Information Modelling using RDF

Constructs for Modular Description of Complex Systems

Graham Klyne
BGaltimore Technologies, Content Security Group
1220 Parkview, Arlington Business Park, Theale, RG7 4SA, UK.
`Graham.Klyne@Baltimore.com`

Abstract

This paper describes some experimental work for modelling complex systems with RDF. Basic RDF represents information at a very fine level of granularity. The thrust of this work is to build higher-level constructs in RDF that allow complex systems to be modelled incrementally, without necessarily having full knowledge of the detailed ontological structure of the complete system description.

The constructs used draw on two central ideas: statement sets as contexts (based in part on ideas of McCarthy and Guha[1,2]) to stand for a composition of individual RDF statements that can be used in certain circumstances as a statement, and a system of "proper naming" that allows entity prototypes to be described in a frame-like fashion, but over a wider scope than is afforded by class- and instance- based mechanisms.

Keywords: RDF, metadata, information modelling, semantic web

Table of Contents

1. Introduction

This paper suggests some ways of modelling complex systems with RDF[3]. Basic RDF represents information at a very fine level of granularity, and the RDF description of a system of significant complexity can be very difficult to design and construct. The thrust of this work is to build higher-level constructs in RDF that allow complex systems to be modelled incrementally, without necessarily having full knowledge of the detailed ontological structure of the complete system description.

These ideas are being used in experimental projects to describe and analyse systems consisting of complex interacting components, such as networked computer systems and software packages[19]. One could imagine these techniques being used to describe complex machines like cars, or social processes and interactions. We also plan to use some of these ideas to perform trust modelling.

The constructs used draw on two central ideas:

- statement sets as contexts, based in part on ideas of McCarthy[1] and Guha[2], to stand for compositions of individual RDF statements that can be used or referenced as a single entities, and
- a system of "proper naming", loosely derived from some ideas by Pat Hayes[13], that allows an entity to be referenced using a locally assigned identifier rather than a global URI. This is used to create prototype descriptions in which the assigned name can reference a characteristic of more than one entity.

The goal is to use these constructs to introduce modularity into complex system descriptions. One might view RDF as an "assembly language" for metadata representation; this work builds higher level descriptive constructs on the RDF base, analogous to the use of functions and subroutines in mathematics and programming languages.

This work overlaps semantic web ontological developments such as OIL and DAML–O, but has an emphasis on more easily assembled constructs for expressivity in RDF rather than semantic web inference and proof.

Comments are welcome, and may be directed to the RDF-interest email distribution list[14].

1.1 Terminology

Statement set: a collection of reified statements, whose container is an RDF resource.

Context: a *statement set*, with additional properties capturing certain structural and logical properties associated with contexts, described in this document.

Lifting rule: any rule that allows us to deduce that truth of some statement in a context from the truth of another statement in a related context.

Ontology is unhelpfully defined by my dictionary as “the branch of metaphysics dealing with the nature of being”. As a computer systems engineer I find it more helpful to think of ontology as the application of data typing mechanisms to arbitrary objects and concepts.

Ontological structure is the set of classes or types used in a description of some concept, and the relationship between them. RDF uses ***classes*** to indicate categories of objects of concepts. *Classes* may be related through multiple inheritance, and an RDF resource may be an ***instance*** of any number of classes.

1.2 Descriptive conventions and notation

This paper is about structures in RDF. It focuses entirely on the graph syntax of RDF, described in section 5 of the RDF Model and Syntax specification[3], and does not make any assumption about the particular RDF serialization format used.

This note was originally drafted before N3 (Notation 3) [25] was widely known. Future versions may use N3 in place of the notation described here.

In describing RDF statements, square brackets are used to denote RDF resources, containing a name for the resource, or quoted strings to describe literal object values. RDF properties are shown as labelled arrows from subject to object:

```
[SubjectName] --propertyName--> [ObjectName]

[SubjectName] --propertyName--> "Literal-string"
```

RDF classes are given names starting with an upper case letter, RDF properties with lower case.

Specific resource identifiers may be replaced by the expected `rdf:Type` of the resource in angle brackets. Thus, a general RDF statement might be described as:

```
[<rdfs:Resource>] --<rdf:Property>--> [<rdfs:Resource>]
```

To give a name to the reification of a statement, the following notation is used:

```
Statement-id: [SubjectName] --propertyName--> [ObjectName]
```

which can be interpreted as introducing a resource identified as `Statement-id`, and the following additional triples, corresponding to a reification of the statement:

```
[Statement-id] --rdf:type-------> [rdf:Statement]
[Statement-id] --rdf:predicate--> [propertyName]
[Statement-id] --rdf:subject----> [SubjectName]
[Statement-id] --rdf:object-----> [ObjectName]
```

To apply a property to the reification of each of a set of statements, the statements are enclosed in braces:

```
[Resource] --property-->
  {
  Statement-set
  }
```

This shows that the reification of each member of *`Statement-set`* appears as the object of an RDF statement of the form:

```
[Resource] --property--> [Statement]
```

The namespace prefix '*`rdfc:`*' is used for RDF classes and properties introduced by this paper to represent container and context information. The namespace prefix '*`rdfn:`*' is used for RDF elements introduced to represent naming information. Prefixes *`rdf:`* and *`rdfs:`* are used for namespaces defined in the RDF Model and Syntax[3] and RDF Schema Specification[4]. RDF resources and properties without prefixes are used in hyothetical examples with no defined relationship to any specific namespace.

