Representing data as resources in RDF and OWL

Pierre-Antoine Champin

LIRIS, Université Claude Bernard Lyon 1,
pchampin@liris.cnrs.fr

Abstract. This paper presents an RDF vocabulary for representing data values as resources. An intended application is the representation of relational databases in RDF, and reasoning with them using semantically rich dialects of RDF, such as OWL. After discussing the rationale for this work, we present the structure of the vocabulary, some implementation considerations and possible extensions, which open new research directions for integrating databases in the Semantic Web.

1 Introduction

RDF, the Resource Description Framework [1] has been recommended by the W3C as the language of choice for knowledge representation in the so called Semantic Web (SW). In RDF, objects of the domain are represented as interrelated *resources* and identified by Uniform Resources Identifiers (URI), while attribute values are represented by *literals* (a string, possibly characterized by a language or a datatype). The OWL language [2], a dialect of RDF, adds the formal semantics of very expressive description logics [3] and powerful inference mechanisms[1].

The initial motivation of this work was to represent in RDF/OWL data from legacy sources, e.g. relational databases. Our contribution is a standard RDF vocabulary for representing data values as resources. Although that is redundant with RDF literals, we argue in favor of resources over literals in section 2. In section 3, we give the formal specification of our vocabulary. We discuss a number of current and future developments to this work in section 4. Finally, we conclude.

2 Rationale

In this section we demonstrate some advantages of representing data values as resources rather than literals. Some arguments are concerning RDF as a whole, while some others are more specifically concerning OWL.

[1] Actually, OWL itself has three dialects (called species): Lite, DL and Full. Only the first two of them are description logics. OWL-Full, on the other hand has no decidable inference mechanism. In the following, mentions to OWL will only refer to its decidable species.

2.1 Making RDF more uniform

Any RDF description can be seen as a set of *triples*, composed by a *subject*, a *predicate* and an *object*. Only the object of a triple may be a literal, while the subject and predicate are necessarily resources[2]. Each triple is usually thought of as a labelled arc between the subject and the object, hence RDF descriptions are often referred to as RDF *graphs*.

From another point of view, an RDF description can be seen as a set of resources (subjects), which are described by a set of predicate-object pairs (more usually thought of as attribute-value pairs), where each object can either be another resource or a literal. From that point of view, we see that literals are somewhat "weaker" than resources, since they can not in turn be described (i.e. they can not be the subject of any triple, they are pits in the RDF graph). This "weakness" will be even more pronounced in OWL (see section 2.2).

Since resources in RDF are intended to represent virtually anything, they could have been targeted from the start to represent data values, and thus the irregularity in the language introduced by literals would have been eschewed. However, literals do have some good properties:

- The distinction between objects and atomic values is widely accepted: it holds in the most popular object-oriented programming languages, such as C++ or Java, as well as in modelling formalisms (e.g. Entity-Relationship).
- Literals fit nicely in the recommended XML syntax of RDF (any text node in the XML tree becomes a literal in the RDF graph). This eases the transition from any XML DTD or Schema into valid RDF [4].
- Data values come with a number of implicit properties and relations, (e.g. natural order, arithmetic operations, etc.) which could not reasonably been made explicit as an RDF graph, essentially because the domain of the datatype are usually huge or even infinite. So it seems appropriate to manage them differently from objects.

Let us note that the first two arguments are more "marketing" than technical: they aim at easing the acceptance of RDF. The third one is the only real technical argument against our approach, and we will discuss it in section 4.3. Anyway, our goal is not to ban literals from RDF, but to propose a alternative to them, which is not ideal either but has, amongst others, the advantage of uniformity. It is even possible for data resources and literals to coexist (see section 4.1).

