MINES Saint-Étienne

Université Jean Monnet Saint-Etienne

The Resource Description Framework (RDF 1.1)

RDF

- RDF is to the Semantic Web what HTML is to the WWW
- RDF is simple: everything is *just* triples
- RDF is a *data model*: it is **not** a file format!
- RDF is a *logical formalism*: it has a **formal semantics**
- RDF is more than XML: XML has a *tree-based* model, RDF has a *graph-based* model
- RDF is a Web standard: W3C recommendation

RDF: *lingua franca* of the Semantic Web

- Like HTML for documents, there can be many models that would achieve a Web of Data and a Semantic Web
- PDF documents can be linked and visualised in Web browsers…
- …but HTML makes it easy to read and write documents (whereas one cannot edit PDFs in a text editor)
- Design decisions govern HTML and RDF
- The decisions may not be the best for your application, but they fix a common norm

RDF 1.1 Abstract Syntax (1)

- **RDF graphs**: a set of **triples**
- **Triple**: a 3-uple with:
  - A **subject** (an **IRI** or a **blank node**)
  - A **predicate** (an **IRI**)
  - An **object** (an **IRI**, a **blank node** or a **literal**)
- **IRI**: Internationalized Resource Identifier (in RDF 1.0, it was *URI references*) is a UNICODE string conforming to RFC 3987

Note: to shorten notations, we use namespace prefixes, e.g., `rdf:` is for `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

RDF 1.1 Abstract Syntax (2)

- **Literal**: has 2 or 3 elements, including:
  - A **lexical form** is a UNICODE string
  - A **datatype IRI**

and in case the datatype IRI is `rdf:langString`:

  - A **language tag**, as defined in IETF BCP47

- A literal with lang tag is a **language-tagged string**
- **Blank node**: an element of an infinite set disjoint from the IRIs and the literals (but otherwise undefined)

The RDF 1.1 abstract syntax is specified at:

http://www.w3.org/TR/rdf11-concepts/

RDF 1.1 Primer is a gentle introduction to RDF 1.1:

http://www.w3.org/TR/rdf11-primer/

Read the RDF 1.1 Primer for next week.

It’s relatively simple with many examples.

Datatypes

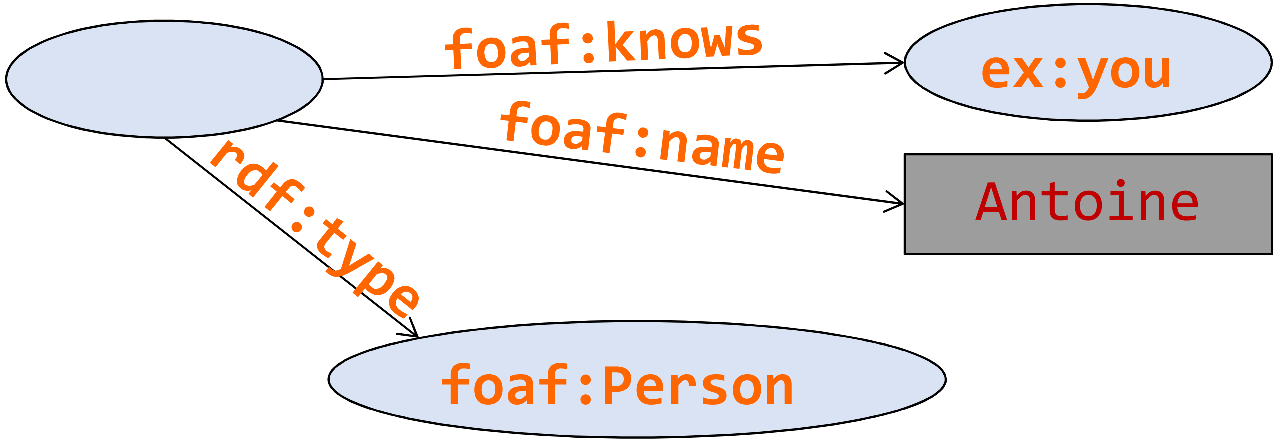
- Datatype definition borrowed from XML Schema
- **Datatype**: has 3 components:
  - The **lexical space**, a set of UNICODE strings
  - The **value space**, a set of values
  - The **lexical-to-value mapping**, a function from the lexical space to the value space
- E.g., `xsd:boolean` has the lexical space {"true","false","1","0"}, the value space {***true***, ***false***} and the lexical-to-value mapping {("true", ***true***), ("false", ***false***), ("1", ***true***), ("0", ***false***)}