2. Motivation and background

The main goals addressed by this work are to define RDF mechanisms that:

- create information models using higher level elements composed from basic RDF statements, and
- describe frame-like generic models that can be used as prototypes to simplify the description of more specific entities.

These are required to be usable in an incremental, top-down fashion, where higher level descriptions and usage do not need to be aware of the details of lower level ontological structures.

2.1 Resource Description Framework (RDF)

Resource Description Framework (RDF)[3] is a W3C recommendation for a standard representation of metadata, based on ideas with roots in knowledge representation research conducted over the past 30 years or so. The specification defines an abstract directed labelled graph model for RDF, and an XML-based serialization (there is some discussion about defining other serialization formats for various purposes). The nodes of this graph are RDF resources, and the arcs are RDF properties. A companion RDF Schema specification[4] describes how RDF may itself be used to define a type system based on RDF classes, and constraints on the ways in which RDF classes and properties may be combined in a description.

The importance of RDF is not that it is demonstrably better than any other form of knowledge representation, but that it has a reasonable chance of becoming a widely used Internet standard, and that it is designed for use in an open web environment. To exchange information (as opposed to raw data) between computer systems or applications requires agreement about its representation. To this end, I build upon, rather than extend or modify, the core concepts of RDF, and the resulting framework is fully expressible within the graph structure of basic RDF. This adherence to basic RDF structure does not constrain the internal wortkings of implementations; for example, reified RDF statements may be represented in ways that are far more efficient than that suggested by basic RDF. The relationship to basic RDF structure is needed when exchanging information between applications.

The RDF specification is felt by some to be confusing as an *ab initio* guide to RDF, and there are several papers on the web that provide a gentler introduction. One such introduction that may be particularly helpful is by Pierre-Antoine Champin[23], which starts by explaining the RDF graph model that is the underlying basis for the constructs described here. A topic that often causes particular confusion about RDF, and which is quite fundamental to this paper, is "reification": this is the construct that allows us to make statements about statements, by constructing an RDF resource to stand for any given RDF statement. Thus, we can have a resource identifier that stands for an RDF statement, and using that identifier we can make statements about that statement.

2.2 Describing complex systems

The structures described here evolved from an attempt to describe complex system components in RDF, from which it was required to draw inferences about the compatibility of various combinations of component. Originally, we tried to construct meaningful descriptions using the basic RDF property-subject-object constructs, but found it was very difficult to predict the required graph structure without first assembling all of the information to be described. As new factors came into play, we were finding that existing structures had to be redesigned. Later, we approached the same problem using an informal high-level abstraction to stand for a component description consisting of arbitrary RDF statements. In this way, we found that high-level interactions between components could be captured, and subsequently refined with more detailed information.

Information Modelling using RDF

Suppose I wish to describe the kind of fuel needed to operate my car. A statement like this might be used:

```
[MyCar] --fuelType--> "Diesel"
```

meaning that my car runs on diesel fuel. This is just fine for determining which pump I need to stop at when refuelling. But for an engineer dealing with the car as a piece of machinery, more detailed information is needed, and the `fuelType` might be considered a property of of the car's engine, rather than of the car as a whole. The engine might be replaced with one using a different kind of fuel. At this level of detail, the model suggested above is not adequate, and a different structure is needed; e.g.

```
[MyCar] --hasEngine--> [MyCarEngine] --fuelType--> "Diesel"
```

This simple example shows how simple RDF model ontological structures may need to be adjusted to take account of new levels of detail. If, however, we treat the description of my car as a collection of statements, then their relationship to the overall entity that is my car may be more flexible:

```
[MyCar] --description-->
  {
  :
  [FuelUsed] --fuelType--> "Diesel"
  :
  }
```

On more detailed examination, this might turn out to be structured as:

```
[MyCar] --description-->
  {
  :
  [Engine] --description-->
    {
    :
    [FuelUsed] --fuelType--> "Diesel"
    :
    }
  :
  }
```

2.3 Descriptions based on generic entities

Another recurring idea is that entities are often variations on a generic theme. A car can be broadly classified in terms of a make and model, with variations of body colour, engine type, equipment levels and optional accessories. On a manufacturing level, what is outwardly the same vehicle type may be produced in a number of subtlely different forms as product improvements or economies are devised.

To effectively manage descriptions of such generic components, it is important to be able to refer to a common description, and then describe local variations from the base description. This idea is fundamental to CC/PP[5], one of the first protocols to be based on RDF, which uses standard descriptions and local variations to describe web client capabilities and characteristics.

In classic AI and knowledge representation research, one way this effect is achieved uses "frames". Minsky[6] describes a frame as a "remembered framework to be adapted to fit reality by changing details as necessary". More recently, building on the basic framework of RDF schemas[4], ontological work with RDF[8,9] has implemented similar structures by associating "slots" and "slot constraints" with RDF classes. Thus, an instance of an RDF class is presumed to have an associated default value for each of the defined slots. Where a specific value is not specified for a given instance, a value may be supplied by the slot description in the class definition. CC/PP[5] adopts an ad-hoc approach using a specific RDF property ('default') that exploits the regular structure of CC/PP profiles.