2.2 Making OWL inferences more powerful

As we mentioned before, the difference between resources and literals is even more important in OWL than it is in plain RDF. OWL inference engines can reason about *classes* of resources: they can for example decide whether a class

[2] For the sake of simplicity, we will not mention the difference between URI nodes and blank nodes, which both represent resources, while Literals are completely different objects.

is included in another one (subsumption), whether a class can contain resources (satisfiability), whether a resource belongs to a given class (instance checking), or list the classes to which a given resource belongs (realization). They can not, on the other hand, do the same kind of reasoning with literals, an usually rely on an additional component, called an *oracle*, to handle them [5].

It follows that OWL makes a strict distinction between *object properties*, which can only be predicate between two resources, and *datatype properties*, which can only be predicate between a resource and a literal. Those two kinds of properties inherit the differences of their different values. On the one hand, object properties can be transitive, symmetric, functional (i.e. a resource may have only one value for that property) or inverse-functional (i.e. a resource may be the value of only one resource for that property). The inverse property of an object property can also be defined.

On the other hand, datatype properties have much less options. Obviously, stating that they are transitive or symmetric would make no sense since the subject and object of such properties are not of the same kind. For the same reason, it is not possible to define the inverse of a datatype property (remember that literals can not be subject). The only thing one can state about a datatype property is that it is functional. Note that it is impossible to state that a datatype property is inverse-functional.

Now, let us consider the following example (illustrated in figure 1). We represent persons as resources. We define the object property `spouseOf`, which is inverse-functional[3], i.e. no two persons can be spouse of the same third. In other words, if we know that x is the spouse of y and z is also the spouse of y, then we can infer that x and z represent the very same person. We also want to represent the social security number of people by the datatype property `ssn`. We can state that `ssn` is functional (i.e. a person has at most one social security number), but we can not state that it is inverse-functional (i.e. that no two persons share the same security number), hence we can not infer that individual u and t are the same, having the same `ssn`.

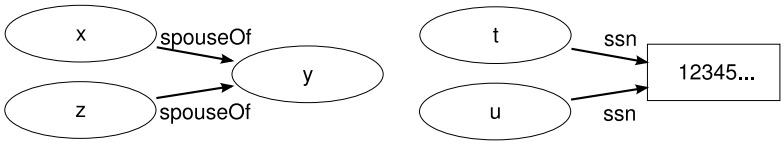

Fig. 1. Example of inverse-functional properties. Ellipses represent resources, rectangles represent literals.

We see here that some reasoning tasks can be performed when they imply only resources (the `spouseOf` property), but not when they imply literals. This

[3] It would also be functional and symmetric, but we will only use the fact that it is inverse-functional in the example.

is all the more disturbing that the relational model, usually considered as semantically less expressive than description logic, has no problem dealing with *unique indexes* (notably used for primary keys), which have the exact semantics of inverse-functional datatype properties[4]. So we may want to limit our use of literals, and represent some data values with resources, in order to take full advantage of OWL inference, especially when representing relational data in OWL. To benefit from the kind of reasoning described above, we need to ensure that each data value has a *single* URI, so as to be recognized as the same value by inference engines. The next section proposes such a unique URI for number values, text values and tuples.

3 Vocabulary definition

In this section, we give and discuss the formal specification of our vocabulary for representing data values as resources. Figure 2 describes the vocabulary as a BNF grammar: terminals are represented between double quotes (`"`), optional elements are postfixed with a question mark (`?`), elements which may be repeated zero to many times are postfixed with a star (`*`) and alternatives are represented by the vertical bar (`|`). The rest of the section discusses some design choices.

```
data_uri    :=  namespace suffix
namespace   :=  "http://liris.cnrs.fr/2006/08/data-uri#"
suffix      :=  simple_val | tuple_val
simple_val  :=  number_val | text_val

number_val  :=  "num:0" | "num:" "-"? mantissa ("e" exponent)?
mantissa    :=  digit ( ("0" | digit)* digit )?
exponent    :=  ("-" | "+") digit ("0" | digit)*
digit       :=  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

text_val    :=  "txt:" url_encoded_utf8_string

tuple_val   :=  "tpl:" element_val ("%00" element_val)*
element_val :=  simple_val | ""
```

Fig. 2. The specification of the `data-uri` vocabulary

[4] Relational databases have another capability lacking from OWL: the unicity constraint may hold on a *set* of columns rather than a single column. Our approach partially addresses that other problem by identifying tuples as resources (see 3.4). A more general proposed solution for so called Combined Inverse Functional Properties (addressing both object and datatype properties) is discussed at `http://esw.w3.org/topic/CIFP`.

