fhiso

CFPS 97
(Call for Papers Submission number 97)

Type safety in an extensible data model

Submitted by: Smith, Richard

Type: A comment on a submitted paper

Comment on: 79

Created: 2013-11-25

URL: Most recent version: http://fhiso.org/files/cfp/cfps97.pdf
This version: http://fhiso.org/files/cfp/cfps97_v1-0.pdf

Description: This paper resolves some type-safety issues raised in CFPS 79 about the statement formalism of CFPS 77, though which apply equally to CFPS 4's node formalism.

Keywords: data model, statement, node, type safety, extensibility, discovery

Abstract

This paper considers one specific objection to the statement framework of CFPS 77 in regards to its perceived lack of type safety. It is demonstrated that in this respect the statement and node formalisms have precisely the same deficiencies, although the different choice of language used in the two formalisms may have obscured the similarity. Four types of vocabulary item are identified as present in both formalisms: namely datatype properties, object properties, classes and datatypes. The paper goes on to discuss what information an application requires in order to ensure type safety. Finally, three possible means of discovering this information are discussed.

1 Introduction

The paper *The Principle of Sensible Disbelief* (CFPS 79) contains the following observation [1]:

> While CFPS 77 does increase the generality of CFPS 4, [...] it also adds in a level of generality that allows various nonsensical and incomplete data to be encoded without any null or broken pointers or other data-structure-level inconsistencies.

It is true that certain examples of nonsensical or incomplete data could potentially be encoded in the statement formalism of CFPS 77 [2], but exactly the same data can be encoded in the node formalism of CFPS 4 [3]. In this respect the two formalisms are identical. (They differ in some other respects, such as how metadata and information about the beliefs of researchers is considered. These differences, however, are beyond the scope of this paper.)

2 Labels *versus* pointers

It may be that some of this confusion is simply a confusion due to the different terminology used in the two formalisms. A *reference* is the term CFPS 4 uses when one node refers to another; for example, it states that "a property node has a single reference to another node of any type". With one exception, CFPS 79 uses the term *pointer* instead of reference. Both terms have connotations of objects in memory being referred to by an address. Perhaps an object-oriented data model is imagined. Perhaps it suggests some form of compiler or run-time mechanism to enforce valid references. But neither paper places any such requirements.

By contrast, CFPS 77 uses the term *label* in referencing entities. If two labels are the same, they refer to the same entity. It is certainly understandable if the use of this terms brings to mind a weaker sort of link, a link without any means of enforcing

reference integrity, and where a typo could easily cause an unintended reference. But just as neither CFPS 4 nor CFPS 79 made any requirement on pointers integrity, CFPS 77 made no prohibition on label reference integrity.

The difference in the language used is mostly down to a difference between the language of an in-memory object model and the language of a serialised object model. Pointers are a common technique for implementing references between the in-memory representation of objects, but it makes little sense to talk about pointers in a serialisation. They are typically converted to labels in some implementation-defined way. Equally, labels are generally a poor choice of implementation strategy when representing objects in-memory.

Certainly in writing CFPS 77 no particular implementation was intended, either in memory or in the serialisation. Indeed, the obvious in-memory of CFPS 77's statements would a node-like object, and the links between them might naturally be pointers. The statement formalism was not conceived as an alternative to the node formalism: rather it was intended to simplify the specification

Is there, then, any reason to prefer a specification in terms of statements rather than nodes? Arguably yes. The in-memory representation of objects is an implementation detail best left to the application. Some applications will hold objects in-memory, while others will choose to use a relational database or a 'NOSQL' database. For those applications that delegate to a database engine, the pointer-based formalism is inappropriate. By contrast, the label formalism is directly applicable to the serialisation of any data model (at least for a structure more complicated that a tree). In GEDCOM, for example, labels are used to link individuals to family records and the GEDCOM specification is written in those terms; nevertheless, many applications probably implement those links using pointers [4]. Other links in GEDCOM are implicit, such as the link between an individual and their events and attributes. Very likely the same will be true of a future FHISO serialisation format.

