Paper TT06

Linked Data

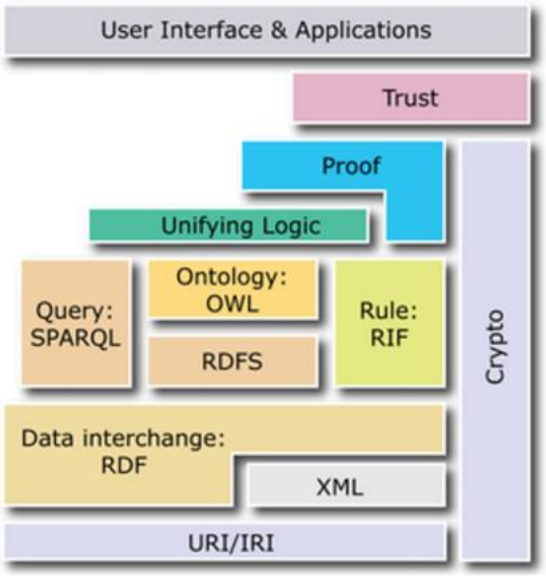
Nicolas Dupuis, d-wise, Basel, Switzerland

ABSTRACT

Linked Data is on the radar for some time now. My intention is to provide an overview of the technology stack: URI, RDF, SPARQL, RDFS and OWL. You will understand semantics data, how to query it, how to build ontologies and how to infer knowledge with them. The industry has great plans for Linked Data and Graph Databases. It can appear challenging to implement but you will hopefully realize its benefits.

INTRODUCTION

Linked Data is a method of publishing structured data and is a recommendation from the World Wide Web Consortium (W3C). Tim Berners-Lee, the inventor of the World Wide Web (WWW), is the driving force for Linked Data. One of the goal is to be able to publish data that can be interlinked and become more useful through semantic queries. Linked Data is supported by the web infrastructure and have a technology stack. We will have a deeper look at some of the elements in this stack.



(Source)

RECTANGULAR DATA: THE SHORTCOMINGS

Before we talk about Linked Data, let's see why need it so badly. We all know rectangular data (rows and columns), the current paradigm in our industry. We are familiar with it but they come with lots of problems.

Say that we want to merge these two sets of rectangular data we found somewhere:

Name	Spouse	Secrete_Identity
Clark Kent	Lois	Superman
Peter Parker	Mary-Jane	Spyderman

Name	Activity	DOB
L. Lane	Journalist	1937
MJ. Watson	Model	1965
MJ. Watson	Actress	1865
C. Kent	Journalist	1938
P. Parker	Photographer	1963

We can see already the problems:

- What are the key variables? Where is the documentation?
- There are many ambiguities in the column names:
  - What is "Spouse"? Does it mean what we think it means? It's just a name, nothing prevents you to misname a column.
  - What is "DOB"? Can we assume it means date of birth?
  - "Secrete_identity", "Activity"? For sure, there are many ways to name these concepts. According to Murphy's Law they will all be used, making our life harder when it comes to aggregating data.
- There are many ambiguities in the data itself:
  - Is "C. Kent" born in 1938 the same person as "Clark Kent" married with 'Lois'? How can we be sure? Merging by name seems reasonable but what about homonyms?
  - What is a "Model"? There are many meanings for that. Is MJ a statistical model?
- The data is not clean and needs pre-processing/cleaning:
  - Inconsistencies: "Clark Kent" / "C. Kent"
  - Typos: "Spyderman"
- There is redundancy: per design there are two lines for MJ. Watson (she has two jobs!), therefore two DOBs. As Peter's uncle would say "With great redundancy comes great discrepancy"
- Any inference we want to do is manual work, e.g. when was Superman born? Who is Lois' husband?

Wow… Most of these problems can be solved with a smart programmer and some coffee but it's hugely time consuming. Is it sustainable with large datasets? One problem remains: any inference must be imagined by humans. You need understanding of the data to imagine extra things that you could get from it. For example, you need a human brain to see that spouse is something symmetric (A has spouse B implies that B has spouse A) and therefore there's hidden data.

Linked Data address all these shortcomings. Let's see how.

SEMANTIC DATA

Semantics is the linguistic study of meaning, i.e. the relationship between a word and what it stands for. It's an old journey that started thousands of years ago. An ultimate goal of the W3C is to create a semantic web, and not just a document web. In a semantic web, computers could make use and analyze machine-readable data. Let's see how we can have it machine-readable. Usually in rectangular data, we can describe semantics in the metadata but it's often meant for humans.

