

# Experience in using RDF in Agent-mediated Knowledge Architectures

Kit Hui, Stuart Chalmers, Peter Gray, Alun Preece

University of Aberdeen, Computing Science Department  
 Aberdeen AB24 3UE, Scotland  
 Phone: +44 1224 272291; FAX: +44 1224 273422  
 Email: {khui|schalmer|pgray|apreece}@csd.abdn.ac.uk

## Abstract

We report on experience with using RDF to provide a rich content language for use with FIPA agent toolkits, and on RDFS as a metadata language. We emphasise their utility for programmers working in agent applications and their value in Agent-Oriented Software Engineering. Agent applications covered include Intelligent Information Agents, and agents forming Virtual Organisations. We believe our experience vindicates more direct use of RDF, including use of RDF triples, in programming knowledge architectures for a variety of applications.

## Introduction

Resource Description Framework (RDF) and the associated RDF Schema (RDFS)<sup>1</sup> were introduced by W3C as a portable framework for passing structured data and its associated metadata over the web. To date, much of the work on RDF and RDFS and on higher level languages such as DAML has concentrated on its role in supporting ontologies, rather than on how it helps a programmer working with contemporary agent architectures using Java toolkits. In this paper we wish to distil experience from using RDF in two very different agent-based systems (AKT and Conoise) where we needed to use an agent platform (such as JADE) to exchange rich content.

We believe that the virtues of RDFS as a sparse extensible metalanguage for describing data moved across the web have been lost sight of in the rush to explore more elaborate languages such as DAML and OWL. One should remember that these languages are themselves layered on RDF/RDFS. Also, languages such as OWL are evolving rapidly, while RDFS has remained surprisingly stable. Lastly, there is now very good support for RDF/RDFS in Java-based class libraries which can easily be integrated with FIPA-compliant Java-based agent toolkits, such as JADE. Thus from the viewpoint of Software Architectures there is much to commend the sparse elegance and stability of RDF/RDFS in a fast-changing world. Indeed,

Copyright © 2003, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

<sup>1</sup><http://www.w3.org/TR/REC-rdf-syntax>

we now believe that it is timely for FIPA agent languages to move from a LISP-based syntax to an alternative XML version, and to adopt RDF as a standard content language.

A particular advantage of RDFS is its use from Java as a convenient accessible metalanguage giving the types of structured data, including possible specialisations. When data is coming across the web in many forms it is essential to have this type information easily available in memory. Thus the programmer does not have to invent their own form of object class for this metadata. They also have Java methods available to populate these classes from input in a widely used interchange format. Contrast this with the intricate methods provided for accessing some fairly basic metadata in JDBC.

We have used this metadata for describing *schemas* (ontologies), *quantified constraints* for planning applications, and *structured data* for use in modelling Virtual Organisations. We describe these in more detail below.

When working with RDF, we have found it useful to convert it into *RDF triples* held in a Java table. This is equivalent to the usual pointer-based graph structure, but easier to search and more convenient when merging in extra RDF triples. Searching the table is easier than trying to work directly with an XML tree structure, because equivalent XML abbreviations are reduced to a single form, like a canonical form.

Besides using Java, we have developed in-memory Prolog term structures to hold these triples, which are more convenient for Prolog pattern matching and reasoning. This is crucial for semantic web applications. Thus we are not tied solely to handling triples in Java.

Lastly, we have experience of using this system with several different transport protocols. For web sources, we find the *HTTP protocol* very useful, particularly where there are firewalls. For remote agents running on different platforms we find *Linda* (Gelernter & Carriero 1992) very useful in conjunction with remote procedure calls, while for Java platforms we may use *Servlet* technology. Because of the well established use of XML, support for these protocols is widely available and tested, which is another plus for RDF/RDFS.

## The RDF Data Model

RDF is not unlike the Entity-Relational data model in its use of Entity identifier as *Subject*, and Property or Relationship names as *Predicate* in RDF triples. However, it also includes features of object data models (such as OQL) in its use of object identifiers and subclasses.

Although it looks simple, it has all the essential features for mapping other data models or layering extra details, as intended in its design. We have found it easy to map data from our existing FDM (an early semantic data model) and also quantified constraints which are formulae of logic expressed in this model. One very satisfying feature of our constraint interchange format in RDF (CIF) is that the tags used make a clean separation between information about *logical formulae* with the usual connectives, and information about *expressions* denoting objects in the data model. Effectively CIF gives another layer with richer semantic information, but it is able to use all the processing convenience of RDF.

