An Expressive Constraint Language for Semantic Web Applications

Peter Gray, Kit Hui, Alun Preece

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

Abstract

We present a framework for semantic web applications based on constraint interchange and processing. At the core of the framework is a well-established semantic data model (P/FDM) with an associated expressive constraint language (Colan). To allow data instances to be transported across a network, we map our data model to the RDF Schema specification. To allow constraints to be transported, we define a Constraint Interchange Format (CIF) in the form of an RDF Schema for Colan, allowing each constraint to be defined as a resource in its own right. We show that, because Colan is essentially a syntactically-sugared form of first-order logic, and P/FDM is based on the widely-used extended ER model, our CIF is actually very widely applicable and reusable. Finally, we outline a set of services for constraint fusion and solving, which are particularly applicable to business-to-business e-commerce applications. All of these services can be accessed using the CIF.

Introduction and Motivation

The *semantic web* vision is to enable rich machine-processing of web information sources (Berners-Lee 1999). Most of the work done to date in realising this vision has focussed upon developing "web friendly" representations for structured data. Building on the XML standard, a number of proposals for expressing data schemas have appeared, including RDF, RDFS, and XML Schema (Fensel *et al.* 2000b). These approaches support the representation and communication of entity-relational information in web applications, for example, allowing a set of instances of some entity type to be gathered from several structured web pages, and transported for storage in a web-connected database.

The next significant stage in the realisation of the semantic web lies in extending the basic data schema representations with information on how the data can and should be used. This information can take various forms, including logical axioms, rules, constraints, or even functional and procedural representations. This work will create the *reasoning services* for the semantic web. Early work in this direction included inference engines for large-scale ontologies (Fensel *et al.* 2000a; Heflin & Hendler 2000), mechanisms for representing and reasoning with business rules (Reeves *et al.* 1999), and mobile constraint languages and constraint-solving frameworks (Torrens & Faltings 1999; Gray, Hui, & Preece 1999). Building on this early work, there is now a significant number of large-scale projects developing semantic web reasoning services, including AKT[1], DAML[2], IBROW[3], and Ontoknowledge[4].

In this paper, we present the current status of a Constraint Interchange Format (CIF) under development as part of the AKT project, aimed specifically at business-to-business e-commerce applications on the semantic web. The constraint language is well-established, having evolved during the course of a number of previous projects, including the P/FDM[5] and KRAFT[6] projects. It has been used successfully in diverse and challenging application domains such as representing and reasoning about protein structures in biomedical applications (Kemp *et al.* 2000), and configuring telecommunications network services (Fiddian *et al.* 1999). Earlier versions of the CIF were based on Prolog term structures; this paper presents a new XML encoding of the CIF, designed to be more open and less platform-dependent than its predecessor. The new XML-CIF has been modelled using the XML Schema specification, so that constraints become web resources about which statements can be made (for example, statements about the authorship and context of a constraint). In addition, the CIF has been designed to refer to XML Schema definitions so that the terms referred to in a constraint can have corresponding XML Schema descriptions. The motivation here is to allow, in principle, any application of RDF Schema to use our CIF and constraint-solving services.

The paper is organised as follows: the next section introduces our constraint language, Colan, and the semantic data model upon which it operates; the following section describes the design of the XML-CIF encoding of Colan expressions; the penultimate section gives an overview of the constraint-solving services already available for semantic web applications using the CIF; the final section con-

[1] www.aktors.org
[2] www.daml.org
[3] www.swi.psy.uva.nl/projects/ibrow/home.html
[4] www.ontoknowledge.com
[5] www.csd.abdn.ac.uk/∼pfdm
[6] www.csd.abdn.ac.uk/research/kraft.html

```
constrain each t in tutor
    such that astatus(t)="research"
no s in advisee(t) has grade(s) =< 30;

constrain each r in residue to have
    distance(atom(r,"sg"),
      atom(disulphide(r),"sg")) < 3.7;
```

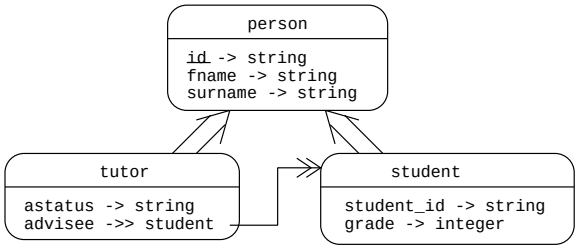

Figure 1: Example Colan constraints from different application domains. The ER diagram models the relationships between entity classes in the first constraint.

cludes.