DATA MODEL

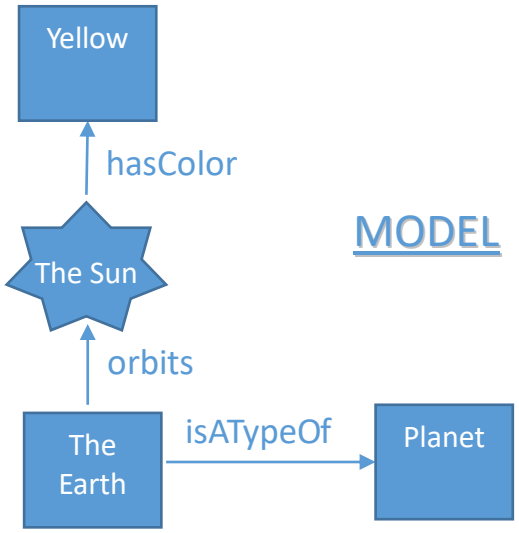
RESOURCE DATA FRAMEWORK

RDF (Resource Data Framework) is the W3C standard data model to make statements about things, to express semantic, so we can model knowledge. These statements are known as triples:

Subject	**Predicate** (property name)	**Object** (property value)
The Sun	hasColor	Yellow
The Earth	isATypeOf	Planet
The Earth	orbits	The Sun

RDF is a data model with only these three specific columns and that will never change. Basically everything could be said with these 3 parts sentences.

We can represent that knowledge graphically, with predicates linking nodes (a resource, i.e. a thing). These predicates are resources too and be subject or object of other triples.



RDF SERIALIZATION

RDF is an abstract model. To store the information in a file, we need some syntax. That's what the serialization formats provide. There are several formats available, I chose here to use the famous Turtle format (Terse RDF triple language), which is published by the W3C. Here's an example:

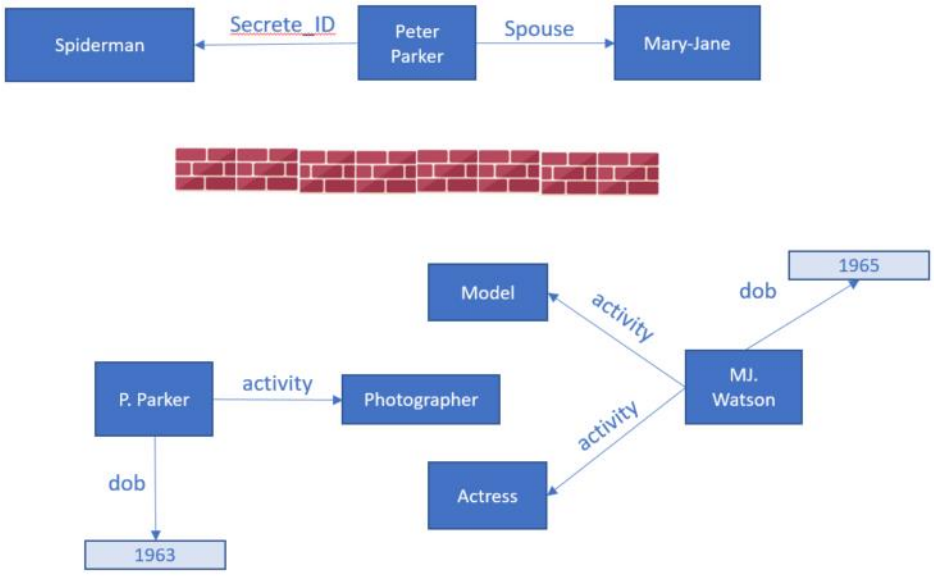
	in Turtle format:
A statement	`green-goblin enemyOf spiderman .`

A list of predicates	`green-goblin enemyOf spiderman ;`
	`type Person ;`
	`name "Green Goblin" .`

A list of objects	`spiderman name "Spiderman"@en ,` `"L'homme araignée"@fr .`

LET'S LINK OUR DATA: #1 ATTEMPT

Let's convert our comics datasets into RDF and see if it helps. We have fixed the typos, created predicates etc. We have two sets of RDF triples.