Expressions in our CIF language refer to facts about entities, their subtypes, attributes and relationships, which of course can be expressed in RDF. Note also that this model abstracts over relational storage, flat files and object-oriented storage, following the principle of data independence. Thus it does not tie one to any particular system, such as Oracle or P/FDM. This is a great advantage to the programmer. Thus we stress the value of using RDFS in place of data structures that tie the programmer to a particular storage schema. The mapping to a particular knowledge source or data source can then take place separately through a wrapper. This makes it very much easier to integrate data from different sources, as is often required over the Web.

We illustrate these programming principles below in two rather different applications. Firstly we consider the fusion of constraint information from different sources on an intranet in the KRAFT project. This uses intelligent information agents as mediators and facilitators. It originally used a textual version of Prolog term structures for interchange. We are now reworking it to use RDF and XML, and to use RDFS in place of P/FDM specific constructs. This is being used in the AKT (Advanced Knowledge Technologies) collaborative project, so that we can fuse data from other partners.

Our second example uses agents that make bids and that come together to form Virtual Organisations. This also uses constraints, but more as a way to give additional desires to BDI agents, whilst allowing them autonomy in how they cope with other conflicting goals and desires.

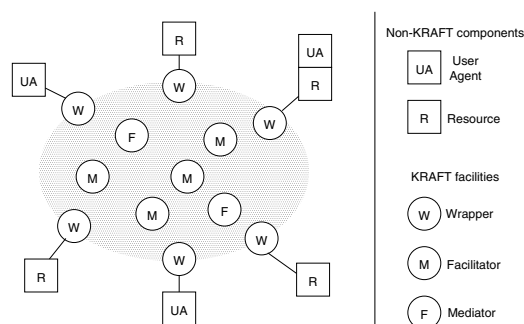


Figure 1: This figure shows a conceptual view of the KRAFT architecture. KRAFT components are round in shape while non-KRAFT ones are marked as squares. The grey area represents the KRAFT domain where a uniform language and communication protocol is respected.

## Agent Architectures for Information Integration and Fusion

The KRAFT<sup>2</sup> system (Preece *et al.* 2001) employs an agent-based architecture inspired by the Knowledge Sharing Effort which has proved to be an effective approach to developing distributed information systems. The basic philosophy of the architecture design is to define a KRAFT domain where certain communication protocols and languages must be respected (figure 1). Within this domain, agents are assumed to cooperate and connections are made dynamically<sup>3</sup>. KRAFT recognises constraints as abstract mobile knowledge which can be extracted, transported, transformed and processed by software components. We use the constraint formalism as a domain-independent framework to represent application problems as constraint satisfaction problems (CSPs). The application domain is modelled by a database *schema* and domain knowledge is captured as *constraints*. Constraints distributed in resources form a library of shareable and reusable knowledge blocks which can be combined to compose problem specifications.

Having a semantic data model extended with constraints and mapped into an open interchange format supports a range of applications in which information needs to be moved across a network with rich meta-level information describing how the information can be used. An example application common in business-to-business e-commerce involves the composition of a package product from components selected from multiple vendors' catalogues. There are various kinds of constraints which must be aggregated and solved over the available component instances: constraints representing customer requirements, constraints represent-

<sup>2</sup>Knowledge Reuse And Fusion/Transformation

<sup>3</sup>Hence the absence of explicit connections in the grey area in figure 1.

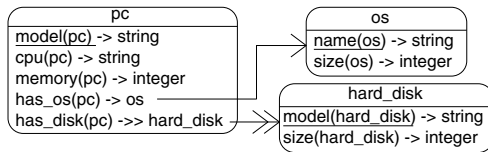


Figure 2: This schema shows three entity classes. The single arrow means that each `pc` may have only one `os` installed. A double arrow means that a `pc` can have multiple `hard-disk`.

ing rules for what constitutes an acceptable package, and constraints representing restrictions on the use of particular components.