Colan: A Constraint Language Based on an Object Data Model

An entity-relationship (ER) diagram (or UML class diagram) is commonly thought of as being useful chiefly as a diagrammatic guide or visualisation of the types of entities and relationships in a database (or an object-oriented application); it is less commonly seen as an operational basis for forming queries or specifications. In our previous work (Bassiliades & Gray 1994; Gray *et al.* 1999) we showed how quantified constraints can be expressed in our Colan language, in a very readable form of first order logic, including evaluable functions which can be computed over data values represented in the ER diagram. Hence, the underlying data model is called the Functional Data Model (FDM). The semantics of the objects referred to in Colan constraints are described in terms of this extended ER data model, which is of the kind in widespread use in UML and in database schemas. Our FDM, P/FDM (Prolog/Functional Data Model) is a Semantic Data Model based on Shipman's original data model (Shipman 1981).[7]

Two example Colan constraints are shown in Figure 1. The first example demonstrates how Colan (Bassiliades & Gray 1994) expresses a constraint on a university database containing student records. The same constraint language is applicable to the domain of protein structure modelling, as shown by the second example constraint restricting bond lengths. In the first example, a variable `t` ranges over an entity type `tutor` which is populated with stored object instances. Each of these instances may be related to instances of student entities through the relationship `advisee`, which delivers a set of related entities as in an object-oriented language. These entities can be restricted by the values of attributes such as `grade`. There are also other entity types such as `residue` (representing parts of protein chains) which have method functions for determining distances by computation. Thus functions may also represent a derived relationship, or method. The entity classes can form part of a subtype hierarchy, in which case all properties and methods on the superclass are inherited by each subclass. Method definitions may be overridden, but not constraints.

This is significant for semantic web applications, since it means that information represented in this way is not restricted to human inspection — it can be proof-checked mechanically, transformed by symbol manipulation, or sent to a remote constraint solver. Moreover, given a standardised interchange format, data and attached constraints from multiple sources can be gathered together, checked for compatibility, and used to derive new information. Because our P/FDM data model is an extended ER model, it maps very easily onto the RDF Schema specification, as we shall show in the following section.

Colan is as expressive as the subset of first-order logic that is useful for expressing integrity constraints: namely, *range-restricted* constraints. This class of constraints includes those first-order logic expressions in which each variable is constrained to be a member of some finite set of values. Its use avoids the classic problem of safety of certain expressions (Van Gelder & Topor 1991).

Thus we have a formalism which gives us a precise denotation for constraints but it does not force us to evaluate them as integrity checks. The constraint expresses a formula of logic which is true when applied to all the instances in a database, but it is also applicable to instances in a solution database which is yet to be populated with constructed solutions by a solver process (Gray *et al.* 1999). Here it is behaving more like a specification than as an integrity check. The power of this in the context of the semantic web is that constraints can be passed as a form of mobile knowledge between agents and processes; they are no longer tied to a piece of database software.

In general, we believe that there is much to be gained from using the FDM as an intermediate representation of queries and constraints sent to remote sites, even when target databases use relational storage (Kemp *et al.* 2000). This is because FDM was originally devised for this purpose, in the Multibase heterogeneous distributed database project (Landers & Rosenberg 1982).

The Role of Constraints in Semantic Data Models

One of our original motivations for the introduction of constraints into P/FDM was to enrich the capability of the data model in capturing semantics. Data models are more than type systems (which they resemble syntactically) because they also represent constraints on the data. Thus a powerful constraint language has enabled us to capture the semantic information that others have endeavoured to express by extending the diagrammatic notation of the ER model. Thus, although the original FDM was somewhat lacking in se-

[7] For full details of P/FDM, Colan, Colan, and related work, see: www.csd.abdn.ac.uk/∼pfdm

ALICE version:

```
all T in tutor (where S.astatus="research")
  implies: (all S1 in S.studadvised implies:
    (S1.grade > 60))
```

Colan version:

```
constrain each t in tutor
  such that astatus(t)="research"
  so that each s in
    advisee(t) has grade(s) > 60;
```

First-order logic version:

$(\forall t,s,g)\ tutor(t) \wedge astatus(t,\text{`research'}) \wedge advisee(t,s) \wedge grade(s,g) \Rightarrow g>60$

Figure 2: Example constraint shown in ALICE, Colan, and first-order logic versions.

mantics, we have been able to more than make up for that by introducing constraints.