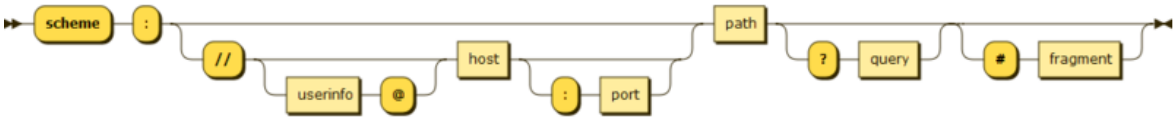

```
"Peter Parker" Spouse     "Mary-Jane" ;
               secrete_ID "Spiderman" .
```

```
"P. Parker"    activity   "Photographer"  ;
               dob        "1963"          .
"MJ. Watson"   activity   "Model"         ,
                          "Actress"       ;
               dob        "1965"          .
```

Can we merge this easily? The data is still ambiguous, we cannot merge the data without manipulation. Metadata is still also ambiguous. Inferences are still manual. Conclusion: there's no magic behind RDF, it's just a data model. We need something more to easily link data. Let's see what.

UNIFORM RESOURCE IDENTIFIER

A URI is a unique string of characters that unambiguously identifies a particular resource (= a thing). To guarantee uniformity, all URIs must follow syntax rules.



(source)

The most common form of URI is the Uniform Resource Locator (URL), which everyone is familiar with. URLs unambiguously identify where to find a resource. In fact, all URLs are URIs. The Linked Data recommendations are:

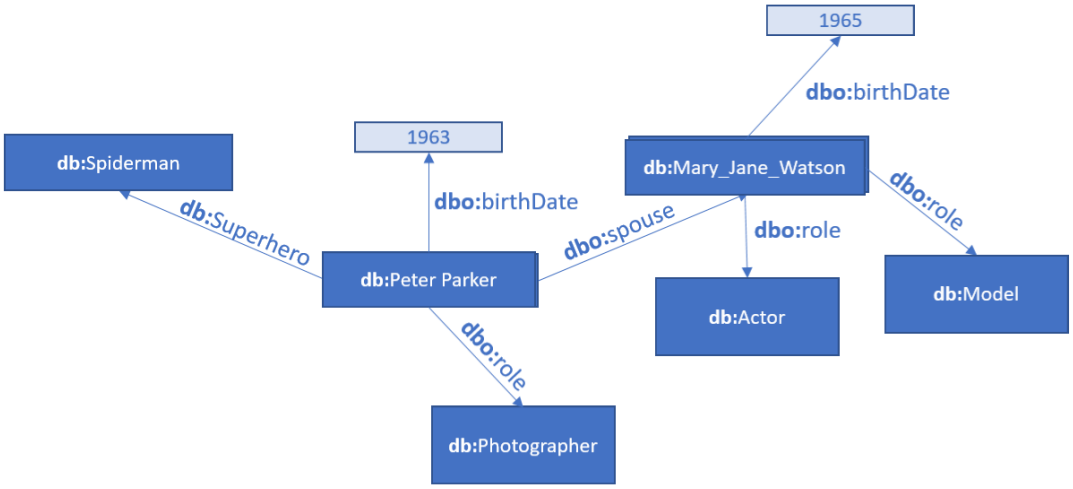
- Define resources with a URI
- URIs should be public URLs to ensure uniqueness and re-usability
- URLs should have browsable content, providing semantics, i.e. what it is you are reading

Don't write "Spyderman", for sure, but don't even say "Spiderman", say instead http://dbpedia.org/page/Spider-Man, which is a proper URI. That way we all talk about the same thing and that thing is clearly defined, no ambiguity on the semantic. Furthermore, the browsable information can be re-used instead of re-inventing.

Dbpedia is a project aiming to extract structured content from Wikipedia. It allows users to semantically query relationships and properties of Wikipedia resources. Tim Berners-Lee described DBpedia as one of the most famous parts of the decentralized Linked Data effort.

LET'S LINK OUR DATA: #2 ATTEMPT

Let's rewrite our triples, this time using URI taken from the web. Subjects, predicates and often objects can be defined this way. No ambiguity. Cherry on the pie: there's no need to merge anything as it's already linked. We just need to join the data sources.



Great but is that all? Is it worth the effort? What else can we achieve using this Linked Data? For example, the data says that MJ. is Peter's spouse but it doesn't say that Peter is MJ.'s spouse. If you query the data for MJ.'s spouse, you will get nothing. Humans understand that spouse is a symmetrical concept (we've been taught) and we don't need the redundant symmetric information. Computers also need to be taught, we'll see how.