In this last case, the ability to store constraints together with data in the P/FDM database system<sup>4</sup> allows instructions, which we called “small print constraints”, to be attached to the class descriptor for data objects in a product catalogue database. When a data object is retrieved, these attached instructions must also be extracted to ensure that the data is properly used. Thus the attached constraint becomes mobile knowledge which is transported, transformed and processed in a distributed environment. This approach differs from a conventional distributed database system where only database queries and data objects are shipped.

### Modelling a Domain in RDF Schema

In the original KRAFT system, we model a domain by a database schema, using the functional data model. The schema effectively serves as an ontology that captures knowledge of classes, attributes and subclass relationships in the domain. The following is part of a schema showing the `pc` and `os` classes and the `memory` property in an application domain where components are put together to configure a workable PC. The complete ER model is shown in figure 2:

```
declare os ->> entity
...
declare pc ->> entity
declare memory(pc) -> integer
declare has_os(pc) -> os
...
```

The functional data model is an extended ER model which can be easily mapped into the RDF schema specification. A mapping program reads meta-data from the database and *generates* the corresponding RDF schema, making this knowledge web-accessible. The following RDFS fragment refers to the schema above:

```
<rdf:Class rdf:ID="pc">
  <rdf:subClassOf rdf:resource=
    "http://www.w3.org/2000/01/rdf-schema#
    Resource"/>
</rdf:Class>
```

<sup>4</sup><http://www.csd.abdn.ac.uk/~pfdm/>

```
<rdf:Class rdf:ID="os">
  <rdf:subClassOf rdf:resource=
    "http://www.w3.org/2000/01/rdf-schema#
    Resource"/>
</rdf:Class>

<rdf:Property rdf:ID="has_os">
  <rdf:domain rdf:resource="#pc"/>
  <rdf:range rdf:resource="#os"/>
</rdf:Property>
```

Mapping a P/FDM schema into RDFS has the advantage of making the domain model available to RDFS-ready software. On the other hand, some semantic information is lost. The cardinality of each attribute, for example, is not expressed in the RDF schema. Information on the *key* of each entity class is also omitted. However, this information can easily be added to other metadata held in the `cif:entmet` layer (Gray, Hui, & Preece 2001)

Briefly, `entmet` is a class of metaobjects whose instances correspond one-to-one with entity classes, and whose property values give metadata such as class name, superclass metaobject and RDF URI. One instance is the object in figure 4 whose ID is `entmet.pc`. This gains us extensibility by adding extra properties, such as *key*, to the metaobjects in `entmet` or in `propmet` (for properties). There is, of course, some redundancy, as where we record subclass information both in `entmet` and by using `rdf:subClassOf`. However, it is kept consistent, and provides a clean layering of extra semantic information to be used by enabled reasoners.

### Capturing Domain Knowledge in RDF

Domain knowledge in KRAFT is captured as integrity constraints expressed against the database schema, using the constraint language CoLan (Bassiliades & Gray 1994). CoLan is based on first-order logic and has proved to be expressive enough to represent complex constraints (Fiddian *et al.* 1999). CoLan constraints have evolved from the usual database state restrictors into mobile problem specifications (Gray *et al.* 1999). The following is an example of a *design constraint* saying that “the size of a hard disk must be big enough to accommodate the chosen operating system in every PC”:

```
constrain each p in pc
  to have size(has_os(p)) =< size(has_disk(p))
```

User requirements are also captured as constraints. The following constraint specifies that the configured PC must use a pentium 4 CPU:

```
constrain each p in pc
  to have cpu(p)="pentium 4"
```

In practice, the human-readable CoLan constraints are compiled into an intermediate format, called *Constraint Interchange Format* (CIF). CIF expressions are syntactically Prolog terms, which are easier to process by software components.

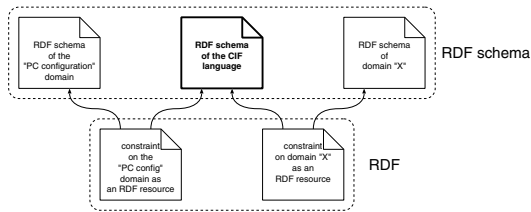


Figure 3: Constraints in RDF make references to the CIF language definition and domain models in RDF schema.

