Extending SWRL to Express Fully-Quantified Constraints

Craig McKenzie, Peter Gray, and Alun Preece

University of Aberdeen, Department of Computing Science
Aberdeen AB24 3UE, UK
{cmckenzie,pgray,apreece}@csd.abdn.ac.uk
http://www.csd.abdn.ac.uk/research/akt/cif

Abstract. Drawing on experience gained over a series of distributed knowledge base and database projects, we argue for the utility of an expressive quantified constraint language for the Semantic Web logic layer. Our Constraint Interchange Format (CIF) is based on classical range-restricted FOL. CIF allows the expression of invariant conditions in Semantic Web data models, but the choice of how to implement the constraints is left to local reasoners.

We develop the quantified constraint representation as an extension of the current proposal for a Semantic Web Rule Language (SWRL). An RDF syntax for our extended CIF/SWRL is given in this paper. While our approach differs from SWRL in that existential quantifiers are handled explicitly rather than using OWL-DL constructs, we believe our proposal is still fully compatible with the use of the various OWL species as well as RDFS.

We demonstrate the use of the CIF/SWRL representation in the context of a practical Semantic Web reasoning application, based on the CS AK-Tive Space demonstrator (the 2003 Semantic Web Challenge winner). We indicate where in our application it makes sense to use the existing SWRL directly, and where our CIF/SWRL allows more complex constraints to be expressed in a natural manner.

1 Introduction and Motivation

Over the course of several projects, we have developed an approach to knowledge fusion in open, distributed environments [2, 10, 11]. The central idea in our approach is, in response to some user's request, to gather pertinent data from multiple network sources, along with constraints on how the data can be used. These data and constraints are then fused by mediator software into a dynamically-composed constraint satisfaction problem (CSP), which is then dispatched to a solver on the network. The solutions (if any) are then relayed back to the user. The data and constraints are expressed against a semantic data model/ontology because it may be necessary to transform them at run-time; for example, entities/classes in the data model may need to be mapped (rewritten) from a local schema/ontology to a common interchange schema/ontology. Constraints

in our approach are represented using an expressive quantified constraint language — the Constraint Interchange Format (CIF) — based on classical range-restricted first-order logic, and derived from the Colan/Daplex constraint/query languages [1].

This approach has been applied chiefly to e-commerce problems, where some package goods need to be assembled and configured to meet some requirement, and each component of the package puts constraints on the other components. For example, in configuring a personal computer system for a user, the choice of operating system may constrain the choice of peripherals; or, in configuring a package holiday, the choice of excursions may constrain the timing of the trip.

In recent work we have applied this approach to the Semantic Web [6, 12]. In many ways, this "new generation" Web is an ideal environment for the kind of problem-solving activity we envisaged in our earlier work, as it fits the W3C's vision of a task-oriented Web "better enabling people and computers to work in cooperation"[1]. The lower layers of the Semantic Web architecture (RDF/RDFS) fit our minimal requirements for data to be expressed against a semantic data model, while the ontology layer (OWL) allows far richer modelling, and also supports some elements of ontology mapping (for example, *equivalentClass*).

The current leading proposal for a representation at the Semantic Web logic layer is the Semantic Web Rule Language (SWRL)[2]. While we embrace this proposal, we will argue in this paper that it is not sufficiently expressive for our own needs, and we therefore propose an extension to SWRL that allows the representation of the kinds of fully-quantified constraints used in our earlier versions of CIF. In doing this, we bring forward some aspects of an earlier RDF-compatible encoding of CIF [12] and align this with the constructs of SWRL, to create a layered CIF/SWRL representation (with an accompanying RDF syntax). We will make arguments for allowing forms of logic more expressive than current SWRL on the open Semantic Web, and also discuss how a representation based on range-restricted FOL (which takes a closed-world assumption) does not in practice contradict the vision of an open-world Web.

To illustrate our approach situated in the Semantic Web context, later sections of the paper introduce a new application which builds on the CS AKTive Space demonstrator (winner of the 2003 Semantic Web Challenge) [13]. The CS AKTive Space is a large-scale repository of semantic metadata on computing science activities in the UK; our application — AKTive Workgroup Builder — uses constraints to select individuals from the CS AKTive Space to form working groups that satisfy particular requirements. This could be used, for example, to form "expert panels", suggest partners for collaborative projects, or organise workshops. We will show how "vanilla SWRL" and CIF/SWRL are both useful for the AKTive Workgroup Builder.