SPARQL

Imagine a huge amount of Linked Data, which can be named 'graph database'. This could get messy to visualize and work with. Retrieving and manipulating information is key for a data model to be usable. Rectangular data has SQL, Linked Data has SPARQL. It's a query language published by the W3C to manipulate RDF data, with the goal to look a bit like SQL. Here's a list of simple SPARQL queries:

SPARQL query | **Result**

```
PREFIX dbo: <http://dbpedia.org/page/>
SELECT ?subject ?job
WHERE {?subject dbo:role ?job .}
```

subject	job
Clark_Kent	Journalist
Mary_Jane_Watson	Model
etc...	

People and their jobs

```
SELECT ?subject
WHERE {?subject dbo:role db:Actor
       ?subject dbo:role db:Model .}
```

subject
Mary_Jane_Watson

People who are Model and Actor

```
SELECT ?subject ?spouse
WHERE {?subject dbo:role db:Journalist .
       OPTIONAL {?subject dbo:spouse ?spouse .}}
```

subject	spouse
Clark_Kent	Lois_Lane
Lois_Lane	

Journalists and their marital status (if any)

```
SELECT ?Journalists ?dob
WHERE {?Journalists dbo:role db:Journalist .
       ?Journalists dbo:birthDate ?dob .
       FILTER (?dob > "1937") }
```

Journalists	dob
Clark_Kent	1938

Journalists born after 1937

```
CONSTRUCT {?object dbo:spouse ?subject}
WHERE {?subject dbo:spouse ?object .}
```

Subject	Predicate	Object
Lois_Lane	dbo:spouse	Clark_Kent
Mary_Jane_Watson	dbo:spouse	Peter_Parker

Spouse's spouses.

```
SELECT ?s
WHERE {?s dbo:birthDate ?dob.}
ORDER BY ?dob
LIMIT 1
```

s
Lois_Lane

Oldest

```
SELECT ?s
WHERE {?s dbo:role db:Journalist .
       FILTER NOT EXISTS {?s dbo:spouse ?o } }
```

s
Lois_Lane

Journalist who are single

```
SELECT (COUNT (?subject) as ?howMany)
WHERE {?subject dbo:role db:Journalist . }
```

howMany
2

How many journalists

```
SELECT ?s (COUNT (?job) as ?jobs)
WHERE {?s dbo:role ?job . }
GROUP BY ?s
HAVING (?jobs > 1)
```

s	jobs
Mary_Jane_Watson	2
Lois_Lane	1

How many jobs per person

SPARQL is a great and easy language for RDF, you can do all the classic things with it.

WEB ONTOLOGIES

Ontology is the study of being, of what there is. It's another very old journey as the ancient Greek philosophers were already busy at it.

Ontologies organize concepts, categories, properties, relationships and constraints. The web ontologies are useful for inference (can we discover something from our data that wasn't explicit?) or federating data (combining data from different places/author). The WWW philosophy is "*Anyone can say Anything about Anything*" (AAA). It has proved to be a very successful approach but the downside is that it can easily (and had) become quite a mess. Ontologies provide structure to describe your data and allow federation and discovery.

RDFS (RDF Schema Language) and OWL (Web Ontology Language) are languages to build web ontologies. RDFS provides modeling tools for knowledge description and discovery. OWL is the W3C published language to author complex ontologies. It builds on RDFS and provides more subtle constructs and finer-grained modeling. The beauty is that they can be expressed in RDF, in other word the metadata and the data can live together in the same file using the same format. No need for a fancy metadata repository.

Most of the ontology constructs have formal semantics, i.e. they can be specified with a well-defined SPARQL query for example. They are best used for inference and not just description, allowing to make best use of the AAA. Practically speaking inferring means that an OWL-aware software (a.k.a. semantic reasoner) is able to create new triples, following the rule defined in the formal semantics. Here is a few examples of such constructs:

Constructs	Formal semantics	In plain English

rdfs:subClassOf	`CONSTRUCT {?s rdf:type ?c2}`
	`SELECT {?c1 rdfs:subClassOf ?c2 .`
	`?s rdf:type ?c1 }`
		If a subject is a type of X and X is a sub-category of Y, then we infer that the subject is also a type of Y.

