Making Linked Data Reasonable using Description Logics, Part 4

by Mike Bergman - Monday, February 23, 2009

http://www.mkbergman.com/478/making-linked-data-reasonable-using-description-logics-part-4/

Concluding with a Simplified Instance Record Vocabulary for Linked Data ABoxes

In Part 1 of this series, I advocated the placement of linked data in an ABox construct from description logics [1] based on a separation of concerns argument. In Part 2, I reinforced that argument from the perspective of the ***work*** to be done within a knowledge base. In Part 3 we surveyed some of the key literature, finding justification for the split of the TBox from the ABox and the use of specialty RDFS and OWL dialects for work-oriented reasoning in the context of an integral logics.

We now conclude this series and try to bring these threads full circle to address what might be a vocabulary for an ABox instance record design. We'd very much like to thank Dr. Jim Pitman of the Bibliographic Knowledge Network project for having stimulated much of the thinking about the benefits and design of simple, human-authored and -readable instance records.

A Re-cap

Up until about six to eight months ago Fred Giasson and I were spending much of our thinking and design time on UMBEL, ontologies and what we now more precisely define as the TBox. Our intent all along was to get our process and thinking down pat there, and then turn ourselves to the representation of the actual entity data.

We have wanted to keep data records separate from logic and structure all along. Some clients have their own specific data records but may still want to interact with Web stuff or apply similar logic. Moreover, some client data is proprietary, some public. By organizing the data into "named entity dictionaries" we could modularize the architecture to allow swapping in and out of data appropriate to the customer or circumstance at hand.

Our initial design of this and what we share publicly has UMBEL and various standard public ontologies (FOAF, DC, SIOC, BIBO, etc) for the TBox, with Wikipedia entities and stuff from the BBC at the entity level (the ABox).

However, earlier work with another client showed us that our initial named entity structure was not sufficiently general or robust. That company's records have complex relationships, such as affiliations for entities embedded in the same data record.

For linked data to become truly successful, we need to find easier ways for data publishers to write, expose and share structured data on the Web.

In order to improve the design, we went back to the drawing board to see if we could find guidance from the literature and other researchers as to how to "best" architect instance data in relation to the logic in the TBox (though we were not yet thinking and framing our questions *viz* description logics, or DL).

This series of postings itself, and some of its predecessor articles, were motivated by probing the description logics space and the guidance it might provide to help determine performant architectures and designs.

Folks, We're Making Linked Data Just Too Tough

For linked data to become truly successful, we need to find easier ways for data publishers to write, expose and share structured data on the Web.

As anyone who reads my blog knows, I frequently rail against poor semantics or other aspects of the linked data space that I feel are counterproductive. At the same time, I'd like to think that I am also a vocal advocate and proponent for linked data. I am indeed a fan.

To me, the fundamental precepts of RDF as a data model able to capture virtually any data structure or relationship, and the use of Web URIs as linkable identifiers for a global 'Web of Data', are simply foundational and game changing. Stuff like this quickens my pulse.

But look at what it takes someone today to publish linked data:

- He must understand the terminology and standards and best practices -- and actually, even amongst current practitioners, few do
- She must assign Web identifiers (URIs) to her data objects, which means finding them and making them (gawd, I hate this word) "dereferencable"
- He must understand the semantics of the relationships and linkages his data asserts (which, unfortunately, many don't)
- She must present her data in serialized subject-predicate-object "triples", which are arcane and difficult for most to understand, and
- They both often confuse data and instances with structure and world views.

Now, come on. This is not the recipe to success.

Simple and *unbreakable* and *forgiving* is the recipe to success.

As I noted in an earlier posting, there are many different data structures ('*structs*') for describing and conveying (transmitting) data records. Most of these are easy to understand and easy to read. We know

that microformats have tried to capture a part of this space, but so has in other ways data serializations such as JSON or others. What can we learn from such formats?

Well, one thing I have learned is that many on the Web positively want to expose their data. Another thing I have learned is that there is much structured data that will not get exposed without hurdle rates that are ***small***.

Revenge of the ABox

The phrase 'revenge of the ABox' comes from Heiko Stoermer's thesis [2]; it conveys well, I think, the fact that everyone wants to capture and structure "world views" via ontologies and the big picture, but many do not want to grub around at the level of individual instances and data records. As he states, ". . . the most valuable knowledge is typically the one about individuals, but research on ontology integration has traditionally concentrated on concepts and relations."