Vocabularies

- **RDF Vocabulary**: a set of IRIs and literals
- There are standardised vocabularies that serve a specific purpose, or have a special meaning (see later)
- Any set of IRIs or literals form a vocabulary, but it is possible to specify a specific set of IRIs to be used in a certain way in RDF graphs → such distinguished vocabularies are sometimes called **ontologies** (more on that later)

Concrete syntaxes: RDF/XML

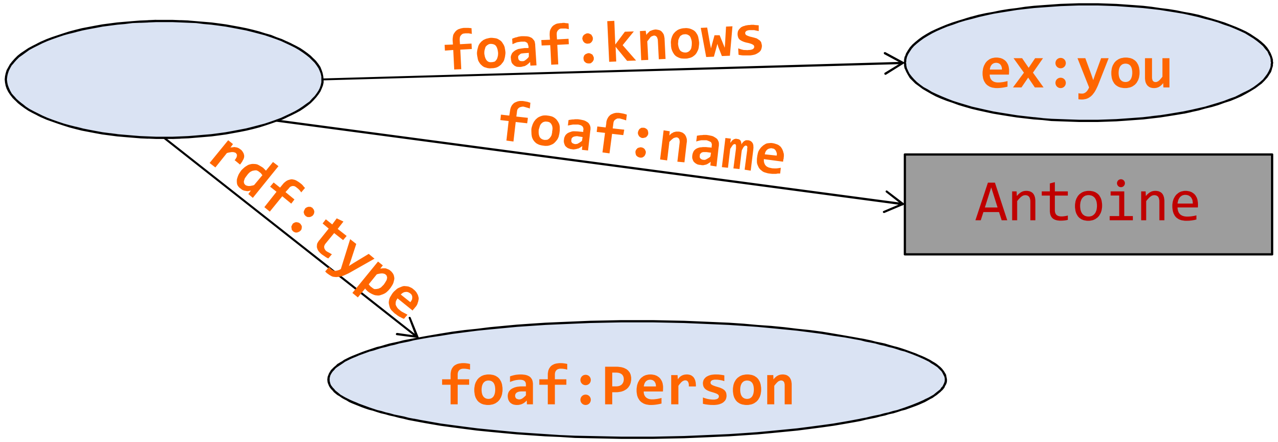
The graph:



```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF>
<rdf:RDF xmlns:ex="http://ex.org/#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/">
<foaf:Person>
   <foaf:knows rdf:resource="http://ex.com/#you"/>
   <foaf:name>Antoine</foaf:name>
</foaf:Person>
</rdf:RDF>
```