3.1 About the limited number of datatypes

While efforts like XML-Schema [6] propose an extensive number of datatypes, we chose to limit ourselves to a very restricted number: basically, number and text. When considering the numerous types proposed in by XML-Schema, we can find two intertwined motivations for their abundance: classifying datatypes and expressing semantics.

The main goal of XML-Schema is to precisely constrain the kind of data that an XML document can contain, hence a plethora of subtypes for `decimal` and `string`. The subtypes may be characterized by structural constraints (e.g. `token`), semantic constraints (e.g. `positiveInteger`), or even implementation-related constraints (e.g. `int`, `byte`). Furthermore, XML-Schema proposes other top-level types, such as `boolean`, `date` or `dateTime`.

Since we represent data by full-fledged resources, we do not need to commit to a particular classification of datatypes: we can rely on existing class systems (e.g. OWL classes) to express such a classification whenever we need it (see 4.3).

As for other other datatypes, we consider them as semantic interpretation of more basic datatypes which can always been brought back to numbers or texts (see section 4.1). Indeed, everything in a computer is represented by numbers and one could argue that even text, from that perspective, is a semantic interpretation of a number in the computer's memory[5]. The limit between "basic" and "interpreted" datatypes is definitely subjective. Our motivation for considering only those two basic types was to make the vocabulary as simple as possible, while remaining usable in most cases.

3.2 About the representation of numbers

A (non-null) number is represented by a signed integer mantissa m not ending with 0, optionally followed by a signed exponent e, the semantics of which being classically $m.10^e$. The absence of the decimal point in the mantissa, as well as the strange constraint on the last digit, are quite unusual. Let us recall that we need each data value to have a *unique* URI, or inference engines may overlook the equality of differently represented values. Relying on the exponent to represent the position of the decimal point with regards to non-null digits, we guarantee that each decimal number is represented in a unique way, if somewhat unusual (e.g. 1e+1 instead of 10).

One may also remark that only decimal numbers can be represented in our scheme; rational numbers with an infinite decimal representation, such as 1/3, can not be exactly represented. We do not consider this limitation to be too serious since classical atomic types can not represent those values either –as a matter of fact, binary floating point numbers are well known to be unable to exactly

[5] The opposite reasoning is also possible: everything in the computer's memory is a *representation* in some language of other entities; from this point of view, text is the most primitive representation, while numbers are one particular interpretation of some texts.

represent even simple decimal numbers. Furthermore, tuples (see section 3.4) can be used to exactly represent ratios if required.

3.3 About the representation of text

The production rule `text_val` only states `url_encoded_utf8_string`; this requires explanations. Recall that UTF-8 is an encoding algorithm allowing to represent any sequence of unicode character [7] as a sequence of bytes. However, according to [8], URIs can only be composed of a subset of ASCII characters[6]. Other byte values are *URL-encoded*, i.e. they are replaced by the percent (`%`) character, followed by their value as two hexadecimal digits. Those two combined encodings allow for any unicode string to be embedded in the URI.

We set another constraint on text URIs: the encoded text must not contain the null character (i.e. character with unicode value 0). This does not seem to be a severe limitation since that character is a mere artifact rather than a real character (it is not used in any language), hence data containing it would hardly qualify as *text*. We need that limitation because we use the null character as a separator in tuples (see below).

3.4 About the representation of tuples

In addition to simple values (i.e. numbers and texts), the BNF grammar in figure 2 allows to represent tuple values. This may seem contradictory with the considerations of section 3.1, stating that we aim at simplicity. Furthermore, RDF provides standard constructs for lists of resources. The reason again is to allow the unambiguous representation of compound values, in order to identify them. This would become obsolete, should Combined Inverse Functional Properties[7] be supported by inference engines, to allow *sets* of properties to be inverse-functional. However, our solution has the advantage of providing a working solution with existing engines.