A drawback with this approach is that it presumes a well defined relationship between the "frame" and the "slots". If it is not known that an given property is attached to a specific type of resource (c.f. The `fuelType` example from the previous section), then there is no clear rule to state where the default value may be found. If the ontological structure is revised, the location of the slot value may change, which works against the goal of incremental modelling without full knowledge of the ontologies involved.

The framework described later allows more flexible linkage between a slot value and a prototype. Rather than depending on a fixed ontological relationship between a slot value and its corresponding definition, arbitrary linkage is allowed through the use of a framework of "proper names", based loosely on some ideas put forward by Pat Hayes[13]. These names are used to link slot definitions to slot values.

Additional structures are used here because RDF has no concept of a local name: all properties and resources are names using URIs, which are, by design, a global namespace. There is no mechanism for a URI to denote one thing in one part of an RDF model, and to denote soemthing else in another part.

Returning to the example of `[MyCar]`: Many of the details about my car may be deduced with a reasonable degree of reliability by knowing the make and model:

```
[MyCar] --isa--> [FordEscort]
```

A description of `[FordEscort]` might include the following information:

```
[FordEscort] --description-->
  {
  :
  [Engine] --description-->
    {
    :
    [FuelUsed] --fuelType--> "Petrol"
    :
    }
  :
  }
```

Information Modelling using RDF

From the above statements, it might be inferred that my car runs on petrol. But what if my car is atypical, and has a diesel engine?

According to RDF, `[FuelUsed]` is a specific resource with a given, globally unique URI. There is no way to allow the resource name `[FuelUsed]` to mean different things in the context of different instances of `[FordEscort]`. Therefore, I cannot just say:

```
[MyCar] --isa--> [FordEscort]
[     ] --description-->
  {
  :
  [Engine] --description-->
    {
    :
    [FuelUsed] --fuelType--> "Diesel"
    :
    }
  :
  }
```

because, using RDF, the resource named `[FuelUsed]` here is necessarily the same as the resource named `[FuelUsed]` in the description of `[FordEscort]`, it cannot be used to indicate that the fuel type for my car is different from the normal fuel type for a `[FordEscort]`.

2.4 Relationship to other semantic web activities

This work overlaps semantic web ontological developments such as OIL[8] and DAML-O[9], in that it defines RDF structures with inference properties, but has an emphasis on defining constructs for expressivity in RDF rather than semantic web inference and proofs. The proposals here do not depend on a full framework of first order logic.

It is not a goal of this work to provide logical capabilities beyond those provided by the ontological work: any information using the structures described here should be expressible using the ontological frameworks being developed by other groups. In due course, mechanical mapping processes should be devised.

That it is possible to write any program in low-level machine code doesn't make it a desirable or useful way to construct large programs. Similarly, this work aims to provide higher level constructs than basic RDF statements for assembling RDF models of complex systems.

As care has been taken to ensure that the entire framework is grounded in the standard RDF[3] and RDF schema[4] specifications, it should be usable with any of the generic RDF handling tools being developed. It is agnostic with respect to API, operating environment and RDF serialization format.

3. Statement sets

RDF defines a way to represent collections of statements which suffers from some practical difficulties:

- The `rdf:Bag` container class and associated containment properties make it diffcult to add new statements to a collection without knowing all of the statements already belonging to that collection.
- It is not possible to use subproperties to represent different containment relations for a single container, because each member has a different containment relation property (*rdf*:`_1`, *rdf*:`_2`, etc.).
- The standard container classes have no way to represent distributive referents within an RDF graph. The construct *rdf*:`aboutEach` is an XML serialization form that must be expanded in the corresponding RDF graph model.

3.1 A general container for sets of resources

A container class and property are defined, overcoming the above problems when used for statement sets:

- *rdfc*:`Set` is a class that may be a container for an arbitrary set of resources.
- *rdfc*:`member` is a property whose domain is an *rdfc*:`Set`, and whose range is any RDF resource type. It is used to indicate that its object is a member of its subject container.

Thus, we have:

```
[Container] --rdf:type-----> [rdfc:Set]
[         ] --rdfc:member--> [SomeResource]
[         ] --rdfc:member--> [AnotherResource]
[         ]        :
                  etc.
```

Note that, unlike the standard RDF container classes, this structure does not permit repeated instances of the same resource or value in an *rdfc*:`Set` container.

- To maximize compatibility with RDF schema[4]:
- *rdfc*:`Set` is defined to be a subclass of *rdfs*:`Container`, and
- *rdfc*:`member` is defined to be an instance of *rdfs*:`ContainerMembershipProperty`.

3.2 Containers for statement sets

Building on the generic set container described above, a statement set and membership thereof are represented by:

- *rdfc*:`StatementSet` is a subclass of *rdfc*:`Set`, and is a collection of reified RDF statements.
- *rdfc*:`quotes` is defined to be a sub-property of *rdfc*:`member`, used to indicate a reified statement that is a member of a statement set. Its domain is restricted to *rdfc*:`StatementSet`, and its range is *rdf*:`Statement`.

Thus, we have:

```
[SS] --rdf:type-----> [rdfc:StatementSet]
[  ] --rdfc:quotes--> [SomeStatement]
[  ] --rdfc:quotes--> [AnotherStatement]
[  ]        :
          etc.
```

3.3 Statement set notation

The brace notation, used in this paper for the purposes of exposition, avoids the need to invent an explicit reified statement identifier for describing statement sets, so we can write the following:

```
[StatementSet] --rdfc:quotes-->
  {
  [Sub1] --prop1--> [Obj1]
  [Sub2] --prop2--> [Obj2]
   :
  (etc.)
  }
```

The corresponding RDF must still use reified statement identifiers, but these are assumed to be automatically generated and not interesting for the purposes of constructing higher level information models.