In doing this, we have capitalised on a cardinal virtue of the FDM, that it enables one to make well-formed mathematically precise computations over data stored as instances of entities related by an ER diagram. Thus we based the Colan formalism for constraints on just these well-formed computations, revising and extending it to use the full expression syntax of P/FDM's Daplex query language. Crucially, because the expressions are fully quantified and referentially transparent, it is straightforward to move them into other contexts and transform them in ways which preserve their semantics. This would not be so if our expression of data semantics had been confined to a diagrammatic notation.

Much early work on constraints, including deductive database approaches, used a predicate logic formalism. Closely related work by (Urban 1989) led to ALICE which is a declarative language for the expression of complex logic-based constraints in an OODB environment, also using a semantic data model. An early example of representing semantic constraints in a functional syntax was in the Extended Functional Data Model project (Kulkarni & Atkinson 1986; Gray, Kulkarni, & Paton 1992). Other proposals for functional constraint languages include PRISM (Shepherd & Kerschberg 1984), ADAPLAN (the Applicative Data Programming Language) (Erwig & Lipeck 1991) and PFL (Persistent Functional Language) (Reddi 1993).

Figure 2 shows the same constraint in three representations: ALICE, Colan, and first-order logic.

Note that, in Colan, the phrase `so that` is optional preceding a quantifier, and that `has` can be replaced by `to have`. These small changes do not affect the mathematical meaning, but they do help to make the functional style of Colan more readable to a non-mathematician than the predicate logic version. Thus, when following a universal quantifier in functional form, the keyword `has` behaves like an implication. However, following an existential quantifier, `has` behaves like a conjunction (possibly with an implied `true`). Further details, showing how universal quantifiers have a weak translation while existential quantifiers have a strong one are given in (Embury & Gray 1995).

In summary, we have introduced our constraint language, Colan, and the semantic data model upon which it operates, and we have argued for the applicability of this language and data model to semantic web applications. The following section describes our new XML encoding of Colan in the form of a Constraint Interchange Format (CIF) based on the RDF Schema specification.

XML Constraint Interchange Format

In defining our Constraint Interchange Format, we were guided by the following design principles:

- the CIF would need to be serialisable into XML, to make it maximally portable and open;
- constraints should be represented as resources in RDF, so that RDF statements can be made about the constraints themselves;
- there must be no modification to the existing RDF and RDF Schema specifications, so that the CIF would be layered cleanly on top of RDF;
- it must be possible for constraints to refer to terms defined in any RDF Schema, with such references made explicit.[8]

As we showed in the previous section, the entity-relational basis of both our P/FDM data model and RDF made it relatively straightforward to map from the former to the latter. In building the RDF Schema for our CIF we were guided by the existing grammar for Colan (Gray *et al.* 1999) which relates constraints to entities, attributes and relationships present in the ER model. This grammar serves as a *metaschema* for the Colan constraints (such metaschemas are very common in relational and object database systems). A number of issues arose in developing the RDF Schema for CIF, discussed in the following subsections.

Metaclasses for Entities and Relations

Our implementation of the P/FDM semantic data model makes use of an `entmet` class that holds information on all *entity* classes, and a `propmet` class that holds information on relationships (functions), both stored and derived (Embury & Gray 1995). The metaschema is fully queryable, and for this purpose the property values of members of these metaclasses are all held as strings (as is common in data dictionaries), so that the answer to a query on them returns the name of an entity or relation and not the contents of the entity or relation. The P/FDM Daplex definitions of the `entmet` and `propmet` classes are shown in Figure 3, together with their superclass, `objmet`.