A tuple is a sequence of simple values, either numbers or texts; some elements of the sequence may also have no value at all (represented by the empty string `""` in production rule `element_val`). Elements are represented using the same scheme as simple values, and separated by the null character (URL-encoded `%00`), which may not appear otherwhise.

4 Using the data-uri vocabulary

We describe here some ongoing and future developments of this work.

[6] A nice feature of UTF-8 is that all the characters present in the ASCII standard are encoded as a single byte whose value is their ASCII code.

[7] `http://esw.w3.org/topic/CIFP`

4.1 Representing relational databases

As stated in introduction, the initial motivation for this work was the representation in RDF of relational databases[8], in order to reason about the relational schema and data with OWL. Simple value URIs as well as tuple URIs proved useful to capture the semantics of unicity constraints and primary keys in the relational schema. Datatypes not fitting exactly into simple datatypes (number and text) are represented by an intermediate resource, linked to a *canonical* representation as a data resource. For example, dates have canonical representations as text (ISO 8601), number (number of seconds since epoch) or of course tuples (year, month, day). Finally, data resources are linked to the corresponding literal in order to make it easier for existing application to exploit the RDF graph.

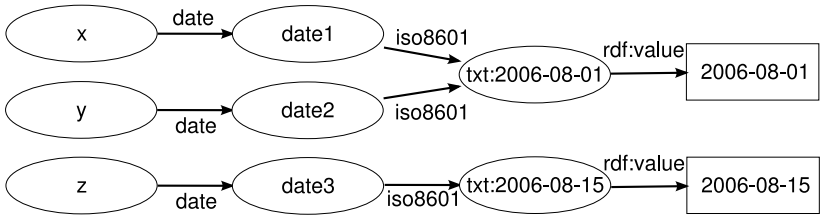

Fig. 3. A simple RDF graph using our vocabulary

An example is given in figure 3, where two resources x and y have a value for the object property `date`. Those resources, as the property name implies, represent values of a complex datatype not handled by our vocabulary. However, they are identified by the inverse-functional property `iso8601`, which link them to the appropriate text resource. Since `iso8601` is inverse-functional, it can be inferred that `date1` and `date2` are actually the same resource. Assuming that `date` is a also inverse-functional (i.e. representing the primary key of a relational table), x and y can in turn be inferred to be identical.

4.2 Identity and difference

Note that in figure 3, the text resource is also linked to the corresponding literal with the standard property `rdf:value`. This is useful for the sake of interoperability: applications unaware of our vocabulary can still rely on the literal value. But this also has an interesting property with respect to reasoning.

We already stressed out in section 2.2 the fact that resources allow more powerful reasoning than literals, because using the same URI twice is recognized as two references to the same resource, which is not the case when using the

[8] See `http://liris.cnrs.fr/~pchampin/dev/cross`.

same literal value twice. However, the use of two different URIs is not necessarily considered by OWL as references to different resources. Indeed, a resource may have several URIs (hence the use of inverse-functional properties to unify synonym URIs).

OWL inference engines will therefore not assume that data resources are distinct, even if their URIs are different, unless we explicitly state that difference, which is not reasonable if we handle a large number of data resources. It seems that, by gaining the ability to recognize identity between data values, we lost the ability to recognize difference (ability that we had with literals).

An elegant solution is to use a *functional* datatype property (`rdf:value` in our example) to link every data resource to its corresponding literal. This forces resources with different values to be different themselves. For example, in figure 3, resources `txt:2006-08-01` and `txt:2006-08-15` would not be deemed different based on the sole fact that they have different URIs. However, since they have different literal values for functional property `rdf:value`, they are necessarily different. We see that, by combining the use of resources and literals, we manage to keep the advantages of both.