3 Things, connections and properties

The three primary types of node in CFPS 4 are the *thing* node, the *connection* node and the *property* node. It was not the contention of CFPS 77 that there was no meaningful distinction between these, nor is that the contention of this paper. But they share similarities too: each talks about a node — its subject in the statement formalism. They differ in what else they contain — their objects in the statement formalism. The objects of a thing node, connection and property, are respectively a class name, another thing node, and a literal.

The predicate in the statement determines whether the object is a class, another

node, or a literal. For example, the 'name' predicate might be defined always to have a string literal as its object, and the 'date' predicate might be defined always to have a date literal as its object. In the terminology of the W3C's OWL language (a language for defining RDF vocabularies), predicates such as these are called *datatype properties* [5] because their object is a literal of some particular datatype, such as a string or date.

Other predicates such as a 'happened at' or 'participated in' connection might be defined to have a place or an event as its object. In OWL's terminology, such predicate are called *object properties*. Where there are current applicable standards using a particular terminology, it seems wise, in the absence of a good reason to the contrary, to harmonise the FHISO terminology with such standards. This paper therefore recommends using OWL's the terms, 'datatype property' and 'object property'. The 'type' predicate used to define thing nodes is neither.

It is an application's responsibility to ensure that a predicate is only ever used with the correct type of object, and this paper does not seek to prescribe a particular implementation strategy for doing so. An application written in a strongly-typed language (such as Java or C) might use typed pointer so that, for example, the subject and object of a connection node could only be a thing node. A implementation backed by a relational database might do the same. But other applications, such as those written in weakly-typed language (like PHP), or those backed by a less featureful type of database might require a different strategy. This is true in the context of CFPS 4's node formalism just as much as in CFPS 77's statement formalism.

4 Extensibility

This paper defines the *range* of a predicate to mean the type of its object, and the *domain* of a predicate to mean the type of its subject. The terms 'range' and 'domain' are aligned with OWL, which is in turn aligned with RDF Schema [6]. When defining a predicate, its range and domain should be specified. When the predicate is an object property (i.e. a connection per CFPS 4), its range will be class such as 'person' or 'event'; and when it defines a defines a datatype property (what CFPS 4 calls a property), its range will be a datatype such as 'string' or 'date'. It would be possible to define the type, domain and range of the 'type' predicate, but it is not clear that this is either necessary or useful; but if desired, RDF Schema gives an example of how it might be done.

Just as an application must ensure, in some implementation-defined way, that the object of datatype property is a literal rather than another node, applications must ensure that the range and domain of each predicate is respected. In the examples in table 1, for example, applications must ensure that the 'date' predicate never

Predicate	*Type*	*Domain*	*Range*
name	*datatype property*	*person*	*string*
date	*datatype property*	*event*	*date*
participated in	*object property*	*person*	*event*
happened at	*object property*	*event*	*place*

Table 1: The range and domain of some example predicates

has 'John Smith' as its value: only a valid date. Similarly, it must ensure the 'participated in' predicate has a person, source or place as its object. How this is achieved should be left up to the implementation.

The formalism proposed here and in CFPS 77 can be thought of as defining, in the terminology of CFPS 20 [7], four distinct partially-controlled vocabularies. Even if the statement formalism of CFPS 77 is rejected, these four vocabularies are also needed in the node formalism of CFPS 4.

Datatype properties
: are the keys used in the name–value pairs in CFPS 4's property nodes. They are likely to be things like 'surname' or 'date'.

Object properties
: are the type labels associated with CFPS 4's connection nodes. Examples given in CFPS 4 were 'participated in', 'happened at' and 'daughter of'.

Classes
: are the type labels associated with CFPS 4's thing nodes. Common example will be 'event', 'individual' or 'source'. Subclasses of these may be defined, such as for 'baptism event'.