Concrete syntaxes: N-Triples

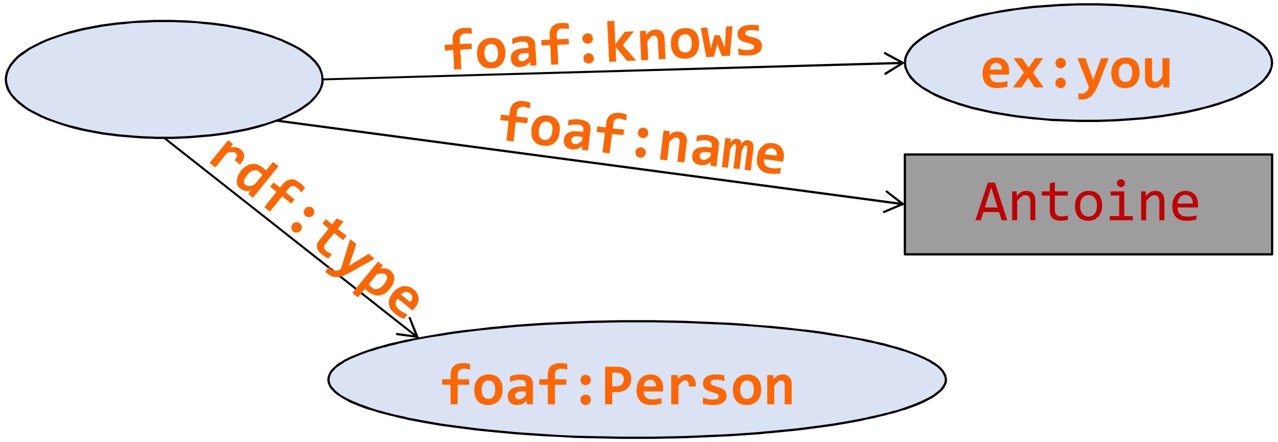
The graph:



```
_:genId384902443 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://http://xmlns.com/foaf/0.1/Person> .
_:genId384902443 <http://xmlns.com/foaf/0.1/knows> <http://ex.org/#you> .
_:genId384902443 <http://xmlns.com/foaf/0.1/name> "Antoine" .
```

Concrete syntaxes: JSON-LD

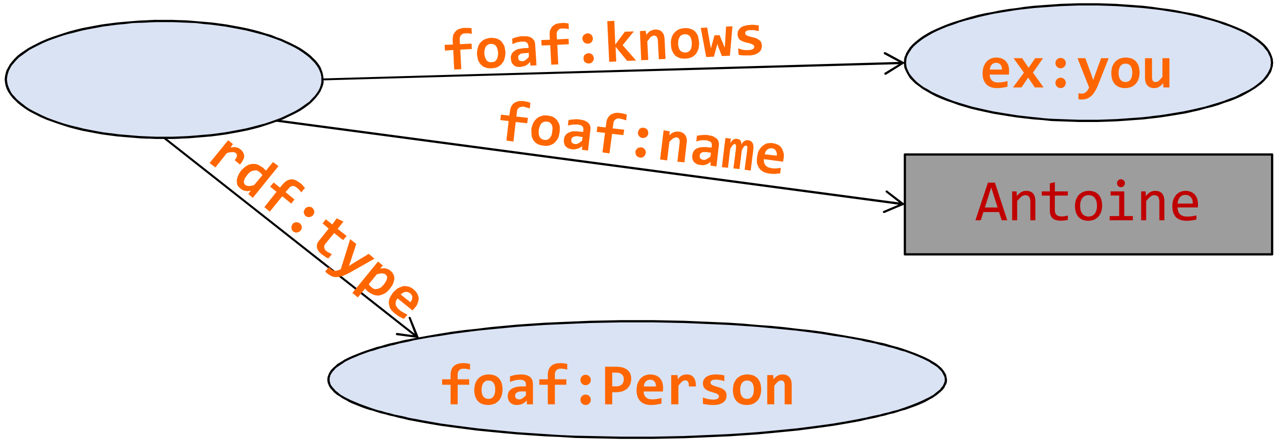
The graph:



```
{
  "@id": "_:bn0",
  "@type": "http://xmlns.com/foaf/0.1/Person",
  "http://xmlns.com/foaf/0.1/knows": [
    { "@id": "http://ex.org/#you" },
    { "@id": "http://ex.org/#him" },
    { "@id": "http://ex.org/#her" }
  ],
  "http://xmlns.com/foaf/0.1/name": "Antoine"
}
```

Concrete syntaxes: Turtle

The graph:



```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://ex.org/#> .

[]  a  foaf:Person;
   foaf:knows  ex:you, ex:him, ex:her;
   foaf:name  "Antoine" .
```