4.3 Ontologies of data

As stated before, we can envision to use OWL (or any other RDF class system) to define classes containing the resources we defined in this paper. Classes `Number`, and `Text` seem obvious, but we could also imagine any relevant subclass hierarchy for them, instead of being limited to the classification proposed by XML-Schema. It is possible, for example, to define `Odd` and `Even` as two disjoint subclasses of `Number`.

A problem remains however: stating exactly *which* resources are instances of those classes is not practically achievable, for there can be an infinity of such resources (as in the Odd/Even example). A consequence is that we are able to reason with values themselves (as demonstrated in section 4.1) or with classes of values, but it is not trivial to link them.

A first solution would be to systematically add information about the data resources used in an RDF graph. For example, when using the resource for number 2, one would add a triple stating that this resource in an instance of `Even`. From that, it could be inferred that 2 is also an `Number`, and is not an instance of `Odd`. The drawback of this approach is that it duplicates information about the data resources in every graph using them, and that it is dependant on the ontology. However, it would enable existing inference engines to make the link between instances and classes.

A second solution, that we experimented, is to take advantage of the datatype oracle of OWL inference engines. As suggested in section 4.2, we link every data resource with the corresponding literal. We can then define data classes based on the datatype of their corresponding literal (e.g. the class `Integer` is the class of data resources linked to an `xsd:integer` literal). The drawbacks are similar to those of the first approach: the literal value of data resources must be duplicated in every graph using them, and only the classification known by the datatype

oracle can be inferred that way. It has however the advantage of working with existing inference engines, in addition to the advantage stressed in section 4.2.

A third solution would be to instrument inference engines so that they could *delegate* the reasoning about data resources to other specialized engines, much in the way the current implementations delegate the reasoning about literals to an oracle. Theoretical work about this kind of distributed reasoning exists in the literature [9, 10], and some of them are implemented by OWL inference engines[9]. It seems to be, in the long run, a more powerful and extensible approach than the one of oracles, for some part of the reasoning can nevertheless be performed locally by standard inference mechanism.

5 Conclusion

We presented an RDF vocabulary for representing data values as resources. After demonstrating some advantages of this approach over the use of RDF literals, we gave the formal specification of the vocabulary, discussed some design choices and presented experimented and intended uses of the vocabulary. We think this work opens a number of interesting research directions for Semantic Web technologies and data integration.

Conversion functions from and to URIs of our vocabulary have been implemented as an open-source library, and are available at `http://liris.cnrs.fr/~pchampin/dev/data-uri`. The author would like to thank Djamal Benslimane and Geert-Jan Houben for their comments on that paper.

References

1. Manola, F., Miller, E.: Resource Description Framework (RDF) Primer. W3C Recommendation, `http://www.w3.org/TR/rdf-primer/` (2004)
2. Dean, M., Schreiber, G.: OWL Web Ontology Language. W3C Recommendation, `http://www.w3.org/TR/owl-ref/` (2004)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
4. Connolly, D.: Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C Working Draft, `http://www.w3.org/TR/grddl/` (2006)
5. Baader, F., Kusters, R., Wolter, F. [3] 227–269
6. Biron, P.V., Malhotra, A.: XML Schema Part 2: Datatypes Second Edition. W3C Recommendation, `http://www.w3.org/TR/xmlschema-2/` (2004)
7. Graham, T.: Unicode: A Primer. MIS Press, M&T Books, Foster City, CA (2000)
8. Berners-Lee, T., Fielding, R., Irvine, U., Masinter, L.: Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, Internet Engineering Task Force (IETF) (1998)
9. Serafini, L., Tamilin, A.: Distributed reasoning services for multiple ontologies. Technical Report DIT-04-029, University of Trento (2004)
10. Grau, B.C., Parsia, B., Sirin, E.: Working with multiple ontologies on the semantic web. In McIlraith, S.A., Plexousakis, D., van Harmelen, F., eds.: ISWC 2004. Volume 3298 of Lecture Notes in Computer Science., Springer (2004) 620–634

[9] `http://www.mindswap.org/2003/pellet/`