To make CIF portable, we encode CIF constraints into RDF by defining an RDF schema for the CIF language that is layered cleanly on top of RDF, serving as a meta-schema (Gray, Hui, & Preece 2001). A constraint encoded in RDF makes explicit references to classes defined in the *domain model* as well as the *CIF language definition* in RDF schema (figure 3). The RDF fragment in figure 4 shows the `cif` namespace definition and references to the CIF language RDF schema resource. In this example, the variable with name `uevar1` is restricted to be an instance of the entity class `pc` which is defined by the domain model in "`http://www.csd.abdn.ac.uk/~schalmer/schema/pc_schema#pc`". References to the domain model can be found as value of the `cif:entmet_rdfname` property.

This approach makes no change to the existing RDF and RDF schema specifications. As constraints are now represented as resources, RDF statements can also be made about the constraints themselves. A detail discussion of the CIF encoding in RDF is presented in (Gray, Hui, & Preece 2001).

### Using Knowledge Encoded in RDF

Knowledge encoded in RDF is a web-accessible resource which can be utilised by knowledge processing components. While the RDF model is simple and may not be expressive enough to represent complex and sophisticated knowledge, higher-level layers can be nicely built above RDF to incorporate the richer semantics. This is demonstrated in our RDF encoding of the CIF constraint language.

RDF provides a uniform framework for the representation of knowledge and meta knowledge. Although it is simple enough to be processed by different software platforms, we believe that the RDF-encoded format should be used for transportation only, not for direct reasoning purposes. Instead, inference and reasoning are better done on the RDF triples; or on Prolog term structures which are mapped from the triples for processing convenience.<sup>5</sup> RDF triples may be representing

<sup>5</sup>We use the Prolog "Pillow" library (<http://clip.dia.fi.upm.es/Software/pillow/pillow.html>) for XML parsing, with an RDF parser sitting on top of Pillow to map XML trees into RDF triples.

```
<rdf:RDF
...
xmlns:cif="http://www.csd.abdn.ac.uk/~khui/
akt/cif/cif-rdfs.xml#"

<cif:impliesconstr rdf:ID="eg1">
  <cif:qvar>
    <cif:setmem>
      <cif:setmem_var>
        <cif:variable rdf:ID="uevar1">
          <cif:varname>uevar1</cif:varname>
        </cif:variable>
      </cif:setmem_var>
    </cif:setmem_set>
  <cif:entset>
    <cif:entset_entclass>
      <cif:entmet rdf:ID="entmet_pc">
        <cif:entmet_rdfname>
          http://www.csd.abdn.ac.uk/
            ~schalmer/schema/pc_schema#pc
        </cif:entmet_rdfname>
      </cif:entmet>
    </cif:entset_entclass>
  </cif:entset>
</cif:setmem_set>
</cif:setmem>
</cif:qvar>
...
```

Figure 4: This fragment of a constraint encoded in RDF shows explicit references to the CIF language definition and the domain model in RDF schema.

knowledge in a small granularity but they can be easily assembled into knowledge chunks of a convenient size for processing purposes. Thus the triple is a format which can be readily used in reasoning or as a stepping stone to other formats. These triples can conveniently be created in memory by Jena (figure 9).

In this section, we show how *constraints*, *meta knowledge* and *domain models* encoded in RDF can be used by knowledge processing components.

#### • Converting Constraints between CIF and RDF

An RDF schema provides knowledge on the class hierarchy and class properties with type information. For example, as we know that the CIF `constraint` class has three subclasses `impliesconstr`, `unquantifiedconstraint` and `existsconstr`, we have to try all these subclasses when we try to parse an RDF statement as a `constraint` resource. Similarly when we try to parse a property of a resource, we know what *class* of resource is valid.

The RDF schema of the CIF language alone does not give us enough information to map an RDF-encoded CIF constraint into its Prolog representation (and vice versa). There are two pieces of knowledge missing:

– What is the Prolog term structure that corresponds to a certain class in the RDF schema (of

the CIF language)?

- Given the Prolog representation of a resource of a certain class, how can we find each property of the resource (as Prolog terms)?

This missing knowledge should not be specified in the RDF schema as it cares only about the semantics of a constraint resource but not the way that it is represented in Prolog.

To solve this problem, we represent this knowledge as mapping rules between a Prolog term structure and its corresponding RDF class (in the CIF language definition RDF schema). Once the missing knowledge is provided, the mapping process between CIF and its RDF encoding is totally driven by the RDF schema of the CIF language. Once again, structuring the task around RDF has made the programming job easier.