Format specifications

- *RDF 1.1 XML Syntax* – W3C Recommendation 25 February 2014

  https://www.w3.org/TR/rdf-syntax-grammar/

- *RDF 1.1 N-Triples - A line-based syntax for an RDF graph* – W3C Recommendation 25 February 2014

  https://www.w3.org/TR/n-triples/

- *JSON-LD 1.0 - A JSON-based Serialization for Linked Data* – W3C Recommendation 16 January 2014

  https://www.w3.org/TR/json-ld/

- RDF 1.1 - Turtle Terse RDF Triple Language – W3C Recommendation 25 February 2014

  https://www.w3.org/TR/turtle/

Publishing RDF on the Web

- Use case: *I want to publish my personal profile in RDF, with my name, affiliation, interests, education, professional relationships, etc.*
- Simple conceptual model but…
  - what IRI should I use (for myself, my company, etc)?
  - what properties?
  - where do I put the data?
  - how do I make the data easily usable?
  - …

See also: *Best Practices for Publishing Linked Data* – W3C Note 9 January 2014

https://www.w3.org/TR/ld-bp/

Linked Data principles

1. Use URIs as names for things
2. Use HTTP URI so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs. so that they can discover more things.

See: *Linked Data*. Tim Berners-Lee's design issues. July 2006 (revised June 2009)

https://www.w3.org/DesignIssues/LinkedData.html

Linked Data principles

1. Use ~~URIs~~ IRIs as names for things
2. Use HTTP ~~URI~~ IRIs so that people can look up those names
3. When someone looks up a ~~URI~~ IRIs, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other ~~URIs~~ IRIs. so that they can discover more things.

Dereferenceing

- **Dereferenceing**: operation that consists in using an IRI as a URL to get whatever document you can access using that URL
- Corresponds to issueing a GET method in HTTP, with the URL stripped of any fragment identifier
- An IRI is **dereferenceable** if it can be used in a HTTP GET request to access a document

Dereferenceing example

`http://danbri.org/foaf#danbri`

→ get rid of the fragment `#danbri`

`http://danbri.org/foaf`

→ issue a GET request:

```
GET /foaf HTTP/1.1
Host: danbri.org
```

→ server replies:

```
HTTP/1.1 200 OK
Date:…
Content-type: application/rdf+xml
…[other stuff]
```

What do HTTP URIs identify?

Rule of thumb:

*if a URL **locates a document** then the URL must **identify the document***

- How do we identify things that are not documents (physical objects, people, ideas, etc.)?
  - Non HTTP URIs? → breaks rule n°2 of Linked Data
  - HTTP URIs that do not locate documents (e.g., gives 404) → breaks rule n°3 of Linked Data

Technical architecture group advice

- If the server returns 200 OK to an IRI look up, then the IRI must denote an **information resource** (≈ a Web document)
- Otherwise, the IRI may denote **anything**
- *Advice*: to identify non-information resources, use either « hash IRIs » or [303-redirected] « slash IRIs »

Warning: controversial decision of the TAG, discussions on this issue have been occasionnally showing up on mailing lists since 2002!

Slash IRIs (1)

- A slash IRI is an IRI with a '/' followed by <u>a local name</u>:

  http://dbpedia.org/resource/**Semantic_Web**

→ issue a GET request:

```
GET /resource/Semantic_Web HTTP/1.1
Host: dbpedia.org
Accept: text/html
```

→ server replies:

```
HTTP/1.1 303 See Other
Location: http://dbpedia.org/page/Semantic_Web
```

→ issue a new GET request:

```
GET /page/Semantic_Web HTTP/1.1
Host: dbpedia.org
Accept: text/html
```

→ server replies:

```
HTTP/1.1 200 OK
```

Slash IRIs (2)

- issue a GET request:

```
GET /resource/Semantic_Web HTTP/1.1
Host: dbpedia.org
Accept: application/rdf+xml
```

- server replies:

```
HTTP/1.1 303 See Other
Location: http://dbpedia.org/data/Semantic_Web
```

- issue a new GET request:

```
GET /data/Semantic_Web HTTP/1.1
Host: dbpedia.org
Accept: application/rdf+xml
```

- server replies:

```
HTTP/1.1 200 OK
```

Hash IRIs

- A hash IRI is an IRI with a fragment identifier:

  `http://danbri.org/foaf#danbri`

- HTTP GET always removes fragment, so a hash IRI cannot be used to return 200 OK.