(The perverse outcome of this is that even though linked data as practiced to date is almost 100% about instance data, the discussion rarely looks at ABox-level work or instance data integrity.)

As this series and its predecessor posts have argued, description logics (DL) is an excellent guiding framework for how to make architectural and design decisions about linked data. DL and the ABox - TBox have meshed beautifully with our earlier intuition to split ontologies and a structural and organizational view of the world (TBox) from the instance records (ABox, or what we had been calling internally our 'named entity dictionaries').

As this four-part series and its predecessor pieces indicate, not only can we gain better conceptual understanding and realization of some of this semantic Web stuff by using DL, but also, perhaps, many of today's silly or inefficient design practices may be remedied by better grounding our architectures in these logics.

One area, for example, that has helped us much is to get away from the confusing terminology of 'individuals' v 'instances'. Once we come to see an instance record as just that (so, that is why collections can play on an equal footing with individual things, for example), we now only need worry about asserting the attributes of the instance. We can defer all of the logic and reasoning about individuals and members and sets and collections and classes, etc., to the TBox and just get on with capturing and conveying our instance record, as an ABox.

For this reason alone (but there are others), Structured Dynamics has now abandoned the terminology of a 'named entity dictionary' in favor or 'instance dictionaries' or ABox (either term of which is understood to contain one or more instance records).

The 'Instance Record'

An instance record is simply a means to either represent or convey the information ("attributes") of a given instance. An instance is the thing at hand, and need not represent an individual; it could, for example, represent the entire holdings or collection of books in a given library.

An instance record may convey information about multiple instances, but each block of information for each instance is about that instance alone. Thus, for example, if the instance is a paper citation, the instance is the paper. If as attributes it asserts multiple authors, each with different institutional affiliations, those affiliations get asserted in a separate instance for each author. They are attributes of the authors, not of the paper.

In this manner it is easy to see attributes as only pertaining to a given instance. If the overall information to be conveyed discusses attributes for multiple instances, than the instance record presents in series each instance that is characterized.

The Simplicity of Key-Value Pairs

The objective is to make it easy for data owners to write, read and publish data. This means the starting format should be a human readable, easily writable means for authoring and conveying these instance records (that is, instances and their attributes and assigned values).

The simplest, naÃ¯ve format (independent of syntax or serialization) is the key-value (name-value) pair. In the key-value pair, the *subject* is always implied. So, for me, MikeBergman, as the subject:

```
first_name:Mike
sex:male
citizenship:USA
town:Iowa City
```

Because an instance record only describes attributes for a single instance at a time, all assertions can easily be transformed into the *subject-predicate-object* (*s-p-o*) "triples" of RDF. So,

```
<subject:MikeBergman> <hasFirstName> <Mike>
```

Now, of course, in conventional linked data many of these entries need to be expressed as URIs in order to "define" the item. Our design allows for that, of course, but also allows the user to simply provide literals (that is, not identifiers, but text strings or numeric or actual values) for each item. Thus, the declaration of a "new" attribute only need occur by its expression, with its value also as simply declared.

Separate, specialized services (see below) may be (and often will need to be!) employed to look up and de-reference URIs, do datatype or data instance validation checks, evaluate identity relationships, disambiguate terms and so forth. The data supplier may choose to publish more-or-less complete "records" on their own, or they may not.

Through this design, nothing need change with regard to how linked data is being done today (other than the addition of some simple converters to accommodate the new format; see below). But, by shifting testing and validation work to external services, we can make it much easier for more data to get exposed and published. It is now time for linked data intermediaries and services to evolve in the linked data ecosystem.

In its most naÃ¯ve form, this key-value pair format allows for fast and easy instance record creation with

the ability to create instances and new attributes on the fly. Sure, these assertions need to be checked, but so does most data when it is asked to participate in any meaningful work.

This simple design, then, is very much in keeping with the limited roles and work associated with an ABox. Only attributes and metadata for an instance are being asserted. Conceptual relationships and specialized work that might be applied against the ABox to determine data validity or whatever is shifted to be external to the instance record, where it properly and logically belongs.

Relation to RDF

In Part 3 we discussed how fragments of the RDF and OWL languages can be used for specialized purposes within a knowledge base while keeping the overall logics of the system integral and decidable. Clearly, this instance record approach where the sole purpose is to assert attributes and values for an instance does not require any OWL. In fact, most linked data to date only brings OWL into the picture for the `owl:sameAs` property, the common errors of which we discussed in Part 2.