The paper is organised as follows: Section 2 discusses various forms of rule used in database and knowledge-based systems, and compares these forms with the kind of fully-quantified constraints used in our approach. Section 3 introduces

---

[1] http://www.w3.org/2001/sw/

[2] http://www.w3.org/Submission/SWRL/

the proposed CIF/SWRL representation, including its abstract and RDF syntaxes, aligned with those of SWRL. Section 4 presents our illustrative AKTive Workgroup Builder application, and highlights examples of the use of SWRL and CIF/SWRL in this demonstrator. Section 5 discusses issues arising from our approach, and Section 6 concludes with some pointers to ongoing and future work.

2   Rules and Constraints: How they Differ

People use the word "rules" rather freely. In fact there are a variety of different kinds which need to be distinguished. We discuss these below, with examples from SWRL and from our own work using Colan and Daplex [1]: derivation rules, rewrite rules, event-condition-action (ECA) rules, and quantified constraints.

Derivation Rules Derivation rules are the simplest form. They are essentially a rule for calculating a derived value on-the-fly, often by some kind of table lookup in a database, They are a technology often used to provide *views* of stored data in databases. For example, we have a rule for a person's uncle (adapted from the SWRL proposal, and using the informal SWRL syntax where `?x` denotes a variable, and adding explicit universal quantifiers):

$(\forall?x,?p,?s,?g)$ `hasParent(?x,?p)` $\land$ `hasSibling(?p,?s)` $\land$
`hasSex(?s,?g)` $\land$ `(?g='male')` $\Rightarrow$ `hasUncle(?x,?s)`

In our Daplex language, originally used to define integration schemas for heterogeneous distributed databases on the Multibase project, we would define the relationship functionally thus:

```
define hasUncle(P in Person) ->> Person
Sb in hasSibling(hasParent(P)) such that hasSex(Sb) = 'male'
```

Here a predicate `rel(X,Y)` is replaced by `Y in relfunc(X)`, where `relfunc` is a function whose values may be stored or computed. In this example it computes the set of those siblings of the parent of `P` who are male. The set is computed by the function hasUncle, and it may, of course, be empty for some individuals. The functional form helps to make clear the functional dependency of the derived information.

Rewrite Rules These rules are useful in query optimisation, for replacing one expression by an equivalent expression, usually involving less database access. For example in our Antibody Protein database [8] we have:

```
with common C in chain, i in integer
rewrite r in residues(C) such that pos(r) = i
into absolutepos(C,i)
```

This replaces a sequential search down a protein chain for the ith residue in sequence by a direct lookup of the residue using a precomputed table-driven function `absolutepos`. In FOL we could write this as:

```
(∀?c,?r,?i) Chain(?c) ∧ hasResidue(?c,?r) ∧ hasPos(?r,?i)
    ⇒ hasAbsolutePos(?c,?i,?r)
```

It looks the same as a derivation rule but it is used differently. The implication should really be replaced by ⇔ (*is logically equivalent to*). This means that we can substitute an occurrence of the body, which has particular expressions denoting values for ?c, ?i and ?r, by the head having used the same expressions in place of ?c, ?i and ?r. Having done this substitution, we can then do algebraic simplification and further substitutions. Consequently the final formula may look very different from the original one. Thus we do not execute a chain of rules at runtime, instead we execute them at compile time and compile away various unneeded computations, which is a very powerful query optimisation technique.

Event-Condition-Action (ECA) Rules The rewrite rule used above relies on the correctness of a stored table, relating residue to position in a protein chain. Fortunately, such relationships do not change, except slowly by evolution, and then we would be referring to a different chain. If we were referring to something more dynamic, like pre-booked seats in a passenger aircraft, then we would need a mechanism to update our stored relationship and keep it in correspondence with changes in passenger bookings for a given flight. This can be done by ECA rules (sometimes called triggered rules). For example:

```
ON Death of Passenger P
WHERE P isBookedOnFlight F and (Other Conditions...)
DO RemoveBooking(P,F).
```

As is well known, combinations of ECA rules can interact in unpredictable ways through side-effects in their state changing actions. A neat way of overcoming this is to code-generate the rules so that they satisfy *invariant constraints* that must be maintained under update [3]. This allows us to keep the declarative stance of pure logic, despite using rules with state changing actions. We shall now review such constraints.