4. Contexts

The idea of contexts was proposed by McCarthy[1,10], and some more detailed theory and applications were developed by R. V. Guha in his PhD thesis[2]. A context is characterized by the fundamental relationship 'is true in', or '`ist`', where:

```
[Statement] --ist--> [Context]
```

means that `[Statement]` is true in `[Context]`. A context may be taken to be an environment within which some statements are held to be true. Thus, to use a statement in some chain of reasoning, one must determine (or assume) that a context in which the statement 'is true in' applies to the circumstances of that reasoning.

Guha describes a context as 'defining its own language', in the sense that it establishes a framework of meaningful expressions based on first order logic. For the purposes of modelling with RDF, the first order logic is replaced by the structure of basic RDF assertions, possibly taken together with some specified RDF properties and associated set of rules that define a language and deductive framework for that context. Tim Berners-Lee describes in *The Semantic Toolbox*[12] one possible form of RDF properties and associated rules to describe first order logic. It seems useful to allow that the applicable RDF logic rules may be associated with a context.

Thus, RDF provides the basic vocabulary and grammar for representing assertions associated with contexts, and RDF schema may be used to describe ways the vocabulary is used, but rules of logical deduction associated with a context (beyond the minimal rules defined by RDF) need to be established by external means.

Another key feature of contexts is the idea of 'lifting rules' or 'lifting axioms': rules to deduce the truth of statements in one context from the truth of statements in some other context. In the absence of applicable lifting axioms, such deductions are not typically valid. I envisage that the topic of lifting between RDF contexts can be approached gradually, through the introduction of specific types of relationship between contexts, rather than by adopting a fully generalized theory of lifting.

McCarthy describes a context as an 'abstract object'. This suggests that one can make assertions *about* contexts as well as *within* contexts. He also describes a context as "a generalization of a collection of assumptions". These assumptions may concern the vocabulary and language used as well as any facts assumed; not all assumptions may be explicitly known. McCarthy also points out that, in order to work within a framework of first order logic, statements must be reified to associate them with contexts. RDF provides a mechanism for reification of statements, but it tends to be cumbersome. For the purpose of discourse the needed reification of RDF statements is implicit in the notation used.

4.1 Representation of contexts in RDF

A context represented in RDF has the following characteristics:

- It is a collection of reified statements.
- The collection and the reified statements are RDF resources.
- The statement resources are explicitly related to the collection resource by RDF properties.
- Different properties may be used to indicate different relationships between collection and statements.
- Contexts may have properties that tell us something about the statements they contain.
- Contexts may be related to other contexts in various ways.

- These characteristics are expressed by describing a *Context* as a *statement set* with some additional structural and logical properties:

- `rdfc:Context` is a subclass of `rdfc:StatementSet`, and represents a context. By inheritance this consists of a set of reified statements.

- `rdfc:asserts` is a sub-property of `rdfc:quotes`, indicating a reified statement that is a member of a context, and which is also asserted to be true in that context. Thus, it corresponds to the 'ist' (i.e. 'is true in') relation described by McCarthy and Guha. It has a domain of `rdfc:Context`, and a range of `rdf:Statement`.

Thus, we have:

```
[SomeContext] --rdf:type------> [rdfc:Context]
[           ] --rdfc:asserts--> [SomeStatement]
[           ] --rdfc:asserts--> [AnotherStatement]
[           ]        :
                    etc.
```

The `rdfc:quotes` property may be used with an `rdfc:Context` to mean that the indicated statement is included in the context, but is not asserted by the context to be true. This is needed if the context makes statements about other statements that are not themselves held to be true; e.g. "Graham says 'the sheep is pink'" might be represented as:

```
[Context] --rdf:type------> [rdfc:Context]
[       ]
[       ] --rdfc:asserts-->
[       ]     {
[       ]     [Graham] --says--> [SheepIsPink]
[       ]     }
[       ]
[       ] --rdfc:quotes-->
[       ]     {
[       ]     SheepIsPink: [The sheep] --colour--> "pink"
[       ]     }
```

> ***NOTE***: it might be questioned whether non-asserted statements actually belong in a context. One of the motivations for this work is to have a way to collect together related parts of an RDF graph, so this has been adopted as a helpful idea. Also, given the way that contexts are constructed from statement sets, it is a natural and easy structure to adopt.

4.1.1 Asserting statements in different contexts

Statements may be associated with multiple contexts, with their various assertions conveying different kinds of information. Suppose that I make the following statements:

```
[MyCar] --fuelEconomy-----> "40mpg"
```

```
[MyCar] --engineCapacity--> "1600cc"
```

The first statement may be something that I believe. The second statement may also be something that I must assure for the purposes of obtaining insurance. Contexts corresponding to statements I believe and statements I assure can be constructed thus:

```
[MyBeliefs] --rdfc:asserts-->
     {
     [MyCar] --fuelEconomy-----> "40mpg"
     [MyCar] --engineCapacity--> "1600cc"
      :
     }
```

and

```
[MyAssurances] --rdfc:asserts-->
     {
     [MyCar] --engineCapacity--> "1600cc"
      :
     }
```

Note that a statement may be true in any number of different contexts. Thus, the `rdfc:asserts` property may be applied any number of times using a given statement. (Naturally, any number of statements may be asserted by a given context: this grouping of statements into larger units is one of the motivations for these proposals.)