The instance record only requires a small subset of the RDF language. But it does require use of RDFS (Schema) because of the appropriate use of datatypes within the instance data record.

At the level of the TBox and the "specialized work" areas, there are other fragments of OWL, now called profiles in the soon to be released OWL 2 [3], that similarly can be applied to areas such as instance checking and validation, identity relation testing, etc., that I mentioned above. In other words, we can logically fragment RDF and OWL to do the individual parts of a complete system in order to simplify things and aid performance and computational efficiency.

The Instance Record Vocabulary

We are implementing this design internally through what we call the *Instance Record Vocabulary* (QName: irv). It is still quite experimental and we are testing some important aspects, some of which we describe below. As we get these nuances worked out better, we will release this vocabulary publicly for any to use and comment.

As we presently see it, the namespace languages required for the IRV vocabulary are RDF, RDFS, DCterms and XSD. The RDFS (Schema) is required because, at minimum, of the incorporation of XML Schema datatypes (XSD), which we think to be a desirable requirement for what is, after all, an instance data specification and transfer protocol. However, the actual RDF and RDFS vocabulary used would be extremely minimal, with no OWL required.

In pseudo-form, with many serializations and simple syntaxes possible, this *Instance Record Vocabulary* has the following properties. Note as discussed above that the <*s*> in *s-p-o* is implied. Thus, in its naÃ¯ve or handwritten form, it could be expressed in pretty simple key-value pairs:

```
<InstanceRecord>
<Instance>
<hasLabel> <[literal]> @en
<hasAltLabel> <[literal]> @en
```

```
<hasURI> <[URI]>
<hasDescription> <[literal]> @en
<Attribute>
<hasAttribute1> <[literal with optional XSD (@en) or URI]>
<hasAttribute2> <[literal with optional XSD (@en) or URI]>
<hasAttribute3> <[literal with optional XSD (@en) or URI]>
<hasAttributeX> <[literal with optional XSD (@en) or URI]>
</Attribute>
<assertIdentity> <[literal or URI]>
<assertType> <[literal or URI]>
<hasSource> <[literal or URI]>
<hasVetting> <[literal or URI]>
</Instance>
<Instance>
. . . repeat as needed . . .
</Instance>
</InstanceRecord>
```

Note that most values allow either literal or URI specifications. Some of the properties are obviously optional, others, such as `hasLabel`, will be required. `hasURI`, for example, is one case of an optional property that then may require a separate lookup service to complete it as a linked data record.

Instance records with literal specifications would need to be validated and checked before actually used for standard linked data or meaningful data purposes. However, this approach is already well-proved through, for example, OpenLink's Virtuoso Sponger cartridges and design. Sure some work would need to be done at time of ingest, but there are no technical challenges.

The language used to write a literal can be specified for any kind of attribute (metadata or not). The language is specified using the "`@lang-tag`" at the end of the literal. This method is similar to the N3 serialization of RDF, which is also equivalent to the XML serialization of RDF using the "`xml:lang`" attribute.

Metadata

Most of the first properties are simply metadata describing the instance. The strings could be qualified by language.

Attributes

The bulk of the instance record is devoted to the attributes and their values. Attributes could be optionally declared with XSD datatypes. URI references could be specified or later substituted by vetting services (see below).

Attributes could also optionally be characterized in a list format, similar to the Lists specification for Notation 3 (N3).

Asserted Relations

Identity and class membership (`rdf:type`) assertions could be made; these could later be checked for correctness or identity relations with external or specialized services. The `assertIdentity` property, in particularly, is the replacement with more appropriate ABox semantics for `owl:sameAs`.

hasSource

A separate Source record is being developed to cover source or dataset characterizations. A single instance extraction from a Web page, for example, would be accompanied by a simple source characterization. Instances of particular types, such as microformats for example, would be so noted (as they might invoke specialized processors or carry certain authority). Instances from large datasets would have a still longer list of possible characterizations.

This property may look closely at what is also being done for the voiD dataset vocabulary.

Certain parameters in a Source record, such as language for example, may also be applied in special ways by the IRV parser at time of ingest with respect to specific literal specifications.

In any event, this is one of the properties still needing much more thought and definition.

hasVetting

This property, too, needs much more thought and definition.