Quantified Constraints A quantified constraint [4] is not just an isolated condition. It is a formula of FOL, where all the free variables have universal or existential (or maybe numerical) quantifiers. Such formulae are very suitable for expressing domain-specific semantics for collections of stored data. For example, suppose it is a rule that all tutors with "research" status in some department only supervise students with computing grades above 60. In FOL this is expressible as:

```
(∀?t,?s,?g) Tutor(?t) ∧ hasStatus(?t,'research') ∧
    supervises(?t,?s) ∧ hasSubjectGrade(?s,'Computing',?g) ⇒ (?g>60)
```

This is easily representable in SWRL, since it is a conjunctive DataLog query. However, the interesting question is how it is interpreted pragmatically. It is not really worth using it to infer an inequality about the value of ?g when using a

database, since we can just ask the database for the actual value of `?g` directly! Instead, it is more about keeping consistency of groups of values stored in a database, so that we can compile away certain checks that would otherwise be made at runtime.

In order to maintain the validity of the constraint efficiently, we code generate a number of ECA rules which are triggered by changes in supervisors' status, or creation of new supervisees, and which make only those checks that are necessary for data to be valid for the given incremental change. The creation of these rules can be done systematically from a knowledge of the constraints and the schema, as explained in [3], following original ideas in [9]. Better still, if new constraints are added, or old ones retracted, then the ECA rules can be updated accordingly — an example of automatic maintenance which is far superior to that of relying on collections of hand-coded checks and triggers installed over time by a mix of different programmers.

Thus the lesson for the Semantic Web is that quantified constraints may have alternative implementations. It should be possible to send a constraint across the Web, and to allow the remote site to process or implement the constraint as it thinks best. If it is an equational constraint the site's processing engine might enforce it by a derivation rule which always calculates the derived property value according to the constraint equation. If the derived property depends on accessing many other stored data values (a large "join", in database terms) then it may be better to cache the derived value and use triggered ECA rules to keep it up to date. There is much literature on how to do this efficiently. We feel that this approach is in keeping with the spirit of the Web, by granting the local site autonomy to choose *how* to implement the constraint, while the constraint itself guarantees *what* is being held invariant.

Using Mixed Quantifiers in Quantified Constraints Quantified Constraints need not be pure DataLog; they may wish to conclude the existence of some fact or the truth of some relationship(association). In the example above we established that research tutors should only supervise bright students, with grades above 60. However, the constraint leaves open the possibility that a research tutor supervises no students. To make a stronger statement we need an existential quantifier on the right-hand side, in the conclusion, as below:

```
(∀?t) Tutor(?t) ∧ hasStatus(?t,'research') ⇒
  (∃?s,?g) supervises(?t,?s) ∧ hasSubjectGrade(?s,'Computing',?g) ∧
    (?g>60)
```

This is a straightforward piece of range-restricted FOL, but it is not DataLog. The existential quantifier is represented by a Skolem function of the enclosing quantified variable `?t`, and thus the formula in conjunctive normal form is no longer function free; it includes terms like `supervises(?t,Student(?t))` which are not allowed in DataLog.

In our experience with using the Colan data sublanguage [1, 4], in a number of projects [3, 6, 11], we have found the need to express constraints of the form:

```
(∀?x,?y) P1(?x) ∧ q1(?x,?y) ∧ ... ⇒
   (∀?r,?s) P2(?r) ∧ q2(?x,?s) ∧ ... ⇒
      (∃?u,?v) Pn(?u) ∧ qn(?y,?v) ∧ ...
```

Thus there are some universally quantified implications, followed by a conjunction of predicates, possibly existentially quantified. There is a special case with no universal quantifiers followed just by existential quantifiers, asserting a minimum cardinality for some entity type or relationship type. We have not found the need for further universal quantifiers inside the existential quantifiers. Our constraint syntax is recursive, and allows for this possibility, but we have not found the need for it.

Constraint Interchange Format (CIF) Following our experience with Colan in the KRAFT and Conoise projects, we proposed [12] a constraint interchange format that was based purely on RDF and RDFS, which was expressive enough to encode constraints of the form above. It was based purely on range-restricted FOL with the usual connectives (and, or, not). The intention was that constraints could be passed to a constraint logic solver or theorem prover, or Prolog solver for Horn clauses with function symbols. We could have included Description Logic, but found that the data model provided by RDFS was perfectly adequate for our uses. This is because we were basically doing A-Box reasoning, relying on the presence of assertions and instances, rather than T-Box reasoning without them.