rdfs:domain	`CONSTRUCT {?s rdf:type ?domain}`
	`WHERE {?prop rdfs:domain ?domain .`
	`?s ?prop ?o .}`
		If a subject is using a property and that property has a domain X, then we infer that the subject is a type of X

rdfs:range	`CONSTRUCT {?o rdf:type ?range}`
	`WHERE {?prop rdfs:range ?range .`
	`?s ?prop ?o .}`
		If a subject is using a property and that property has a range X, then we infer that the object of that subject is a type of X

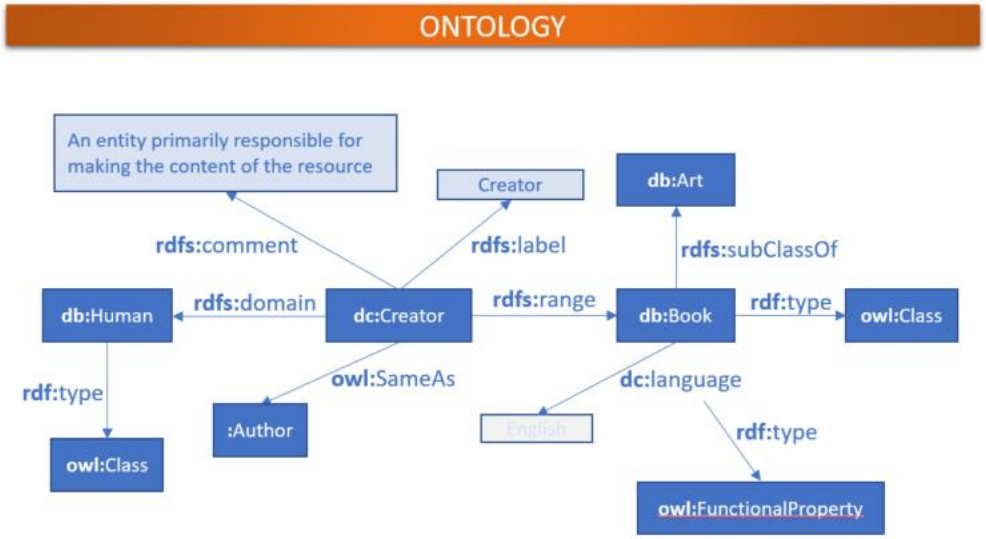
owl:SameAs	`CONSTRUCT {?s2 ?p ?o}`
	`SELECT {?s owl:sameAs ?s2 .`
	`?s ?p ?o .}`
	(and same for p and o)
		If A is the same as B then anything that is said about A can be inferred for B (whether A/B are subjects, predicates or objects)

owl:FunctionalProperty	`CONSTRUCT {?object owl:sameAs ?object2}`
	`WHERE {?prop rdf:type owl:FunctionalProperty .`
	`?s ?prop ?object .`
	`?s ?prop ?object2 .}`
		With a property defined as 'functional', we expect only one object per subject for that property. If there are more than one, then we infer that these objects are the same. It's not a constraint, it's not a data check (remember, AAA), it's a way to infer sameness!

There are many more… We'll see concrete examples now, hopefully it will get clearer.

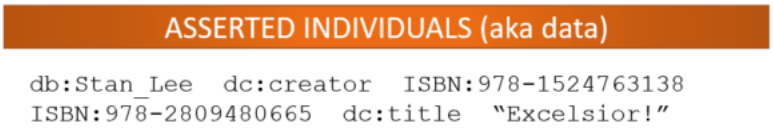
EXAMPLE OF INFERENCE

First let's create an ontology, a framework/metadata for our data, with a few RDF triples using RDFS and OWL constructs.



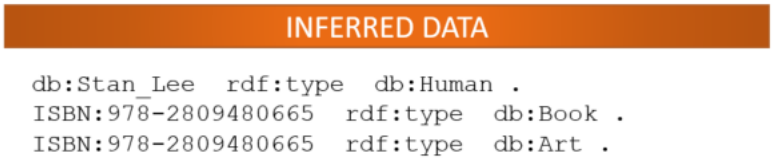
In English: we defined creators, they are of type human (a class of things) and they create books (another class which is a type of art). We stated that authors are the same as creator. We gave a label and comment for creator (two properties that do not have formal semantics, they're just here to help).