- Knowledge Reuse by Constraint Fusion

Declarative constraints stored as self-contained knowledge objects in a distributed system form a shared library of building blocks. The key to reusing this knowledge is the process of *constraint fusion*, which dynamically combines the semantic content to compose problem specification instances. While a single piece of constraint may not contain enough information to solve a CSP, we hope that by combining constraints together, their total value is enhanced, thus making the problem solvable.

The domain independent constraint fusion engine in KRAFT looks for any potential semantic information exchange when constraints are logically conjoined together. Simply speaking, it works by looking for variable pairs which are *generated* in the same way. Under this condition, constraints that apply to one variable may apply to the other variable. For example, variable *x* and *y* share each other's constraints as they are both instances of the *pc* class:

```
constrain all x in pc
to have cpu(x)="pentium 4"
```

```
constrain all y in pc
to have memory(y)>=128
```

In fact, sharing of constraint also happens between variables of a superclass and a subclass, as constraints are inherited by the subclass from the superclass. That means the constraint fusion engine needs knowledge of the class hierarchy in the problem domain in order to make the correct inference. In this case, the domain model is readily available as an RDF schema which can be accessed as RDF triples.

Given a constraint, the fusion engine can easily retrieve the RDF schema of the domain model by following the explicit links. By collecting RDF triples on the `rdfs:subClassOf` predicate, the fusion engine then gets a complete picture of the class hierarchy in the problem domain. The small granularity of knowledge represented in the triple form allows the knowledge processing component (in this case, the

```
constrain all t in travel_plan such that
travel_method(t) = train and
travel_class(t) = sleeper_train
to have arrival_time(t) < 0830
```

Figure 5: A call for services in CIF

constraint fusion engine) to selectively access the required knowledge in a quick and convenient manner.

## Conoise<sup>6</sup> and Virtual Organisations

Virtual organisations (VOs) in Conoise<sup>7</sup> are composed of a number of autonomous entities (representing different individuals, departments and organisations) each of which has a range of problem solving capabilities and resources at their disposal. These entities co-exist and sometimes compete with one another in a ubiquitous virtual market place. Each entity attempts to attract the attention of potential customers and ultimately tries to sell them its services by describing the cost and quality of the service.

Sometimes, however, one or more of the entities may realise there are potential benefits to be obtained from pooling resources: either with a competitor (to form a coalition) or with an entity with complementary expertise (to offer a new type of service). When this potential is recognised, the relevant entities go through a process of trying to form a new VO to exploit the perceived niche.

Given the independent nature of the entities involved, this process may succeed or it may fail. If it succeeds, the collection of independent entities have to start acting as a single conceptual unit. In particular, they need to cooperate and coordinate to deliver the services of the newly formed organisation. In dynamic environments, the context may change at any time, such that the VO is no longer viable. Then it will either need to disband or re-arrange itself into a new organisation that better fits the prevailing circumstances.

In Conoise we represent the knowledge and communication between the agents using CIF/RDF. We build on the ability to combine the CIF/RDF knowledge (Gray, Hui, & Preece 1999) so that we can use many disparate information sources to help in the formation, management and dissolution of these VOs. The ability to combine information from these different sources means that our decision making process uses as much knowledge as is available at the time to aid in the process of choosing other agents as VO partners.

## Conoise Agent Design

Conoise uses CIF/RDF constraints to represent the services required in such VO's, and uses a CSP (Constraint Satisfaction Program) solver to provide the reasoning

<sup>6</sup>Constraint Oriented Negotiation in Open Information Seeking Environments

<sup>7</sup><http://www.conoise.org>

```

constrain all s in service_description
such that name(s) = Euston
to have travel_info(s) = cancelled

```

Figure 6: Environment information held as CIF

process for their identification, formation, management and dissolution. The reason for using a CSP is that, as in KRAFT:

- We get data independence across platforms
- It is easy to combine conjunctive First Order Logic constraints compared to imperative code

Typically the starting point for this process will consist of an agent receiving a call for bids to provide a service. The agent must then decide what course of action to take to provide that service. This can be either:

- To provide a bid based on its own resources.
- To provide a bid based on the resources available from its membership of an existing Virtual Organisation.
- To provide a bid by creating a new Virtual Organisation, thereby initiating a new call for bids to find new VO partners.