We defined constraint types recursively by the following BNF[3]:

```
<Constraint> ::=  <ImpliesConstraint> | <ExistsConstraint> | <unQuantifiedBody>
<ImpliesConstraint> ::= Each <Var> in <Entity> [SuchThat <BoolExp>] <Constraint>
<ExistsConstraint>  ::= Some <Var> in <Entity> [SuchThat <BoolExp>] <Constraint>
<unQuantifiedBody>  ::= <BoolExp>
```

Here `<BoolExp>` expands to any well-formed formula using only variables that have been quantified in an outer construct. The Predicates in the formulae may refer to membership of a specific collection of entities or stored relationships, or may be evaluable predicates (as in Prolog). The entities and relationships have RDFS declarations. This syntax is serialisable in RDF/XML.

Compared to SWRL, this form of RDF had the virtue that it treated universal and existential quantifiers on a similar footing and was easy to parse. By contrast, we feel that SWRL has got into difficulties by trying to combine RuleML, based on DataLog which does not allow existential quantifiers in the consequent [5], with OWL which allows existential quantifiers in T-Box fashion as a *someValuesFrom* construct in an OWL DL expression. This means that existential quantifiers become a restricted special case which is much harder to parse, analyse and transform.

Our vision is in accordance with that of Section 7 of the May 2004 SWRL proposal[4], in that we want to see a Semantic Web logic language that can be

---

[3] http://www.csd.abdn.ac.uk/research/akt/cif/

[4] http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

used with a variety of reasoners. In particular, we have a vision of constraints expressed against one ontology that can be transformed, by simple homomorphism, to apply to data stored in another ontology, or a shared ontology, in which the constraints can then be combined with constraints similarly transformed from other ontologies.

3   Extending SWRL to CIF/SWRL

In designing a re-formulation of CIF we undertook to incorporate SWRL constructs where possible, while also striving to simplify the original CIF syntax. Constraints are essentially defined as quantified implications, so we re-use the implication structure from SWRL, but allow for nested quantified implications within the consequent of an implication. The innermost-nested implication will have an empty body as it is always of the form "*true* ⇒ ...". In line with the presentation of SWRL, we first introduce an informal, human-readable syntax, then present the formal abstract syntax and finally the RDF serialisation.

The human-readable syntax is straightforward, as it simply adds the quantifiers and supports nested implications, where the innermost has an empty body:

```
(∀?x∈X, ?y∈Y) p(?x,?y) ∧ Q(?x) ⇒
  (∀?z∈Z) q(?x,?z) ∧ R(?z) ⇒
    (∃?v∈V) s(?y,?v)
```

Abstract Syntax Here we focus on the extensions to the abstract syntax given in SWRL and OWL documentation, using the same EBNF syntax. A `constraint` structure retains the `URIreference` and `annotation` features from OWL/SWRL so as to allow statements to be made about the constraints themselves (see Section 5). Note that nesting is handled by extending the original SWRL grammar, allowing a `constraint` to appear recursively inside a `consequent`. The definition of `antecedent` is unchanged from SWRL and appears here only for completeness. As defined by the SWRL EBNF, an `atom` may be a unary (class) predicate (for example, `P(I-variable(x))` or a binary (property) predicate (for example, `q(I-variable(y) I-variable(z))`). The only other significant new piece of syntax is the `quantifiers` structure, a list of individual quantifier expressions, each of which contains a reference to a SWRL `I-variable` and an OWL/RDFS class. So, in the informal expression "?x ∈ X" x is an `I-variable` and X is an OWL/RDFS class identifier.

We have simplified the original CIF syntax to have just one generic form of constraint, which may have a mixture of quanitifiers in any order desired. In practice this greatly flattens the nested structures.

```
constraint  ::= 'Implies(' [ URIreference ] { annotation }
                quantifiers antecedent consequent ')'
antecedent  ::= 'Antecedent(' { atom } ')'
```

```
consequent  ::= 'Consequent(' constraint | { atom } ')'
quantifiers ::= 'Quantifiers(' { q-atom } ')'
q-atom      ::= quantifier '(' q-var q-set ')'
quantifier  ::= 'forall' | 'exists'
q-var       ::= I-variable
q-set       ::= classID
```

Here is the informal example re-cast into the abstract syntax. Note the empty antecedent in the innermost-nested implication.

```
Implies(
    Quantifiers(forall(I-variable(x) X) forall(I-variable(y) Y))
    Antecedent(p(I-variable(x) I-variable(y)) Q(I-variable(x)))
    Consequent(
        Implies(
            Quantifiers(forall(I-variable(z) Z))
            Antecedent(q(I-variable(x) I-variable(z)) R(I-variable(z)))
            Consequent(
                Implies(
                    Quantifiers(exists(I-variable(v) V))
                    Antecedent()
                    Consequent(s(I-variable(y) I-variable(v))))))))
```