[8] Essentially, we use RDF Schemas as a simple but practical form of ontology; previous work in the context of the KRAFT project (Preece *et al.* 2001) highlighted the importance of making explicit ontological mappings within knowledge resources.

```
% objmet - superclass of entity and prop-
erty metaclasses
declare objmet ->> entity
declare oname(objmet) -> string

% entmet - the metaclass of all en-
tity classes
declare entmet ->> objmet
declare super(entmet) -> entmet
declare rdfname(entmet) -> string
% link to RDF Schema

% propmet - the metaclass of all proper-
ties (functions)
declare propmet ->> objmet
declare fname(propmet) -> string
declare firstargtype(propmet) -> entmet
declare resulttype(propmet) -> entmet
declare has_inv(propmet) -> boolean
declare rdfname(propmet) -> string
% link to RDF Schema
```

Figure 3: P/FDM Daplex definitions for entity and property metaclasses.

The property `rdfname` on the `entmet` and `propmet` classes holds the unique URI for an RDF resource, and thus provides an explicit link to the RDF Schema definition for the corresponding RDF Schema class or property. Thus, constraints carry explicit relationships to the domain ontology (as represented by an RDF Schema) for the terminology to which they refer.

In the RDF Schema we introduce the corresponding metaclasses, `entmet` and `propmet`, which will record the graph of classes and properties. Thus there will be one instance of the `entmet` class for each actual class representing a real-world entity in the instance level RDF Schema for a given application domain. These metaclasses then provide the natural result types for properties used in constraints. Thus, for example, we can use them to say that an atomic boolean value in a predicate in a constraint can be derived by comparing the property value of a variable which holds an entity identifier with another value given by an expression. This entity and property must be known. We could even write a metalevel constraint to require their consistency, as checked by a type checker.

Figure 4 shows the RDF Schema definitions corresponding to the Daplex definitions of the `objmet` and `entmet` classes from Figure 3. It is worth noting that, because properties in RDF are global, some of the original local P/FDM property names must be renamed (for example, `entmet_rdfname` in Figure 4, renamed from `rdfname` in Figure 3).

The basic rules we used when mapping the P/FDM declarations to RDF Schema are as follows:

- a P/FDM class c defined as an `entity` (declared as c `->> entity`) maps to an RDF resource of type `rdfs:Class` (where `rdfs` is the namespace prefix for the RDF Schema descriptions);

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

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

<rdf:Property rdf:ID="oname">
  <rdfs:domain rdf:resource="#objmet"/>
  <rdfs:range rdf:resource=
  "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdfs:Class rdf:ID="entmet">
  <rdfs:subClassOf rdf:resource="#objmet"/>
</rdfs:Class>

<rdf:Property rdf:ID="super">
  <rdfs:domain rdf:resource="#entmet"/>
  <rdfs:range rdf:resource="#entmet"/>
</rdf:Property>

<rdf:Property rdf:ID="entmet_rdfname">
  <rdfs:domain rdf:resource="#entmet"/>
  <rdfs:range rdf:resource=
  "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
...
</rdf:RDF>
```

Figure 4: RDF Schema definitions for the `objmet` and `entmet` classes.

- a P/FDM class c declared to be a subtype of another class s (declared as c `->>` s) maps to an RDF resource of type `rdfs:Class`, with an `rdfs:subClassOf` property the value of which is the class named s;
- a P/FDM function f declared on entities of class c, with result type r (declared as $f(c)$ `->` r) maps to an RDF resource of type `rdf:Property` with an `rdfs:domain` of c and an `rdfs:range` of r.

Representation of Variables in Constraints

A fundamental notion in Colan is that a variable is always introduced in conjunction with a set that it ranges over. Thus terms such as `(p in pc)` and `(e in employee)` are common, as in the example expressions:

```
(p in pc) such that name (p) = "xxx"
(e in employee) such that
  salary (e) > 5000 and age (e) < 50
```

This is represented in the syntax by the `setmem` metaclass, while variables themselves are described by the `variable` class, both defined as shown in Figure 5. An instance of the `variable` class is a legal instance of an `expr` class (representing an expression) by virtue of a series of subclass relationships. There is also a semantic constraint that such an instance must already have been introduced with a quantifier; this is not currently captured in the RDF Schema but could possibly be represented using the RDF Schema `ConstraintResource` extension mechanism. An example XML-CIF fragment corresponding to the Colan fragment `(p in pc)` is shown in Figure 6.