The `hasVetting` property, for which multiples are allowed, would identify the specific checks and services applied to the instance data. Depending on service, such checks might include URI lookup or de-referencing, identity relations and testing, record completeness and sufficiency checks, data type checking and validation, general instance checking, disambiguation, and so forth (see "*Specialized Work*" below).

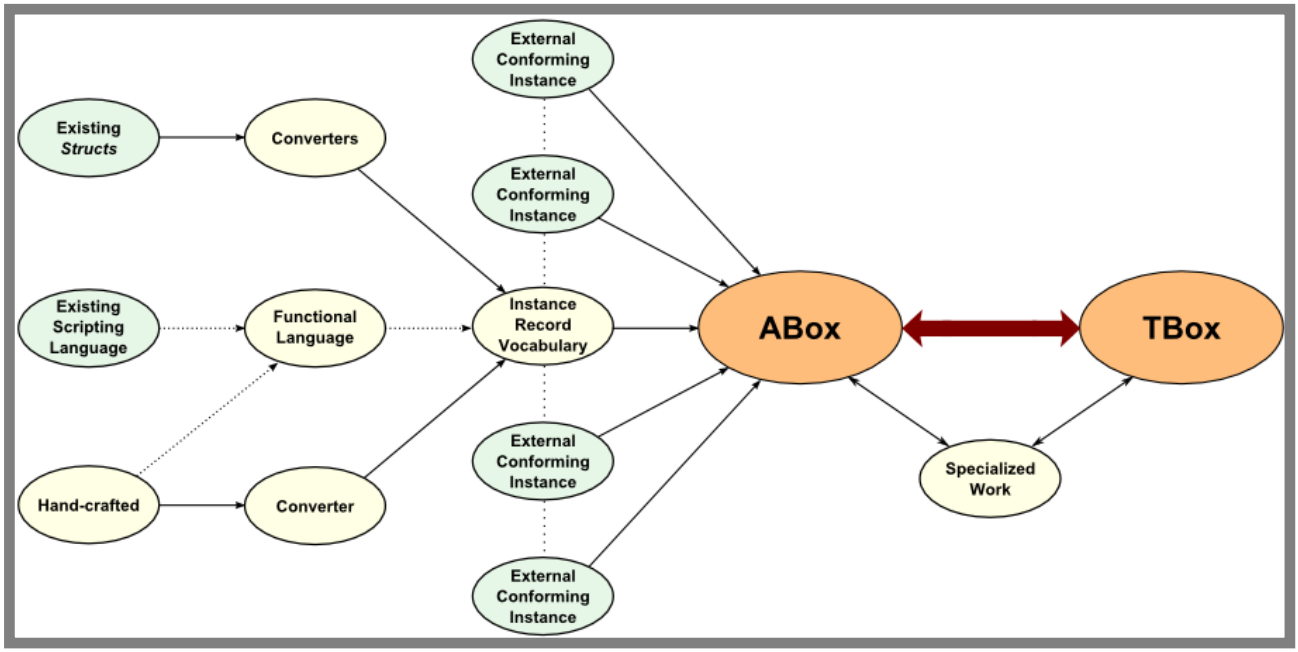
Some services might also re-write the instance record with corrected values or URIs returned in place of literals.

Best practice for external services would suggest identifying them by URI, though literals would also be allowed to identify internal checks or for lookup purposes.

This property is meant to be a key indicator of how third parties may want to rely on the data. Combined with `hasSource`, these `hasVetting` entries provide essential authority and provenance information about the data at hand.

Putting it All Together

This diagram attempts to show the relationship of how many of these pieces may interact:



Some of these bubbles deserve some additional commentary.

Hand-crafted Input

An important objective in this design is to allow naïve, simple text specifications to be hand-crafted for instance records. There are many relatively simple formats for specifying key-value pairs with a relatively few conventions, ranging from BibTeX to YAML and JSON and others. There are literally hundreds of such formats available, as my earlier overview of Naïve Representations and Structs discussed.

There may be justification for still another form in relation to this *Instance Record Vocabulary* or not; this topic is still under active discussion.

External Structs

However, whether there is a separate format or not, that same earlier piece overviewed the many simple data *structs* presently out there. It also noted the nearly 100 existing converters for these forms to RDF. These same converters, with quite slight modifications, could all output the *Instance Record Vocabulary* in an appropriate serialization as well.