  → so it can be used for non-information resources

Advantages of hash VS slash:

http://www.w3.org/wiki/HashVsSlash

See also: *Cool URIs for the Semantic Web* – W3C Interest Group Note 3 December 2008

https://www.w3.org/TR/cooluris/

Means of publishing RDF

- Put RDF files online (in RDF/XML, Turtle, etc)
- Publish RDF along with web pages (RDFa)
  - Some CMS generate RDFa automatically (e.g., Drupal 7)
  - You'll see more about RDFa later
- Generate RDF from other existing formats
  - Triplifiers: http://www.w3.org/wiki/ConverterToRdf
  - Mapping languages:
    - For relational DBs: W3C R2RML and Direct Mapping
    - For other formats: XSLT, RM, SPARQL Generate
- Keep RDF inside database, but provide access via queries (SPARQL endpoints)

Existing online RDF datasets

- The Linked Open Data Cloud:
  - http://lod-cloud.net/
- List of SPARQL endpoints and availability
  - http://sparqles.ai.wu.ac.at/

Defining vocabularies

- There are IRIs that identify generic things:
  - Types / Classes
  - Properties
- These are likely to be useful in many applications
  - Reuse existing terms (Linked Data principle #4)
  - How to find the existing terms?
  - How to define new terms that will be used by many?

RDF Schema (RDFS) (1)

- A basic vocabulary for defining vocabularies
  - **rdf:type** (relates an instance to one of its classes)

    ```
    ex:me  rdf:type  foaf:Person .
    ```

  - **rdf:Property** (the class of all properties)

    ```
    foaf:name  rdf:type  rdf:Property .
    ```

  - **rdfs:Class** (the class of all classes)

    ```
    foaf:Person  rdf:type  rdfs:Class .
    ```

  - **rdfs:Resource** (the class of everything)

    ```
    rdfs:Resource  rdf:type  rdfs:Resource .
    ```

  - **rdfs:Datatype** (the class of data types)

    ```
    xsd:integer  rdf:type  rdfs:Datatype .
    ```

RDF Schema (RDFS) (2)

- `rdfs:subClassOf` (relates a class to one of its super classes)

  ```
  foaf:Person  rdfs:subClassOf  foaf:Agent.
  ```

- `rdfs:subPropertyOf` (relates a property to one of its super properties)

  ```
  foaf:skypeID  rdfs:subPropertyOf  foaf:nick .
  ```

- `rdfs:domain` (relates a property to a class of things it is about)

  ```
  foaf:firstName  rdfs:domain  foaf:Person .
  ```

- `rdfs:range` (relates a property to a class of things it relates to)

  ```
  foaf:homepage  rdfs:range  foaf:Document .
  ```

And more

rdf:type

- Paul is a person

```
ex:paul  rdf:type  ex:Person
```

- Product number 87876R5 is a laptop

```
product:87876R5  rdf:type  ex:Laptop
```

- X was employed by Y between 2010 and 2013

```
a:e2010-2013  rdf:type  ex:Employment
```

rdf:Property

- People know other people

```
foaf:knows  ex:type  rdf:Property
```

- Products have prices

```
ex:price  rdf:type  rdf:Property
```

- People are employed by companies for a time

```
ex:employment  rdf:type  rdf:Property
```

Instance of properties

- Paul knows Rémi

```
ex:paul  foaf:knows  ex:remi
```

- Laptop X in store Y costs €1200

```
ex:laptopXY  ex:price  "1200"^^xsd:decimal
```

- Paul was employed by Google between 2008 and 2013

```
ex:paul  ex:employment  _:e255 .
_:e255  ex:by  g:Google .
_:e255  ex:starting  "2008"^^xsd:gYear   …
```

rdfs:Class

- People, products, employment, etc

```
ex:Person   rdf:type   rdfs:Class
ex:Product   rdf:type   rdfs:Class
ex:Employment   rdf:type   rdfs:Class
```