When deciding on which action to take the agent must be aware of the current status of other agents and their abilities as well as what resources it can itself provide.

Figure 5 shows an example call to provide a service (a `travel_plan` representing a complete travel 'package' to a specified destination (Chalmers, Gray, & Preece 2002)). The RDF is not shown for reasons of space, but is similar to the example in figure 4. This constraint determines that if the travel plan includes travel by train and it is a sleeper train, then it must arrive at its destination by 8:30am. The added bonus of such representation is that, once in the CIF format, we can combine this call for services with other information (also held as CIF/RDF), such as previous knowledge, current commitments and environment information in the CLP process, thus providing us with a more detailed constraint problem to manage the VO.

For example, this train service information can be combined with information that the agent receives about the availability of trains from London Euston (figure 6).

This means that the `travel_plan` will not only take into account the sleeper train and arrival time specific constraints, but can also combine this with knowledge on a specific station it has received.

### Conoise Agent Implementation

The Conoise agent architecture is shown in figure 7. The agents are built using the JADE Java-based agent platform<sup>8</sup>. Each agent in Jade communicates using

<sup>8</sup><http://sharon.csel.it/projects/jade/>

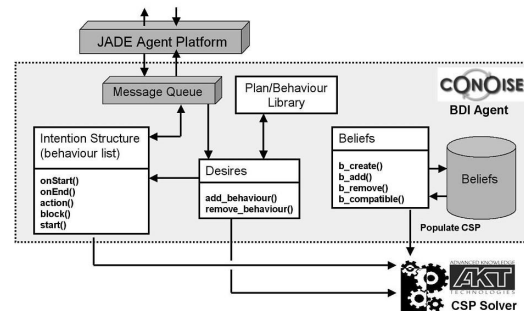


Figure 7: The Conoise Agent Architecture

```

Do
  options := option_generator(event_queue,B,D,I)
  selected-options := deliberate(options,B,D,I)
  update-intentions(selected-options,I)
  execute(I)
  get-new-external-events();
  drop-successful-attitudes(B,D,I)
until quit

```

Figure 8: BDI Algorithm

FIPA ACL<sup>9</sup>, leaving the content language to be specified by the agent (here CIF/RDF).

The agent architecture is based on the Beliefs, Desires, Intentions (BDI) agent model (Rao & Georgeff 1995). This uses data structures to represent the beliefs (what the agent knows), the desires (what the agent wants to do) and the intentions (how the agent achieves its desires) and the agents reasoning and decision making process comes from the interaction of these data structures according to a deliberation process represented as a top level pseudo-code algorithm in figure 8). The desires of the agent are represented as constraints, and therefore exchangeable between agents as CIF/RDF constructs.

To parse the RDF we are using the Jena toolkit<sup>10</sup>, a Java API for parsing and manipulating RDF data models. It takes the RDF constraint and stores it as a set of Subject-Predicate-Object triples in a `Model` object. This object can then be queried using the API methods `getSubject()`, `getPredicate()` and `getObject()`. The Java code fragment in figure 9 shows the `model` variable being instantiated with the RDF from the URI specified by the `constraintURI` variable. The `while` loop shows how we can then parse this model and extract the necessary subject-predicate-object values.

Once a message has been received by the agent it is passed into the message queue (figure 7), where it is parsed into a new Jena `Model` (as shown). The agent

<sup>9</sup><http://www.fipa.org>

<sup>10</sup><http://www.hpl.hp.com/semweb>

```

Model model = new ModelMem();
JenaReader r= new JenaReader();
r.read(model, constraintURI);
StmtIterator iter = model.listStatements();

while (iter.hasNext()) {
    com.hp.hpl.mesa.rdf.jena.model.Statement stmt
        = iter.next();
    Resource subject = stmt.getSubject();
    Property predicate = stmt.getPredicate();
    RDFNode object = stmt.getObject();
    .....
    .....
}

```

Figure 9: Using Jena to create RDF triples.

then creates a desire object and, from this, creates a set of possible intentions describing ways of completing this desire. It then chooses between these competing intentions (using the CLP solver and its beliefs) and executes the chosen one, which will complete the necessary steps to fulfill the desire. As we have all the agents other commitments (the other desires it is doing at the same time) stored in desire objects as CIF/RDF parsed Jena models, we can combine these with the current desire when deliberating to provide a choice of intention.