Hooks to Functional and Scripting Languages

Another option is to combine this design with a functional language front-end to generate these records. (Though they could be produced in other ways, as well.) For example, lambda calculus or even a domain-specific language (DSL) could be used to create this very simple record generator. This simple system, in turn, could have a straightforward API that would allow existing scripting languages (such as Python or others) to be used as well.

Specialized Work

So, in fact, we can also now see the specialized work (see also Part 2) that itself is not part of the ABox but can and often should be applied to the instance data in the ABox:

- Record sufficiency checking
- De-duplication
- Membership testing
- Most specific concept identifying
- Datatype checking
- Identity relation testing
- New attribute checking
- ABox consistency testing
- Data range checking
- Disambiguation
- Source-specific testing
- Uniqueness testing
- URI lookup
- URI de-referencing
- Satisfiability checking
- Others . . .

Though, strictly speaking, such specialty work could be seen to occur at the TBox level, it is actually different and separate logic from "standard" inferencing or reasoning. Specialized work can therefore often occur as separate tests or in batch mode with fragments of OWL or other dedicated indexes and algorithms. Some of this specialized work may take advantage of the conceptual relationships in the TBox, but may not necessarily need to do so. In these manners, the inferencing work of the TBox can be kept clean and efficient.

Beyond Browsing and Unvalidated Queries

Today, linked data has largely been used for browsing and providing unvalidated responses to queries; focus and attention to its ABox roles are important to move beyond this baseline into meaningful work [2]. In those limited instances where this linked data has been looked at and evaluated as a complete knowledge base, such as the SWSE search engine with the SAOR approach as discussed in Part 3, more than 97% of the RDF triples provided in those cases were removed from consideration, often for logical or mis-assertion reasons [4].

The ideas presented here for a simpler linked data specification that can be easily represented in readable text is not new. RDF in JSON has been looked at in this way by Talis and JDIL, YAML has been looked at similarly, and similar and simpler approaches have been looked at closely for topic maps. There are other examples.

A key thrust of these efforts is to make it easier for the data publisher, thereby encouraging the exposure of more structured data.

These emerging ideas do not change in any way the usefulness of current linked data. Our suggested approach interoperates seamlessly with current practices and easily co-resides with them. But, these ideas do:

- Provide a simpler path for writing and publishing human-readable instance data
- Provide an ABox instance record structure that can have much specialized work applied against it

in a consistent way, and
- Contributes to an overall logic and architecture that is performant and scalable for doing meaningful work.

Though still needing further thought and refinement, this broad outline of roles and architecture and structure for the ABox completes the last missing piece to Structured Dynamics' overall approach to linked data and RDF. Much time, thought and research have gone into it. Again, we'd very much like to thank Jim Pitman for his ideas that have helped catalyze this design [5].

We think the combination of a generalized *Instance Record Vocabulary* that can be reasoned over for ABox-level data checking, and that works with a simple, text-based key-value pair input format, might be a winning combination.

---

[1] This is our working definition for description logics:

"Description logics and their semantics traditionally split *concepts* and their relationships from the different treatment of *instances* and their attributes and roles, expressed as fact assertions. The concept split is known as the TBox (for *terminological* knowledge, the basis for *T* in *TBox*) and represents the schema or taxonomy of the domain at hand. The TBox is the structural and intensional component of conceptual relationships. The second split of instances is known as the ABox (for *assertions*, the basis for *A* in *ABox*) and describes the attributes of instances (and individuals), the roles between instances, and other assertions about instances regarding their class membership with the TBox concepts."

[2] Heiko Stoermer, 2008. *Okkam: Enabling Entity-centric Information Integration in the Semantic Web*, Ph.D. thesis presented to the DIT - University of Trento, January 2008, 185 pp. See http://eprints.biblio.unitn.it/archive/00001394/01/dissertation_camera_ready.pdf.

[3] Boris Motik *et al.*, eds., 2008. "OWL 2 Web Ontology Language: Profiles," a *W3C Working Draft*, December 2, 2008. See http://www.w3.org/TR/owl2-profiles/.

[4] Aidan Hogan, Andreas Harth and Axel Polleres, 2008. "Scalable Authoritative OWL Reasoning on a Billion Triples," in *Proceedings of Billion Triple Semantic Web Challenge 2008*, at the *7th International Semantic Web Conference (ISWC2008)*, Karlsruhe, Germany, 2008. See http://sw.deri.org/~aidanh/docs/saor_billiontc08.pdf.

[5] This input has come as a result of research supported in part by NSF Award 0835851, Bibliographic Knowledge Network.