Datatypes
: are the means of standardising the structure of the values in CFPS 4's property nodes. An example would be a 'date' datatype. A 'string' datatype is probably required for simple text as in a name.

Irrespective of the terminology used, in each case standardised yet extensible vocabularies are required. As a trivial example, it is no good if one application uses 'surname' where a second application uses 'family name'. They are points of extensibility where applications or future standards may wish to define additional vocabulary. It may also be that the FHISO wishes to adopt a modular approach to standardisation, in which case the core standard need not define any terms in these four vocabularies, and FHISO modules would take the form of a list of new terms, together with careful definitions of their intended semantics.

This paper does not consider how to avoid collisions between independent third-party extensions. One possible means is discussed in CFPS 37 [8], but this paper does rely on that particular mechanism.

5 Vocabulary discovery

It is important that an application encountering an unknown term can still function. At the very least it must store and include the uses of the unknown term in its output unless directed to the contrary by a user. But ideally it should be able to go further. Suppose an application encounters a hitherto-unknown 'amrit sanchar' event. This is a Sikh ceremony with certain similarities to Christianity's confirmation ceremony. If the application can determine that it is an event type, it may wish to allow users to create further events of this type. The same is true to a greater or lesser extent of other forms of extension, though it would be naïve to believe an application would always be able to handle all unknown data seamlessly.

This paper discusses three possible means by which an application can discover about new vocabulary items.

Implicit declaration
: is when an application infers the basic properties of a term from its usage. For example, the 'amrit sanchar' event might be inferred to be a subclass of event because it is used in a context where only a event would be valid, perhaps as the object of a 'participated in' predicate.

Internal declaration
: is when the document using an extension item of vocabulary also includes some information describing the term. With the example 'amrit sanchar' event, the document would somehow declare this as a subclass of the event class.

External declaration
: is when the vocabulary name is somehow associated with a URL from which further machine-readable information about the term can be fetched. In the specific extensibility mechanism proposed in CFPS 37, the URL would be the namespace URI.

This paper requires support for at least one of these mechanisms in a future FHISO standard but reaches no conclusion as to which of them it should be; nor does this paper preclude support for all three. If the FHISO supports implicit declaration, then a future paper is required to document what inferences may be made. If internal or external declarations are supported, then a formalism is required for declaring the intended usage of new vocabulary items. At a minimum this must allow an application to determine the type of a new vocabulary item (i.e. whether it is a datatype property, object property, class or database). For properties, it must allow the range and domain to be specified. For classes, it must allow them to be declared as a subclass of some other class.

References

[1] Luther Tychonievich, 2013, *The Principle of Sensible Disbelief* (CFPS 79), http://fhiso.org/files/cfp/cfps79.pdf

[2] Richard Smith, 2013, *A unified formalism for genealogical statements* (CFPS 77), http://fhiso.org/files/cfp/cfps77.pdf

[3] Luther Tychonievich, 2013, *Modeling Research, not Conclusions* (CFPS 4), http://fhiso.org/files/cfp/cfps4.pdf

[4] Church of Jesus Christ of Latter-day Saints, 1996, *The GEDCOM Standard (Release 5.5)*, https://devnet.familysearch.org/docs/gedcom/gedcom55.pdf

[5] World Wide Web Consortium, 2004, *OWL Web Ontology Language: Guide* http://www.w3.org/TR/owl-guide/

[6] World Wide Web Consortium, 2004, *RDF Vocabulary Description Language 1.0: RDF Schema* http://www.w3.org/TR/rdf-schema/

[7] Tony Proctor, 2013, *Proposal for Handling Partially Controlled Vocabularies* (CFPS 20), http://fhiso.org/files/cfp/cfps20.pdf

[8] Richard Smith, 2013, *Proposal for a scalable extensibility mechanism* (CFPS 37), http://fhiso.org/files/cfp/cfps37.pdf