- rdfs:Resource, rdfs:Datatype:
  - Not particularly needed in modelling, everything is a rdfs:Resource, datatypes are pre-defined in general

rdfs:subClassOf

- People are agents

```
ex:Person  rdfs:subClassOf  ex:Agent
```

- Laptops are products

```
ex:Laptop  rdfs:subClassOf  ex:Product
```

- Employments are events

```
ex:Employment  rdfs:subClassOf  ex:Event .
```

rdfs:subPropertyOf

- Being friend is knowing

```
ex:friendOf   rdfs:subPropertyOf   ex:knows
```

- Being inside is being near

```
ex:isInside   rdfs:subPropertyOf   ex:basedNear
```

- …

rdfs:domain and rdfs:range

- Only people are employed

```
ex:employment  rdfs:domain  ex:Person
```

- Something is based near a location

```
ex:basedNear  rdfs:range  ex:Location
```

- Events starts at a date and time

```
ex:startsAt  rdfs:range  xsd:dateTime
```

…

Other useful things

- `rdfs:label` – a human readable "name" for a thing

```
_:e255  rdfs:label  "Paul’s employment 2012"@en
```

- `rdfs:comment` – a description or commentary for a thing

```
ex:laptopXY  rdfs:comment "Laptop in my office,
with 8 GB RAM, 2.9 GHz Quad Core"@en
```

Other useful things

- rdf:List, rdf:first, rdf:rest, rdf:nil

```
isbn:1617290394  ex:author _:authorList .
_:authorList  rdf:type  rdf:List .
_:authorList  rdf:first  ex:dwood .
_:authorList  rdf:rest  _:restList .
_:restList  rdf:first  ex:mzaidman .
...
_:endList  rdf:first  ex:mhausenblas .
_:endList  rdf:rest  rdf:nil .
```

- In Turtle:

```
isbn:1617290394  ex:author
       (ex:dwood ex:mzaidman ex:lruth ex:mhausenblas) .
```

Inferences with RDFS semantics (1)

- **Given:**
```
ex:C  rdfs:subClassOf  ex:D .
ex:D  rdfs:subClassOf  ex:E .
```
- It can be proved that:
```
ex:C  rdfs:subClassOf  ex:E .
```
- **Given:**
```
ex:p  rdfs:subPropertyOf  ex:q .
ex:q  rdfs:subPropertyOf  ex:r .
```
- It can be proved that:
```
ex:p  rdfs:subPropertyOf  ex:r .
```
- **Given:**
```
ex:C  rdfs:subClassOf  ex:D .
ex:x  rdf:type  ex:C .
```
- It can be proved that:
```
ex:x  rdf:type  ex:D .
```

Inferences with RDFS semantics (2)

- **Given:** `ex:x  ex:p  ex:y .`
  `ex:p  rdfs:subPropertyOf  ex:q .`
- It can be proved that:

  `ex:x  ex:q  ex:y .`
- **Given:** `ex:p  rdfs:domain  ex:C .`
  `ex:x  ex:p  ex:y .`
- It can be proved that:

  `ex:x  rdf:type  ex:C .`
- **Given:** `ex:q  rdfs:range  ex:D .`
  `ex:x  ex:q  ex:y .`
- It can be proved that:

  `ex:x  rdf:type  ex:D .`

And more

Finding existing vocabularies

- Reuse well known vocabularies (Dublin Core, FOAF, SIOC, Good Relations, SKOS, voiD, etc.)
- Try an ontology / vocabulary search engine or repository:
  - *Search engines*: FalconS, SWSE, Sindice, OU's Watson, Swoogle, vocab.cc
  - *Repositories*: Linked Open Vocabulary, ScheWeb, Schemapedia, Cupboard, Knoodl, Ontology Design Patterns, prefix.cc, DERI vocabularies, OWL Seek, SchemaCache
- Ask mailing lists, forums (semantic-web@w3.org, public-lod@w3.org, answers.semanticweb.com)