4.2 Using contexts: logical constructs

4.2.1 Contexts as containers

An obvious use of a context is as a container for some collection of statements. An RDF document may contain a number of statements. Assertions applied to the document might be taken to apply to each of the statements contained in the document.

For example, a signature applied to a document creates some kind of assurance about the content of that document. This assurance would reasonably be considered to apply to each of the statements within the document. To apply such assurances individually to each statement seems cumbersome, especially given the fine-grained nature of individual RDF statements. Thus, we have one application for contexts that is to facilitate making assertions about collections of statements.

A simple approach that has been suggested would be to allow statements about a collection to be applied automatically to its members. This allows no way to distinguish statements about the collection itself from statements about its members. Suppose I use a context `[MyWebSite]` to contain a set of statements about my web site, and attach a property that says that certain people are authorized to make changes. Does this property apply to the collection of statements about the web site, or does it apply individually to each of the contained statements?

I propose a level of indirection that makes explicit the distribution of assertions over contained statements. It follows the idea of 'interpretation properties' described by Tim Berners-Lee[12]. In this case, the proposed 'interpretation property' is one that explicitly applies assertions to all of the contained statements:

```
[CarInsurance] --rdfc:asserts----->
[            ]     {
[            ]     [Car] --registration----> "ABC123X"
[            ]     [Car] --engineCapacity--> "1600cc"
[            ]      :
[            ]     }
[            ] --rdfc:applyToAll--> [_stmt]

[_stmt] --assuredBy--> [Car owner]
[_stmt] --assuredBy--> [Car registration]
```

Here, *rdfc*:applyToAll indicates quantification over statements within the context, and [_stmt] is a place-holder for such statements. For each statement contained within the context, and for each statement involving the place-holder resource [_stmt], a corresponding statement is asserted. In the above example, the statements of assurance by both the car owner and the car registration document are applied to the statements about registration mark and engine capacity that are contained within the context:

```
S1: [Car] --registration----> "ABC123X"
S2: [Car] --engineCapacity--> "1600cc"
     :

    [S1] --assuredBy--> [Car owner]
    [S1] --assuredBy--> [Car registration]
    [S2] --assuredBy--> [Car owner]
    [S2] --assuredBy--> [Car registration]
     :
```

Thus, *rdfc*:applyToAll indicates that all properties applied to the object resource are to be applied individually to each statement that holds in the subject context. The resource [_stmt] is a local place-holder of implied type *rdf*:Statement, identified by an arbitrary URI-reference.

> ***NOTE***: *rdfc*:applyToAll is presented here as a one-off case for use with contexts. The idea can be generalized to apply to an arbitrary container class and member type[20].

> ***NOTE***: This is an alternative way to model statements about sets of statements than that described in the RDF Model and Syntax[3], section 4.2. A significant difference is that this approach represents the distributed referent structure within the RDF graph, and therefore can be applied to contextual information possibly not otherwise available when the RDF is de-serialized.

4.2.2 Contexts as statements

A motivation for this work has been to find ways of combining basic RDF statements into higher-level statements that can be treated as basic statements. A context can be treated as such a composite statement.

The reification of a statement is a structure that stands irrespective of whether the statement is true, and a statement set is a collection of reified statements that does not consider the truth of them. A reified statement is *asserted* in a context if it is considered to be true in that context. Similarly a context contains statements that are true under some circumstance, irrespective of whether that cirumstance applies. A context may be *asserted* in some other context, meaning that statements true that context are also regarded as true in the surrounding context.

In this respect, a context may assume some of the attributes of a statement: it may be regarded as true or false (or unknown) in the sense that the statements that it embodies are all true, or otherwise. Its truth (or otherwise) may depend upon the truth of some other context. The kinds of metalogical operations that can be applied to statements (conjunction, implication, negation, etc.) could be usefully applied to contexts, treating a context as a kind of composite statement. It can even make sense to combine statements and contexts in metalogical expressions.

These ideas for interchangeability of contexts and RDF statements can be expressed by creating a unifying superclass for them. Defining one as a subclass of the other is not satisfactory because each has properties not shared by the other. (A statement has a very specific structure and relationship to the RDF model; a context can be used in ways not applicable to a statement.) The class `rdfc:Assertable` is defined to be a superclass of both `rdf:Statement` and `rdfc:Context`, e.g. with RDF scheme statements of the form:

```
[rdfc:Assertable] ---rdf:type---------> [rdfs:Class]
[rdf:Statement] -----rdfs:subClassOf--> [rdfc:Assertable]
[rdf:StatementSet] --rdfs:subClassOf--> [rdfc:Assertable]
[rdfc:Context] ------rdfs:subClassOf--> [rdfc:Assertable]
```

Where a statement, statement set *or* a context may be used as the domain or range of some property, this can be expressed in a schema by the `rdfc:Assertable` class. Thus, the ranges of `rdfc:quotes` and `rdfc:asserts` are revised, relaxing the initially specified object type of `rdfc:Statement` to allow statements of the form:

```
[<rdfc:Context>] --rdfc:quotes---> [<rdfc:Assertable>]
[<rdfc:Context>] --rdfc:asserts--> [<rdfc:Assertable>]
```

where `rdfc:Assertable` can be a statement, statement set or context.