## Discussion & Related Work

In this paper, we have demonstrated how RDF has a far broader applicability than its original role as a metadata format for web resources. Our usage of RDF places it at the core of an agent-mediated distributed knowledge architecture, in which RDF provides: a carrier for the communication of entity-relational information between web services, a format for mobile and self-describing constraints, and a content language for inter-agent communication. In this section, we summarise our experience in using RDF, and draw comparisons with related work.

The most common approach for transport of structured information between web services is to define the data using XML Schemas (or DTDs) and use an XML-RPC framework such as SOAP. The use of RDF as a layer on top of XML means that the communicated information has a data model (semantics), whereas the XML Schema/DTD-defined data has only a structure (syntax). Most of the work that acknowledges the importance of a semantics for communicated data — essentially the founding principle of the Semantic Web movement — proceeds to define additional layers on top of RDF and RDFS (DAML+OIL, OWL, etc). *0-Telos-RDF* (Nejdl, Dhraief, & Wolpers 2001) even proposes an alternative to RDF, serialisable in XML, but with much clearer axiomatic semantics. In contrast, we view RDF and RDFS as sufficiently useful and extensible in itself; in defence of this position, we have shown that the RDF data model is adequately

expressive to transport data originally stored against the P/FDM semantic data model. Other work within the AKT project further supports this position, where RDF is used to represent a large repository of information on the research activities of UK universities, and to draw inferences from this information (Alani, O'Hara, & Shadbolt 2002). Further evidence for the utility of RDF in knowledge management applications is provided by the EU COMMA project (Gandon & Dieng-Kuntz 2002). In all of these approaches, the sufficiency of RDF and RDFS to represent and communicate both ontological (schema) information and individual instances is demonstrated.

Our original inspiration for using RDF and RDFS as a format for mobile and self-describing constraints was the original OIL proposal (Broekstra *et al.* 2001), wherein the RDF data model was extended with logical connectives (and, or, not, etc) which were themselves syntactically defined using RDFS. The more recent Semantic Web work on DAML+OIL has backed-away from this original position, apparently because the logical apparatus of boolean expressions has been “pushed up” to a higher *logic layer*, above the DAML+OIL *ontology layer*<sup>11</sup>. While this is not unreasonable, we have opted to define our constraint format CIF directly on top of RDFS, as we view RDF itself as a major application of constraints. For example, CIF can be used to define integrity constraints on RDF classes, even though they are not expressible in RDFS itself. Our CIF work is related to the ongoing RuleML initiative<sup>12</sup> (positioned at the Semantic Web logic layer) and, while we would see our work coming into alignment with RuleML in the future, we note that currently RuleML is more concerned with traditional if-then rules rather than declarative constraints.

OCL is a declarative expression language for annotating UML class diagrams with invariants, especially more complex cardinality constraints. It has been used to create formal models of configuration problems (Felfernig, Friedrich, & Jannach 2001), which is more concerned with formal correctness than with AI problem-solving. However their logical formalism is very similar to ours, and a UML class diagram is just an ER diagram with methods, where we use functions.

Our high-level architecture is agent-based, and RDF serves as the content language. Again, this communication framework stands in contrast to the mainstream RPC model of web inter-application communication, where SOAP dominates. However, agent-to-agent communication is far more flexible than RPC, being loosely-coupled, asynchronous, and allowing individual network nodes full autonomy (Vinoski 2002). Regrettably, the current state-of-the-art in agent communication languages is not directly compatible with RDF. Both FIPA and KQML rely primarily on a LISP-

<sup>11</sup><http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html>

<sup>12</sup><http://www.dfki.uni-kl.de/ruleml/>

based syntax rather than XML and, while FIPA informally allows RDF as a content language, this is not part of the standard. Nevertheless, we assert that the combination of an agent architecture and RDF as a content language is powerful and open, and believe that a fusion of web standards with agent standards is a desirable goal of both communities.

