

Chapter 4 – Creating Linked Data from Relational Databases

RDBMS with Semantic Web applications interfaces

Motivations

Benefits

Related literature survey

- Approach categorization

Proof-of-concept use case

- Convert data from an open access repository to Linked Data

Chapter 5 – Generating Linked Data in Real-time from Sensor Data Streams

Basic concepts

- Introduction: real-time processing, context-awareness, windowing and information fusion

Related Issues

System description

- An intelligent, semantically-enabled data layer to integrate sensor information

Overall Contribution (1)

Formal and informal introduction of Data Science concepts

- Semantics
- Ontologies
- Data and Information
- Knowledge Bases
- Reasoning
- Annotation
- Metadata
- Real-time
- Context-awareness
- Integration
- Interoperability
- Fusion, etc.

Overall Contribution (2)

A detailed state-of-the-art survey

- Technologies, methodologies, tools, and approaches

Discussions on Linked Data creation

- From relational databases
- From sensor data streams

Two detailed architectural and behavioral descriptions of two domain-specific scenarios

Outline

Introduction

Recap

Discussion

Open Research Challenges

Discussion (1)

LOD ecosystem

- An open and distributed system
- Heterogeneity is inevitable
  - At a syntactic, terminological, conceptual, or semiotic/pragmatic level
- The Linked Data paradigm
  - Offers solutions to reduce heterogeneity at all four levels
    - Defines relations across the heterogeneous sources

Discussion (2)

Server-side: many steps, many components involved

- No "standard" approach
- No deterministic manner in setting behavioral priorities
- The importance of scalability
  - Volume of the produced, stored and processed information
  - The number of the data sources
  - The nature of the data
    - E.g. sensor/multimedia/social network data streams

Discussion (3)

Integration with third parties

- Semantic Web technology adoption is crucial in order to assure unambiguous definition of the information and the semantics it conveys
- E.g. as in the example in the scholarly/cultural heritage domain

Data repository turned into a Knowledge Base

- Virtually endless possibilities
  - E.g. analytical reasoning, intelligent analysis
    - Data mining, pattern extraction, etc.
  - Even in unprecedented ways

Benefits (1)

Increased discoverability

- Description of the dataset contents
- Links towards instances in other parts of the LOD cloud
- Inbound links are welcome

Reduced effort for schema modifications

- New relations allowed without modifying the database schema or contents
  - E.g. define new classes and properties

Benefits (2)

Synthesis

- Integration, fusion, mashups
- Allow searches spanning various repositories, from a single SPARQL endpoint
- Allow download of parts or the whole data

Inference

- Reasoning support
- Implicit facts can be inferred, based on the existing ones, then added to the graph

Benefits (3)

Reusability

- Third parties can reuse the data in their systems
- Including the information in their datasets, or
- By reference to the published resources

Technical Difficulties (1)

Multidisciplinarity

- Annotation task not to be underestimated
- Contributions required from several scientific domains
  - Domain-specific expertise
  - Close collaboration of the implementation team

Technology barrier

- Tools are not as mature yet as to provide guidance or warnings
- E.g. in the linking or mapping procedure
  - No design-time validity of the result

Technical Difficulties (2)

Error-prone result

- Even syntactically correct
- No automatic check of whether the concepts and properties involved are used as intended
- Errors or bad practices can go unnoticed

Concept mismatch

- Extraction of stored values into RDF not always possible
    - Identical mappings may not always be found

Technical Difficulties (3)

Exceptions to the general rule

- Automated changes will apply to the majority of the data
  - The remaining portion will require manual intervention
- Post-publishing manual interventions will be required

Open Government Data

Emerging, across governments and organizations from all over the world

- E.g. US, UK

Foster transparency, collaborative governance, innovation

Enhance citizens' quality of life through the development of novel applications

Data value decreases if not released in open formats, allowing combination and linking with other open data

Ongoing efforts to integrate open governmental data to the LOD cloud

Bibliographic Archives (1)

A huge wealth of human knowledge exists in digital libraries and open access repositories

Structured metadata

- Cataloguing, indexing, searching
- Typically trapped inside monolithic systems that support Web-unfriendly protocols for data access

Linked Data

- Allows direct reuse of the work of other librarians
  - Transforms the item-centric cataloguing to entity-based descriptions

Bibliographic Archives (2)

Several efforts at national and regional level

- E.g. Library of Congress, British National Bibliography

Linking of bibliographic data with LOD datasets from other domains

- Expected to give rise to novel applications that exploit library data in combination with other (e.g. geographical) data

Internet of Things

Billions of sensing devices currently deployed worldwide

- Already form a giant network of connected "things“
- Their number expected to continuously grow
- Uniquely identified and accessed through standard Internet protocols

Applications in diverse domains

- E.g. environmental monitoring, energy management, healthcare and home and city automation

Intelligence in IoT

- Use of ontologies and other Semantic Web technologies that support inference
- Integration of developed independently deployed IoT platforms

Outline

Introduction

Recap

Discussion

Open Research Challenges

Open Research Challenges

Data Science

The study of the generalizable extraction of knowledge from data

LOD provision is the first step

- Not a goal in itself; a means to an end

Consuming (as opposed to producing)

- Extract intelligence, generate additional value
- E.g. visualization, analytics, text mining, named entity recognition, etc.
- Quality assessment
  - LOD quality ranges from extensively curated datasets to crowd-sourced and extracted data of relatively low quality

Big (Linked) Data (1)

Data in several media channels

- E.g. social networks, blogs, multimedia sharing services
- Generated in growing rates
- Influences professional and personal decisions and actions of individuals

Size of the dataset is a part of the problem itself

- Millions or billions of facts
- Difficult to process using conventional data processing applications
- Storage and querying problems

Big (Linked) Data (2)

No formal definition of what exactly Big Data is (and what is not)

- Commonly characterized by different properties
- All V’s for some mysterious reason
  - Volume
  - Velocity
  - Variety
  - Value
  - Veracity

Big (Linked) Data (3)

Need for

- Timely, accurate, efficient analysis of Big Data volumes
- Managing, querying and consuming
- Handling the vast amounts of data to be generated in the near future

Linked Data

- Part of the Big Data landscape
- Ideal testbed for researching key Big Data challenges

(Even) More Research Challenges

Privacy

Legal aspects

Integration and reconciliation from diverse data sources