4.2.3 Contexts as resources

One of the difficulties I have observed in trying to use RDF to model real world situations is that it is very difficult to construct a model of meaningful complexity without becoming overwhelmed by the details of the RDF graph. The idea of contexts as containers leads quite naturally to a treatment of contexts as first class resources in their own right, representing the collection of statements describing some physical-world object. Further, the use of contexts as statements allows the statements contained within a context themselves to be contexts.

Using these ideas, and the extra RDF properties described above, one can construct models in terms of high-level resource and property concepts. The validity or consequences of these high-level relationships is discovered by looking inside the corresponding contexts to find the component parts and the local interactions between them. Thus, one of the goals of this work is realized, allowing high-level models of complex systems to be constructed without knowledge of the more detailed structures involved. Consider the description of a car:

```
[MyCar] --manufacturer--> [Ford]
[     ] --model---------> [Escort]
[     ] --rdfc:asserts-->
            {
            [Engine] --model--> [CVH]
            [      ] --rdfc:asserts-->
                         {
                         [Fuel] --fuelType--> "petrol"
                         [Spec] --capacity--> "1600cc"
                           :
                         }
            [Body] --style--> "Hatchback"
            [    ] --rdfc:asserts-->
                       {
                       [Shell] --material--> "steel"
                       [     ] --weight----> "200Kg"
                       [Doors] --count-----> "5"
                        :
                       }
             :
            }
```

This example illustrates how a description of a complex object or system can be built up component-wise from contexts describing the various components. At each level, a context is used as a resource that stands for the entity being described. Statements that are clearly about the resource as a whole can be attached to the resource directly (e.g. `manufacturer` above), while statements about properties of component parts can be associated with the appropriate part in a way that is not invalidated if the internal structure needs to change. For example, consider `fuelType`, which is associated above with the engine type. Suppose the engine description is restructured to consist of mechanical, fuel and electrical subsystems:

```
[Engine] --model--> [CVH]
[      ] --rdfc:asserts-->
             {
             [MechSubSystem] --rdfc:asserts-->
                {
                [Cylinders] --count-----> "4"
                [         ] --capacity--> "400cc"
                [Valves] -----count-----> "8"
                [CamShaft] ---location--> "Overhead"
                [        ] ---drive-----> "Toothed belt"
                 :
                }
             [FuelSubSystem] --rdfc:asserts-->
                {
                [Fuel] --fuelType----> "petrol"
                [    ] --systemType--> "injection"
                 :
                }
             [ElecSubSystem] --rdfc:asserts-->
                {
                [Ignition] --systemType--> "electronic"
                  :
                }
               :
             }
```

For the purposes of refuelling, the assertion '`[Fuel] --fuelType--> "petrol"`' is sufficient. This statement can be inferred in exactly that form, even though the internal structure of the description of the car has been changed. I believe that this ability to describe and infer certain facts independently of the detailed ontological structure is crucially important to the practical construction of models describing complex systems. Some initial exercises have suggested that using contexts in this way very greatly improves our ability to design models of real world objects.

5. Generic entities

Section 2.3 has introduced the motivation for generic entities, and the difficulties of using ontologically based frame descriptions. Here, I describe a system of "proper naming" for RDF that is loosely based on some ideas put forward by Pat Hayes[13] on the RDF logic mailing list[14], and illustrate its use to construct generic models.

5.1 URIs and Proper Names

RDF uses URIs and URI-references, defined by RFC 2396[16], to name resources. URIs are generally defined to be globally unique: a URI that appears in different places is always expected to indicate the same "resource" (which does not necessarily mean the same data, but that is not pertinent to the present discussion). URI references have a relative form, but RFC2396 is quite clear that this is a lexical convention and that the relative form should map to an absolute (globally unique) form depending on the lexical environment in which it occurs. Typically, the URI of a containing document is used to construct the global URI corresponding to a relative URI.

Pat Hayes observes[13] that much human discourse does not depend on global names to identify the topics being discussed; rather, proper names are used that have accepted meaning within some context of discourse. For example, in a discussion that alludes to American geography, "Boston" is typically understood to be a city in the US state of Massachusetts, but in other contexts it might mean something else. Here, the meaning of "Boston" is understood through a binding indicated by social context.

In logic, mathematics and computer programming, "local names" are often used to stand for a concept within some defined environment. The fact that the local name is not bound to some global meaning gives rise to its expressive power. Formal parameters of functions and procedures are responsible for much of the expressive power of notations used in logic, mathematics and computer programming. There has been discussion about how to introduce quantification into RDF to extend its expressive power (e.g. Tim Berners-Lee's "Semantic toolbox"[11], and many others); there is no question that some such mechanism is required.

Using the social concept of proper names for local variables may seem strange; I suggest that this is an appropriate unifying of mechanisms, allowing a continuum of locally bound names (in expressing purely logical constructs) to socially bound names (in descriptions of ideas with social scope). Specifically, ideas that start out as being socially defined may be carried into contexts where the definition is logically bound. For example, the name "Boston" may be introduced in a social context with purely social binding, but may be carried over into a context of air transport where it is ultimately bound into a mathematical framework of spatial coordinates that ultimately define where an airplane must land.