```
<rdfs:Class rdf:ID="variable">
  <rdfs:subClassOf rdf:resource="#singleton"/>
</rdfs:Class>

<rdf:Property rdf:ID="varname">
  <rdfs:domain rdf:resource="#variable"/>
  <rdfs:range rdf:resource=
  "http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdfs:Class rdf:ID="setmem">
  <rdfs:subClassOf rdf:resource="#boolprim"/>
</rdfs:Class>

<rdf:Property rdf:ID="var">
  <rdfs:domain rdf:resource="#setmem"/>
  <rdfs:range rdf:resource="#variable"/>
</rdf:Property>

<rdf:Property rdf:ID="set">
  <rdfs:domain rdf:resource="#setmem"/>
  <rdfs:range rdf:resource="#setexpr"/>
</rdf:Property>
```

Figure 5: RDF Schema definitions relating to the `setmem` metaclass.

```
<cif:setmem>
  <cif:var>
    <cif:variable ID="#p">
      <cif:varname>p</cif:varname>
    </cif:variable>
  </cif:var>
  <cif:set>
    <cif:entset>
      <cif:entclass>
      http://www.aktors.org/domain/pc_config#pc
      </cif:entclass>
    </cif:entset>
  </cif:set>
</cif:setmem>
```

Figure 6: XML-CIF fragment corresponding to the Colan fragment `(p in pc)`.

In summary, our CIF RDF Schema[9] serves the purpose of describing what are valid constraints, themselves expressed at an instance level in RDF. It combines the information in a grammar, which is normally used by a syntax checker or a parser, with information normally held in a database schema and used to validate database queries. The interesting thing is that the P/FDM data description language is expressive enough to capture this, especially cardinality constraints. RDF is not so expressive, although it does provide for schema constraints such as cardinality through the `ConstraintResource` extension mechanism.

It should be noted that the metaschema makes a clean separation between the description of *constraints* (both universal and existential) and *expressions*. Constraints and their boolean components are a representation of first-order logic, with the usual connectives. Any knowledge source that uses FOL should be able to understand this. Expressions refer to facts about entities, their subtypes, attributes and relationships, and is based on the concepts of an ER model, which are very widely used. The ER model abstracts over relational storage, flat files and object-oriented storage, following the principle of data independence. It does not tie one to any particular system, such as Oracle or P/FDM.

[9] Space prohibits us from including the full CIF RDF Schema here, but both it and the full P/FDM schema are available at: www.csd.abdn.ac.uk/research/akt/cif/

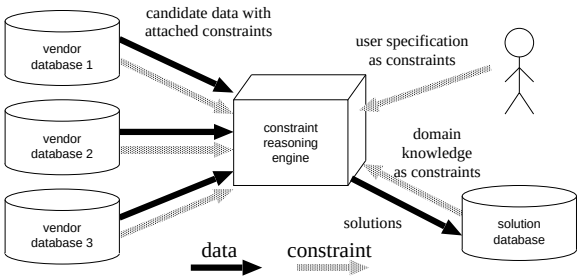

Figure 7: KRAFT utilises both data and constraints which are mobile in the KRAFT domain; constraint knowledge flow is shown as grey arrows while data flow is in black.

The following section describes our existing framework for using CIF constraints, as originally developed in the KRAFT project (Preece *et al.* 2001) for supporting virtual organisations in which communication and coordination are achieved by the exchange and fusion of constraint knowledge.

Fusion of Constraints from Different Sources

Having a semantic data model (P/FDM) extended with constraints (Colan) and mapped into an open interchange format (RDF Schema and XML-CIF) supports a range of applications in which information needs to be moved across a network with rich metalevel 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 (for example, consumer electronic equipment, package holidays, fitted kitchens, or financial products). In each case, there are various kinds of constraints which must be aggregated and solved over the available component instances: constraints representing customer requirements ("I want a PC with a colour printer"), constraints representing rules for what constitutes an acceptable package ("any printer must have a driver that is compatible with the PC OS"), and constraints representing restrictions on the use of particular components ("this printer has drivers only for Windows OSes").

In this last case, the ability to store constraints together with data in P/FDM allows instructions 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 used in the KRAFT project (Preece *et al.* 2001) differs from a conventional distributed database system where only database queries and data objects are shipped.

Since constraint processing is a more general formalism, it can also be used to represent user specifications

on the required solutions. Here, the user-agent (Figure 7) serves as another information source feeding user requirement knowledge into the system in the form of constraints.

In general, there are three different ways to utilise the fused constraints:

1. We can check the constraints against sets of objects retrieved by a distributed database query across the network, so as to reject any not satisfying the conditions.
2. We can use some combination of selection information in the constraint to refine the distributed database query, and thus do it more efficiently. This could also use the principles of semantic query optimisation.
3. We can use constraint logic solving techniques to see if a complex set of constraints, whose form is not known until runtime, does have a solution.

A Constraint Fusion Example

To demonstrate constraint fusion from different sources, consider a configuration problem where a PC is built by combining components from vendors. All of the constraints below are expressed in Colan and so would be exchanged using the XML-CIF representation described in the previous section.

The user specifies his requirement in the form of constraint through the user-agent. In this example, he specifies that the PC must use a "pentium2" processor but not the "win95" OS:

```
constrain each p in pc
    to have cpu(p)="pentium2"
    and name(has_os(p)) <> "win95"
```

For the components to fit together, they must satisfy certain constraints imposed by the solution database. For example, the size of the OS must be smaller or equal to the hard disk space for a proper installation:

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

Now the candidate components from different vendors may have instructions attached to them as constraints. In the vendor database of operating systems, "winNT" requires a memory of at least 32 megabytes:

```
constrain each p in pc such that
    manufacturer(p) = "HAL"
    and name(has_os(p))="winNT"
        to have memory(p) >= 32
```

When we fuse all constraints together so that they apply to the solution database, we get the description of an equivalent constraint satisfaction problem (note the *conditional constraint* in the last line):

```
constrain each p in pc
    to have cpu(p)="pentium2"
    and name(has_os(p)) <> "win95"
    and size(has_os(p)) =< size(has_disk(p))
    and if manufacturer(p) = "HAL"
        and  name(has_os(p))="winNT"
        then memory(p)) >= 32 else true
```

The process of solving the application problem, therefore, is to retrieve data from other databases and populate the solution database while satisfying (i) all the integrity constraints attached to the solution database, (ii) constraints on data objects and (iii) user requirement constraints. This process corresponds to a generate-and-test approach where invalid candidates are rejected by database integrity constraints. A more efficient prune-and-search approach can be achieved by exporting constraint fragments to a constraint fusing mediator which composes the overall description as a constraint satisfaction problem (CSP) for a configuration task (Gray *et al.* 1998) so that it may plan the solution. The CSP is then analysed and decomposed into database queries and constraint logic programs which are fed across to distributed databases and constraint solvers, under the control of a mediator (Hui & Gray 2000; Hui 2000).

Discussion and Conclusion

In this paper, we have presented a framework for semantic web applications based on constraint interchange and processing. At the core of the framework is a well-established semantic data model (P/FDM) with an associated expressive constraint language (Colan). To allow data instances to be transported across a network, we have mapped our data model to the less expressive (but adequate) RDF Schema. To allow constraints to be transported, we have provided a Constraint Interchange Format (CIF) in the form of an RDF Schema for Colan, allowing each constraint to be defined as a resource in its own right. Because Colan is essentially a syntactically-sugared form of first-order logic, and P/FDM is based on the widely-used extended ER model, our CIF is actually very widely applicable and reusable. From the KRAFT project, we have implemented a set of services for constraint fusion and solving, which are particularly applicable to business-to-business e-commerce applications. All of these services can be accessed using the CIF.

We targeted our approach for use with RDF Schema, and in particular the XML encoding of RDF Schema, in an effort to maximise the applicability of our work. RDF Schema is the simplest and most universal of the proposed semantic web data representations, while still being adequately expressive for our purposes. In linking Colan to RDF Schema, we also allow its usage with more expressive data modelling languages built on top of RDF Schema, including DAML-ONT[10] and OIL[11]. However, a basic requirement of our approach in defining the RDF Schema for Colan expressions was that it should in no way require modification to the underlying RDF definitions (this is in contrast to the OIL approach, which requires modification at the RDF layer in order to capture certain kinds of expression (Decker *et al.* 2000)).

Our constraint interchange and solving services are being incorporated into the AKT infrastructure, as one of the basic knowledge reuse mechanisms in the AKT service

[10] www.daml.org

[11] www.ontoknowledge.com/oil

layer. Further information on this work can be found at: www.aktors.org

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.

References

Bassiliades, N., and Gray, P. 1994. CoLan: a Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering* 14:203–249.

Berners-Lee, T., ed. 1999. *Weaving the Web*. Orion.