We created boxes, now let's stuff them with actual data:



In English: Stan Lee wrote a book called "Excelsior!". These resources are non-ambiguous and can be seamlessly combined in a larger triplestore (a graph database that holds triples) as long as the data consistently uses the same URI/prefixes…

The semantic reasoner (i.e. an OWL-aware application) is able to **create** new triples without human intervention:

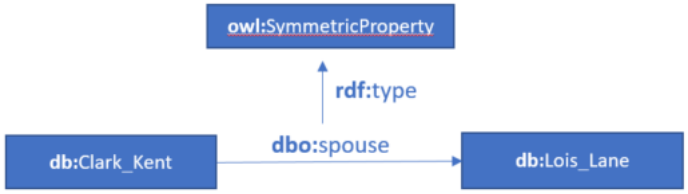


We said that `db:Stan_Lee` was the creator of something. We also specified the domain of the `dc:creator` predicate (i.e. the expected subjects): `db.human`. We can therefore infer that Stan Lee was human (forget the rumors). We also specified the range of the `dc:creator` predicate (i.e. the expected objects): `db.book`. So we infer that the object of `dc:creator` is a type of `db:book`. And since we said that books are a type of art, then Stan Lee's book is a type of art. These statements are not in the data, yet they can be automatically created, stored permanently and searched just like the asserted triples.

The dc prefix is for 'Dublin Core', a famous ontology providing a small set of vocabulary terms that can be used to describe digital resources (video, images, web pages, etc.).

FUN CHALLENGE #1: SYMMETRY AND INFERENCE

Say that we assert this data, with two triples:



The semantic reasoner knows the formal semantic for that OWL construct:

```
CONSTRUCT {?o ?prop ?s}
    WHERE {?prop rdf:type owl:SymmetricProperty .
           ?s ?prop ?o .}
```

In English: for every triple where a subject/object are using a predicate defined as symmetric, we'll get a new triple inverting subject/object for the same predicate. So, if we run the following SPARQL query with the reasoner, we will get two answers: Clark and Lois, and not just Clark as asserted.

```
SELECT ?s
WHERE {?s dbo:spouse ?o .}
```

With a simple triple `dbo:spouse rdf:type owl:SymmetricProperty` added to our triplestore, we have generated more information from our data. Small effort, maybe big reward. Maybe we should have a look at the other predicates we have and see if we could describe them with RDFS or OWL properties. We might get lucky, augmenting our model could teach us something unexpected.

FUN CHALLENGE #2: DATA FEDERATION

Federating data is about aggregating multiple sources of information (a.k.a. endpoints) and making sense of them. Let's look at these two sources of triples coming from the Daily Planet triplestore. Lois and Jimmy wrote this:

<http://www.dailyplanet.com/Lois/sparql>

```
:Clark :email "clarkkent@dailyplanet.com"
:Clark :email "Ibelieve@IcanFly.com"
:Clark foaf:name "Clark Kent"
:Lois :likes :Clark
```

<http://www.dailyplanet.com/Jimmy/sparql>

```
:Superman :isLocated "Metropolis"
:Superman :emailAddress "Ibelieve@IcanFly.com"
```

Perry, their boss, wants to know everything about Superman. He just learnt SPARQL and queries the Daily Planet's triple store:

```
SELECT ?p ?o
WHERE {:Superman ?p ?o.}
```

This query will retrieve Superman's email and in which city he lives, nothing really exciting.

While exploring the data Perry noticed a lack of standardization. The `:email` and `:emailAdress` predicates seem to mean the same. He can address this with a simple triple saying that one is a sub-property of the other, gently federating these 2 concepts. Then came the brilliant idea: one person can have several emails but an email should only belong to one person. This is the inverse of the functional property we saw earlier, the goal remains to infer sameness among different resources (remember, AAA).

Behind the scene:

Constructs	Formal semantics
owl:subPropertyOf	CONSTRUCT {?s ?p2 ?o} WHERE {?p1 owl:subPropertyOf ?p2 . ?s ?p1 ?o . }
owl:InverseFunctionalProperty	CONSTRUCT {?subject owl:sameAs ?subject2} WHERE {?prop rdf:type owl:InverseFunctionalProperty . ?subject ?prop ?o . ?subject2 ?prop ?o .}

So, Perry added two triples Perry to the triple store:

<http://www.dailyplanet.com/Perry/sparql>

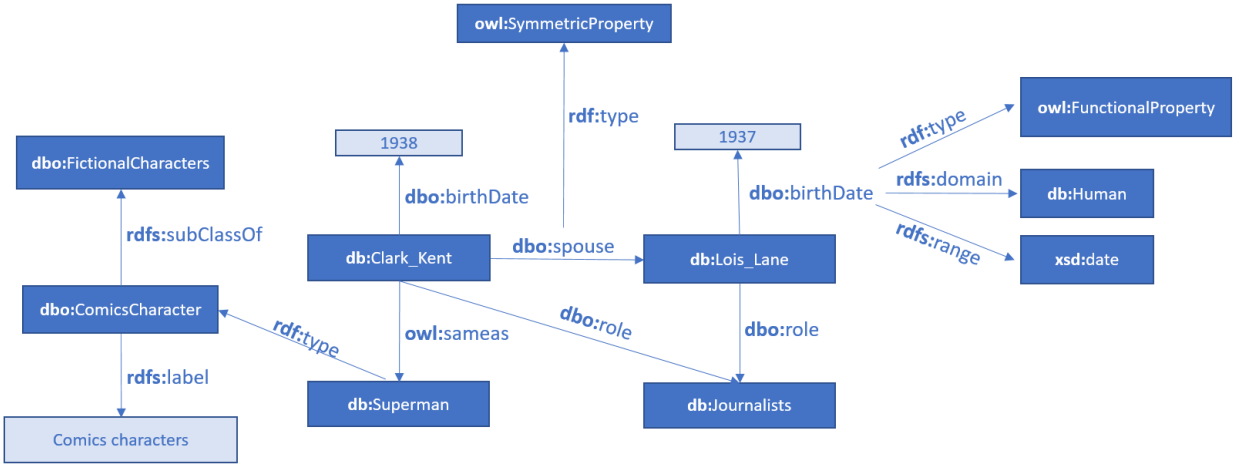
```
:emailAdress owl:subPropertyOf :email
```

```
:email rdf:type owl:InverseFunctionalProperty
```

Now if he runs the SPARQL query again, a miraculous triple appears `:Superman OWL:sameAs :Clark` ! That's because the predicate `:email` expects only one subject per object (the object being IBelieve@IcanFly.com). Since there're two (Clark and Superman), sameness is inferred for them. So, Superman is Clark Kent and all statements made for one is true for the other. Stop press, breaking news!

FUN CHALLENGE #3: PUSHING IT TOO FAR

Ontology constructs are powerful but can be misused. A simple triple used too liberally can have strange consequences. Let's have a look at this ontology:



Can you guess what could go wrong, i.e. what inference could we make that we know would be wrong?

There is sameness between Clark Kent and Superman, whether it's asserted or inferred. Whatever is said about Clark can be said about Superman. We asserted a date of birth for Clark, this is therefore inferred for Superman too. Finally we have set the domain for `dbo:BirthDate` to be Human, therefore we can infer Clark and Superman to be of type human. That's not quite right is it? We were too liberal defining the domain for `dbo:BirthDate`.

MORE?

There's more to the technology stack. There are also different ways to implement this technology. The Open Source players are a good start. Protege will help you build an ontology, create triples and infer. With Python and some libraries (RDFLib for example) you can store triples, run SPARQL query and infer. The Apache Jena offers free, open source tools including a triplestore. Proprietary software offers comprehensive and user friendly solutions for Graph Databases, e.g. Neo4j, TopBraid, Anzo.

CONCLUSION

We have seen simple examples that show the potential of Linked Data. I'm sure you can already think of concrete applications for your daily data, work related or not (I'm using Protégé for my mountaineering projects!). Our industry heavily relies on rectangular data, whether for clinical, operational and other sorts of data. Other industries have left this paradigm and make use of Linked Data. Why can't we? What's stopping us? Are we so happy with rectangular data? Relational databases have their strengths but in the era of data lakes, are they the answer? Clearly conducting a clinical trial and using Linked Data seems quite futuristic, though we see at PhUSE some efforts to go in that direction (thanks to Tim Williams and others). Whether it's baby steps or shooting for the moon, we could make great use of Linked Data and graph db in many different ways.

ACKNOWLEDGMENTS

Thanks to my colleagues in d-wise for their support and review, especially Ali Dootson.

RECOMMENDED READING

"Learning SPARQL" by Bob DuCharme, O'Reilly editions
"Semantic Web for the Working Ontologist" by Dean Allemang & James Hendler, MK editions

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at: ndupuis@protonmail.com