5.2 Proper names in RDF

This section proposes an experimental mechanism for expressing proper names within standard RDF. A proper name may be an arbitrary string of characters. International proper names, required to support social naming conventions, are accommodated by allowing characters from the UCS-4 (Unicode)[22] repertoire.

One or more proper names may be associated with any RDF resource by the *rdfn*:properName property. The domain of *rdfn*:properName is any RDF resource, and its range is a literal string containing the proper name value:

```
[MyCar] --rdfn:properName--> "my car"
[     ] --rdfn:properName--> "ABC123X
[     ] --rdfn:properName--> "Emily
```

These three properName properties indicate different names by which [MyCar] may be referenced: in informal conversation, I talk about “my car”, in formal communications (e.g. insurance application) I may refer to it by a registration mark “ABC123X”, and my wife might refer to it in an anthropomorphic style as “Emily”. Clearly, someone else might use the name “my car” to refer to a completely different vehicle. In general, a proper name string is not bound to unique entity outside of a given context of usage.

A particular context in which a proper name applies may be indicated by including the *rdfn*:`propername` statement in that context.

> ***NOTE***: the resource `[MyCar]` above is an RDF resource that is generally assumed to have a globally unique identifier in the form of a URI-reference. It may be that the URI-reference is a generated value like a `uuid:` or `cid:` URI with no obvious relationship to any specific entity. There has also been some discussion in the W3C RDF interest group[15] about anonymous resources (without names, or having names that are explicitly not globally unique URIs). This is a matter for continuing debate. In any case, the *rdfn*:`properName` property associates the indicated name string with an RDF resource, however identified.

An inference rule associated with *rdfn*:`properName` tells us that two resources with the same proper name value are equivalent. That is, any property true of one such resource may be taken to be true of the other. Thus, proper names provide a logical framework for asserting the equivalence of two resources.

> ***NOTE***: this description skirts an issue of the relationship between resources and resource identifiers. One school of thought asserts that the relationship between resources and identifiers is *1:1-onto*; i.e. each resource identifier indicates exactly one resource, and each resource has exactly one resource identifier. Another view allows multiple resource identifiers to indicate the same resource. This debate is avoided by talking about *equivalence* of resources while remaining agnostic on the issue of *sameness*.

5.3 Using proper names in generic descriptions

Suppose my car is a Ford Escort. From this assertion, one can immediately infer a range of facts about my car as “Ford Escort” is an established kind of car. Proper names allow us to construct RDF descriptions that capture this kind of prototype-based inference.

```
[FordEscort] --rdfc:asserts-->
  {
  [FordEscortBody] --rdfn:properName--> "Ford Escort body"
  [              ] --rdfc:asserts-->
    {
    (details about car body)
    }
  [FordEscortEngine] --rdfn:properName--> "Ford Escort engine"
  [                ] --rdfc:asserts-->
```

```
        {
        (details about car engine)
        }
      :
    }

  [MyCar] --rdfc:asserts--> [FordEscort]
    {
    [MyCarBody] --rdfn:properName--> "Ford Escort body"
    [         ] --rdfc:asserts-->
      {
      [BodyPaint] --colour--> "Red"
        :
      }
      :
    }
```

The above examples show a context `[FordEscort]` that describes a generic Ford Escort car, and another context that describes my car as a kind of Ford Escort. The generic Ford Escort is defined in terms of specific RDF resources (`[FordEscortBody]`, `[FordEscortEngine]`, etc.) that stand for components of the generic car type. Thus, of itself, the resource `[FordEscortBody]` does not stand for a specific car body, but expresses properties common to all Ford Escort car bodies.

The description of `[MyCar]` asserts all of the statements that are true of a generic Ford Escort. It also asserts that `[MyCarBody]` has a proper name the same as `[FordEscortBody]`. The inference rules associated with *`rdfn`*`:properName` then allow us to deduce that, *in the context of `[MyCar]`*, all statements about `[FordEscortBody]` are also applicable to `[MyCarBody]`. Additional statements about `[MyCarBody]`, such as it having red paint, also apply in this context.

6. Conclusions and further work

RDF mechanisms based on statement sets and contexts have been described that allow descriptions of complex systems to be constructed without necessarily having detailed knowledge of the ontological structure of the system components used. I believe this is a key enabler for the practical construction of complex system models in RDF.

An additional RDF mechanism has been described, based on the idea of a "proper name", that allows descriptions to be based on prototypes, in a fashion similar to the frame based descriptions proposed by AI researchers, but not depending on detailed knowledge of the ontological structures involved.

Articulated visions for the Semantic Web require that anyone must be able to say anything about anything[21]. It is unreasonable to expect everyone to adopt exactly the same ontological structure for making statements about an entity; apart from political and perceptual differences, that approach cannot scale. This leads to my assertion that practical modelling of complex systems requires statements that can stand independently of finer ontological details. This is not a dismissal of ontological structures; work on onological frameworks such as OIL[8] and DAML-O[9] is needed to underpin verification of web-based information. In due course, I would expect a theory to emerge that relates descriptions based on incomplete ontologies to more rigorously complete frameworks. I view basic RDF as a kind of "assembly language" for information modelling, and see this use of contexts and proper naming as a parallel to procedures and formal parameters in programming languages, used to aid the construction of complex object descriptions without adding new formal capabilities.