Decker, S.; Melnik, S.; van Harmelen, F.; Fensel, D.; Klein, M.; Broekstra, J.; Erdmann, M.; and Horrocks, I. 2000. The semantic web: The roles of XML and RDF. *IEEE Internet Computing* Sept–Oct:63–74.

Embury, S., and Gray, P. 1995. The Declarative Expression of Semantic Integrity in a Database of Protein Structure. In Illaramendi, A., and Díaz, O., eds., *Data Management Systems: Proceedings of the Basque International Workshop on Information Technology (BIWIT 95)*, 216–224. San Sebastían, Spain: IEEE Computer Society Press.

Erwig, M., and Lipeck, U. 1991. A Functional DBPL Revealing High Level Optimizations. In Kanellakis, P., and Schmidt, J., eds., *Proceedings of the Third International Workshop on Database Programming Languages – Bulk Types and Persistent Data*, 306–321. Nafplion, Greece: Morgan Kaufmann Publishers, Inc.

Fensel, D.; Angele, J.; Decker, S.; Erdmann, M.; Schnurr, H.-P.; Studer, R.; and Witt, A. 2000a. Lessons learned from applying ai to the web. *international Journal of Cooperative Information Systems* 9(4):361–382.

Fensel, D.; Lassila, O.; van Harmelen, F.; Horrocks, I.; Hendler, J.; and McGuinness, D. L. 2000b. The semantic web and its languages. *IEEE Intelligent Systems*.

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.

Gray, P. M. D.; Cui, Z.; Embury, S. M.; Gray, W. A.; Hui, K.; and Preece, A. D. 1998. An Agent-Based System for Handling Distributed Design Constraints. In Boddy, M., and Gini, M., eds., *Proceedings of Agents'98 Workshop on Agent-Based Manufacturing*. Minneapolis, USA: Dept. of Comp. Science and Eng., Univ. of Minnesota.

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 Pring Symposium Technical Report SS-99-03.

Gray, P.; Kulkarni, K.; and Paton, N. 1992. *Object-Oriented Databases: a Semantic Data Model Approach*. Prentice Hall Series in Computer Science. Prentice Hall International Ltd.

Heflin, J., and Hendler, J. 2000. Dynamic ontologies on the web. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, 443–449. Menlo Park, CA: AAAI Press.

Hui, K., and Gray, P. M. D. 2000. Developing finite domain constraints – a data model approach. In *Proceedings of the 1st Internatinal Conference on Computational Logic (CL2000)*, 448–462. Springer-Verlag.

Hui, K. 2000. *Knowledge Fusion and Constraint Solving in a Distributed Environment*. Ph.D. Dissertation, University of Aberdeen.

Kemp, G.; Robertson, C.; Gray, P.; and Angelopoulos, N. 2000. CORBA and XML: Design Choices for Database Federations. In Lings, B., and Jeffery, K., eds., *Advances in Databases: Proc. BNCOD17 Conference*, 191–208. Springer-Verlag(LNCS1832).

Kulkarni, K., and Atkinson, M. 1986. EFDM: Extended Functional Data Model. *The Computer Journal* 29(1):38–46.

Landers, T., and Rosenberg, R. 1982. An Overview of MULTIBASE. In Schneider, H.-J., ed., *Distributed Data Bases*. North-Holland Publishing Company.

Preece, A.; Hui, K.; Gray, A.; Marti, P.; Bench-Capon, T.; Cui, Z.; and Jones, D. 2001. KRAFT: An agent architecture for knowledge fusion. *Inernational Journal of Cooperative Information Systems* 10(1 & 2):171–195.

Reddi, S. 1993. Integrity Constraint Enforcement in the Functional Database Language PFL. In Worboys, M., and Grundy, A., eds., *Proceedings of the 11th British National Conference on Databases*, 238–257. Keele, U.K.: Springer-Verlag.

Reeves, D.; Grosof, B.; Wellman, M.; and Chan, H. 1999. Toward a declarative language for negotiating executable contracts. In *Artificial Intelligence for Electronic Commerce: Papers from the AAAI-99 Workshop*. Menlo Park, CA: AAAI Press.

Shepherd, A., and Kerschberg, L. 1984. Prism: a knowledge based system for semantic integrity specification and enforcement in database systems. In Yormark, B., ed., *SIGMOD 84 Conference*, 307–315. Boston: ACM Press.

Shipman, D. 1981. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems* 6(1):140–173.

Torrens, M., and Faltings, B. 1999. Smart clients: constraint satisfaction as a paradigm for scaleable intelligent information systems. In *Artificial Intelligence for Electronic Commerce: Papers from the AAAI-99 Workshop*. Menlo Park, CA: AAAI Press.

Urban, S. 1989. ALICE: an Assertion Language for Integrity Constraint Expression. In *Proceedings of Conference on Computer Software Applications*.

Van Gelder, A., and Topor, R. 1991. Safety and translation of relational calculus queries. *ACM Transactions on Database Systems* 16:235–278.

Appendix

A complete example constraint in XML-CIF is shown below, together with the corresponding Colan version.

Colan version:

```
constrain all p in pc
  to have name(has_os(p)) <> "win95"
```

XML-CIF version:

```
<rdf:RDF
  xmlns:rdf="http://www.w2.org/1999/
    02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w2.org/2000/
    01/rdf-schema#"
  xmlns:cif="http://www.aktors.org/cif#"
  xmlns:pc_config="http://www.aktors.org/
    domain/pc_config#">

  <cif:impliesconstr ID="qc1">

    <!--p in pc-->
    <cif:qvar>
      <cif:setmem>
        <cif:var>
          <cif:variable ID="#p">
            <cif:varname>p</cif:varname>
          </cif:variable>
        </cif:var>
        <cif:set>
          <cif:entset>
            <cif:entclass>
              http://www.aktors.org/
                domain/pc_config#pc
            </cif:entclass>
          </cif:entset>
        </cif:set>
      </cif:setmem>
    </cif:qvar>

    <cif:if>
      <cif:boolconst>
        <cif:constname>True</cif:constname>
      </cif:boolconst>
    </cif:if>

    <cif:then>
      <cif:body>
        <cif:impliesconstr>
        <!--o in has_os(p)-->
          <cif:qvar>
            <cif:setmem>
              <cif:var>
                <cif:variable ID="#o">
                  <cif:varname>o</cif:varname>
                </cif:variable>
              </cif:var>
              <cif:set>
                <!--o=has_os(p)-->
                <cif:mvfncall>
                  <cif:prop>
                    <cif:fname>has_os</cif:fname>
                    <cif:rdfname>
                      http://www.aktors.org/
                        domain/pc_config#has_os
                    </cif:rdfname>
                  </cif:prop>
                  <cif:arg>
                    <cif:variable about="#p"/>
                  </cif:arg>
                </cif:mvfncall>
              </cif:set>
            </cif:setmem>
          </cif:qvar>
          <cif:if>
            <cif:boolconst>
              <cif:constname>True</cif:constname>
            </cif:boolconst>
          </cif:if>
          <cif:then>
            <cif:body>
              <cif:impliesconstr>

                <!--n=name(o)-->
                <cif:qvar>
                  <cif:setmem>
                    <cif:var>
                      <cif:variable ID="#n">
                      </cif:variable>
                      <cif:varname>n</cif:varname>
                    </cif:var>
                    <cif:set>
                      <cif:mvfncall>
                        <cif:prop>
                          <cif:fname>name</cif:fname>
                        </cif:prop>
                        <cif:arg>
                          <cif:variable about="#o"/>
                        </cif:arg>
                      </cif:mvfncall>
                    </cif:set>
                  </cif:setmem>
                </cif:qvar>
                <cif:if>
                  <cif:boolconst>
                    <cif:constname>True</cif:constname>
                  </cif:boolconst>
                </cif:if>
                <cif:then>
                  <!--n<>"win95"-->
                  <cif:comparison>
                    <cif:op1>
                      <cif:variable about="#n"/>
                    </cif:op1>
                    <cif:op2>
                      <cif:stringmet>
                        <cif:value>win95</cif:value>
                      </cif:stringmet>
                    </cif:op2>
                    <cif:operator>
                      <![CDATA[ <> ]]>
                    </cif:operator>
                  </cif:comparison>
                </cif:then>
              </cif:impliesconstr>
            </cif:body>
          </cif:then>
        </cif:impliesconstr>
      </cif:body>
    </cif:then>
  </cif:impliesconstr>
</rdf:RDF>
```