Build your own vocabulary

- Editors:
  - Protégé, WebProtégé, NeOn TK, SWOOP, Neologism, TopBraid Composer, Vitro, Knoodl, Ontofly, Altova OWL editor, PoolParty, IBM integrated development TK, Anzo for Excel, Euler GUI
- Learn, evaluate:
  - Protégé tutorial, …bits and pieces here and there
  - RDF validator, OWL validator, Linked Data validator, Data Hub LOD Validator
  - Best practices for publishing RDF vocabularies
- Link to other ontologies

more at http://www.w3.org/wiki/Ontology_Dowsing

Use case 1: describing the world

- Describe in RDF the following situation:

"Marco is a student at Université Jean Monnet, studying in the Master 2 programme Web Intelligence. There, he follows the course Semantic Web, taught by Antoine Zimmermann. Marco is italian but lives in Saint-Étienne, place Jean Jaurès, with his friends and flat mates Enrico and José. Marco is interested in Web technologies, theater and sci-fi literature. Enrico is interested in marijuana, reggae and is an activist for world-wide peace. Antoine Zimmermann is associate professor at École des mines, with colleagues Olivier Boissier, Gauthier Picard, etc. École des mines is a higher education establishment depending on the Ministry of industry."

Use case 2: using existing data

- Translate the following tables to RDF:

TeamID	Name	Country	Coach
FRA	XV de France	France	Laporte
NZL	All Blacks	New Zealand	Henry
ENG	XV of the Rose	England	Ashton
…	…	…	…

PlayerID	Name	TeamID	Position
1	Vincent Clerc	FRA	wing
2	Lionel Beauxis	FRA	flyhalf
3	Joe Rokocoko	NZL	wing
…	…	…	…

Use case 3: UML to ontology

Usually, these translations are appropriate:

- UML classes → RDF classes
- UML attribute → RDF properties with literals as range
- UML links → RDF properties
- UML generalization → rdfs:subClassOf
- Visibility and methods are normally not represented in RDF (it’s not a programming language)
- Cardinalities cannot be represented with RDFS, but can in OWL (cf. future courses), but be careful!
- **Note:** *in RDF, properties are not attached to classes. They are first class citizens.*

RDF files and RDF APIs

- RDF files (RDF/XML, Turtle, N-triples, etc) can be read into memory with RDF APIs
- The in-memory model of an RDF graph can be manipulated with API methods
  - Java APIs: Apache Jena (part of documentation in French), Sesame
  - .NET C#: dotNetRdf
  - Python: pyRDF
  - PHP: rdflib (not maintained any longer)
  - Javascript
  - Many more

Storing and managing RDF

- RDF databases are also called **triple stores**
- Some triple stores scale up to *trillions of RDF triples*, given enough hardware:
  - AllegroGraph, OWLIM, Virtuoso, …
- Small capacity triple stores (good for quick development of simple Web apps):
  - Jena Fuzeki, Sesame, and others

Managing multiple RDF graphs

- RDF 2004 philosophy: every triples express something about the world. More graphs mean more knowledge. If we find more graphs, we just add triples.
  - → Putting 2 graphs together to make a single equivalent RDF graph requires more than set union
- **RDF graph merge**: the merge of 2 RDF graphs G1 and G2 is an RDF graph G such that if G is ***true*** then G1 and G2 are true, and if G1 and G2 are both ***true*** then G is also true
  - In practice, it requires making the blank nodes of G1 and G2 disjoint, before taking the union

RDF datasets

- In many situations, graph merge is not ideal:
  - RDF graphs disagree (keep track of who says what)
  - RDF graphs evolve (keep track of temporal evolution)
  - RDF graphs are imprecise (keep track of fuzziness)
  - RDF graphs are sometimes private (keep track of access control)
- RDF 1.1 defines a new data structure (RDF dataset)
- **RDF dataset**: a structure comprising:
  - An RDF graph called the **default graph**
  - Zero or more "**named graph**", which are pairs (IRI, RDF graph), and the IRI is the "**name**" of the named graph