The constructs presented here are being used in the following ongoing experimental developments:

- A graphical tool for RDF modelling[18].
- An experimental RDF-driven expert shell[19].

We also aim to develop mechanisms for trust modelling and inference; modelling social trust structures and overcoming the brittleness of purely cryptographically based approaches to trust in e-commerce, etc. Another area for investigation is the design of mechanisms for managing non monoticic reasoning, and other logical extensions of contexts [20].

In messages to the RDF interest group, Dan Brickley has proposed[17] an alternative approach to labelling anonymous RDF resources; i.e. resources whose formal URI or URI reference is unknown. The outcome of these discussions may affect the exact form of naming preferred.

7. Acknowledgements

This note is based in part on an earlier document[20], for which I received valuable feedback from Brian McBride, Dan Brickley, Jan Grant and others from HP Labs and ILRT in Bristol, UK. The ideas have also benefited from ongoing discussions in the W3C RDF interest group email discussion list[15] and RDF logic email discussion list[14], and particularly from thoughts offered by Jonathan Borden, Sergey Melnik, Seth Russell and Wolfram Conen. I extend my thanks to these, and everyone else who has been so generous with their thoughts in the RDF discussion forums.

8. References

[1] John McCarthy,
Notes on Formalizing Context,
Computer Science Department, Stanford University.
<http://www-formal.stanford.edu/>

[2] Ramanathan V. Guha,
Contexts: A Formalization and Some Applications,
Stanford PhD Thesis, 1991.
<http://www-formal.stanford.edu/>

[3] Ora Lassila, Ralph R Swick,
Resource Description Framework (RDF) Model and Syntax Specification,
W3C Recommendation, 22 February 1999.
<http://www.w3.org/TR/REC-rdf-syntax>

[4] Dan Brickley, R. V. Guha,
Resource Description Framework (RDF) Schema Specification 1.0,
W3C Candidate Recommendation, 27 March 2000.
<http://www.w3.org/TR/rdf-schema>

[5] CC/PP Working Group
<http://www.w3.org/Mobile/CCPP/>

[6] Marvin Minsky,
A Framework for Representing Knowledge,
1975.
(This paper contained in [7]).

[7] Ronald J. Brachman and Hector J Levesque,
Readings in Knowledge Representation,
Morgan Kaufman Publishers, Inc., 1985.
ISBN 0-934613-01-X.

[8] *Ontology Inference Layer (OIL)*
<http://www.ontoknowledge.org/oil/>

[9] *DARPA Agent Markup Language Ontology (DAML-ONT)*
<http://www.daml.org/2000/10/daml-ont.html>

[10] John McCarthy,
Generality in Artificial Intelligence,
Communications of the ACM, Vol 30, December 1987.

[11] Tim Berners-Lee,
The Semantic Toolbox,
Personal note, 24 May 1999.
<http://www.w3.org/DesignIssues/Toolbox.html>

[12] Tim Berners-Lee,
Interpretation properties,
Personal note, 29 Feb 2000.
<http://www.w3.org/DesignIssues/InterpretationProperties.html>

[13] Pat Hayes,
names, URIs and ontologies
<http://lists.w3.org/Archives/Public/www-rdf-logic/2000Oct/0112.html>
(Thoughts about names in RDF, posted to RDF-logic mailing list[14])

[14] <www-rdf-logic@w3.org> *Mail Archives*
<http://lists.w3.org/Archives/Public/www-rdf-logic/>
(RDF logic discussion email distribution list)

[15] <www-rdf-interest@w3.org> *Mail Archives*
<http://lists.w3.org/Archives/Public/www-rdf-interest/>
(Discussion archive for the RDF Interest Group)

[16] T. Berners-Lee, R. Fielding, L. Masinter,
Uniform Resource Identifiers (URI): Generic Syntax,
Internet Engineering Task Force, RFC 2396, August 1998.

[17] Dan Brickley,
Anonymous resource names -versus- variables,
<http://lists.w3.org/Archives/Public/www-rdf-interest/2000May/0032.html>
(Thoughts about resources with unknown URI.)

[18] Craig Pugsley
A graphical modeller for RDF
[[[Software to be published as open source on web]]]

[19] Craig Pugsley, Graham Klyne,
An RDF-Driven, Web-informed Expert System for Task Planning,
[[[Work in progress, November 2000]]]

[20] Graham Klyne,
Contexts for RDF Information Modelling,
Discussion document, in progress, 18 October 2000.

[21] Tim Berners-Lee,
What the Semantic Web can represent,
Personal note, 17 Sept 1998.
<http://www.w3.org/DesignIssues/RDFnot.html>
(See section: *The Semantic Web and Entity-Relationship models*)

[22] The Unicode Consortium,
The Unicode Standard, version 2.0,
Addison Wesley, 1998.
ISBN 0-201-48345-9

[23] Pierre-Antoine Champin,
RDF Tutorial,
28 June 2000.
<http://www710.univ-lyon1.fr/~champin/rdf-tutorial/>

[24] Dan Brickley,
RDF Interest Group - Issue Tracking
<http://www.w3.org/2000/03/rdf-tracking/>

[25] Tim Berners-Lee,
Notation 3 - Ideas about Web architecture
<http://www.w3.org/DesignIssues/Notation3.html>