### Conclusion

In this paper we have argued that, for agent-mediated knowledge systems, there is a clear need not just to represent and communicate information, but also to reason with it. We have shown how we are using RDF to do both: we can readily map entity-relational data into RDF, communicate it within an agent communication language, and reason with it using constraints also defined in RDF. In conclusion, we see the desirable features of RDF for agent-mediated knowledge systems as being:

- *simplicity* of its triple-based data model;
- *uniformity* in the representation of both knowledge (instances) and meta-knowledge (schemas);
- *portability* of the XML serialisation;
- *web-enabled*, compatible with W3C standards;
- *extensibility*, exemplified by the layering of RDFS on RDF.

### Acknowledgements

This work is supported under the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant number GR/N15764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. The constraint fusion services were developed in the context of the KRAFT project, funded by the EPSRC and British Telecom. The CONOISE project is funded by BTextact Technologies.

### References

- Alani, H.; O'Hara, K.; and Shadbolt, N. 2002. Ontocopi: Methods and tools for identifying communities of practice. In Musen, M.; Neumann, B.; and Studer, R., eds., *Intelligent Information Processing*. Kluwer Academic Press. 225–236.
- Bassiliades, N., and Gray, P. 1994. CoLan: a Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering* 14:203–249.
- Broekstra, J.; Klein, M. C. A.; Decker, S.; Fensel, D.; van Harmelen, F.; and Horrocks, I. 2001. Enabling knowledge representation on the web by extending RDF schema. In *World Wide Web*, 467–478.
- Chalmers, S.; Gray, P.; and Preece, A. 2002. Supporting virtual organisations using BDI agents and constraints. In Klusch et al. (2002), 226–240.
- Felfernig, A.; Friedrich, G.; and Jannach, D. 2001. Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering* 15(2):165–176.
- Fiddian, N. J.; Marti, P.; Pazzaglia, J.-C.; Hui, K.; Preece, A.; Jones, D. M.; and Cui, Z. 1999. A knowledge processing system for data service network design. *BT Technical Journal* 17(4):117–130.
- Gandon, F., and Dieng-Kuntz, R. 2002. Distributed artificial intelligence for distributed corporate knowledge management. In Klusch et al. (2002), 202–217.
- Gelernter, D., and Carriero, N. 1992. Coordination languages and their significance. *Communications of the ACM* 35(2).
- Gray, P. M. D.; Embury, S. M.; Hui, K.; and Kemp, G. J. L. 1999. The evolving role of constraints in the functional data model. *Journal of Intelligent Information Systems* 12:113–137.
- Gray, P. M. D.; Hui, K.; and Preece, A. D. 1999. Finding and moving constraints in cyberspace. In *Intelligent Agents in Cyberspace*, 121–127. AAAI Press. Papers from the 1999 AAAI Spring Symposium Technical Report SS-99-03.
- Gray, P. M. D.; Hui, K.; and Preece, A. D. 2001. An expressive constraint language for semantic web applications. In Preece, A., and O'Leary, D., eds., *E-Business and the Intelligent Web: Papers from the IJCAI-01 Workshop*. AAAI Press. 46–53.
- Klusch, M.; Ossowski, S.; and Shehory, O., eds. 2002. *Proceedings of the 6th International Workshop on Cooperative Information Agents (CIA 2002)*, number 2446 in Lecture Notes in Artificial Intelligence. Madrid, Spain: Springer Verlag.
- Nejdl, W.; Dhraief, H.; and Wolpers, M. 2001. O-Telos-RDF: A resource description format with enhanced meta-modeling functionalities based on o-Telos. In *Workshop on Knowledge Markup and Semantic Annotation at the First International Conference on Knowledge Capture (K-CAP'2001)*. URL: <http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2001/kcap01-workshop.pdf>.
- Preece, A.; Hui, K.; Gray, A.; Marti, P.; Bench-Capon, T.; Cui, Z.; and Jones, D. 2001. KRAFT: An agent architecture for knowledge fusion. *International Journal of Cooperative Information Systems* 10(1 & 2):171–195.
- Rao, A. S., and Georgeff, M. P. 1995. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*.
- Vinoski, S. 2002. Putting the "web" into web services: Web services interaction models, part 2. *IEEE Internet Computing* July/Aug, 90–92.

*In Stanford Spring Symposium on Agent-Mediated Knowledge Management, to appear, 2003.*