RDFS Syntax (Sketch) Rather than present the full, verbose RDFS XML definitions for our additional CIF syntax, here we merely sketch the necessary extensions to the SWRL RDF syntax:[5]

- We define a new `rdfs:Class` `cif:Constraint`, with two attached properties `cif:hasQuantifiers` and `cif:hasImplication`. The range of the former is an RDF list (of quantifier structures in practice) and the range of the latter is a `ruleml:Imp`.
- We define the parent class `cif:Quantifier` with two sub-classes: `cif:Forall` and `cif:Exists`. Two properties `cif:var` and `cif:set` complete the implementation of the `q-atom` from the abstract syntax. The range of both is an RDF resource: in the case of `cif:var` this will be a URIref to a SWRL variable, while for `cif:set` it will identify an OWL/RDFS class.
- Note that the SWRL RDF syntax allows the `body` of an implication to be any RDF list, so it already allows the nested inclusion of a `cif:Constraint`.

Example CIF/SWRL constraints in RDF syntax are shown in the next section.

4 Illustrative Application: AKTive Workgroup Builder

As a test-bed for our reformulated CIF we sought a Semantic Web application that was practical, used real-world RDF data, and in which realistic constraints

---

[5] Definition of an XML syntax for CIF/SWRL, extending the SWRL/RuleML XML syntax, would be trivial. We do not cover this here, preferring the pure-RDF approach to Semantic Web structure encoding.

could be expressed. The AKT project (of which the CIF work is a part) had already developed the CS AKTive Space application (winner of the 2003 Semantic Web Challenge) [13]. The CAS includes a very large repository of RDF data covering computing science research in the UK. We therefore decided that the task of dynamically composing "workgroups" from the pool of available computing science academic staff suited our criteria for a test-bed application.

The process of constructing a workgroup involves several steps:

- Defining constraints about the nature of the workgroup; for example, defining the minimum and maximum group size, the focus of the workgroup, etc.
- Gathering the RDF data about the pool of people to be considered.
- Understanding and reasoning against the data to determine eligibility; for example, is a person available to participate, do they have the relevant skills/interests, etc.
- Finally, using a constraint satisfaction problem solver to compose workgroups that satisfy the constraints.

Our AKTive Workgroup Builder (AWB) could be used, for example, to form "expert panels", suggest partners for collaborative projects, or organise workshops.

In its current form, the AWB does not directly import its data from the CAS for several reasons. Most fundamentally, the data in the CAS repository is expressed against the AKT Portal Ontology[6], which is OWL Full. The lack of reasoning support for OWL Full led us to produce a restricted reformulation of the core of the AKT Portal Ontology in OWL Lite, allowing us to use the software included in HPs Jena toolkit[7] to perform ontological inference and reasoning. The other problems in directly using the CAS data related to scalability and data provenance. The sheer size of the full repository (currently over 10 million RDF statements covering over 2000 people and their related projects, publications and other activities) made direct use of this data in developing the AWB very difficult. Because the data is harvested from the open Web/Semantic Web, it inevitably contains contradictions, errors, duplications, incompleteness, and other provenance issues. These problems of scale and trust, while being interesting and exciting problems in their own right, were tangential to our current scope of work.

For the initial version of the AWB, then, we elected to focus on data relating to the members of the five partner groups in the AKT project: Aberdeen, Edinburgh, Open University, Sheffield and Southampton. Essentially, this cut-down AWB would allow us to compose working groups for our own activities. So, the data in the current AWB repository can best be regarded as a locally-cached subset of the full CAS data store, expressed against an OWL Lite cut-down version of the AKT Portal Ontology. The instance data is almost identical to the original, while the ontology definitions are mostly just weakenings of the OWL Full original.

---

[6] http://www.aktors.org/ontology/portal
[7] http://jena.sourceforge.net

The AWB is implemented as a J2EE application with Jena managing the RDF processing, and MySQL as the back-end DBMS. Since the reasoning usually takes longer than a user is prepared to wait with a Web application (even with the cut-down dataset), the AWB uses a messaging mechanism in order to "call back" the user when the results are ready for inspection.

Reasoning in the AWB: SWRL The entailments generated by the Jena Owl Reasoner used by the AWB are all class or property based derivations. For example, a `Professor-In-Academia` is a sub-class of `Person`, therefore all instances of `Professor-In-Academia` are also `Person`s. Similarly, a `Person` with the property `has-supervisor` must be the more specific sub-class of `PhD-Student` as dictated by the domain of this property (only the `PhD-Student` class has the `has-supervisor` property).

For our reasoning to cover realistic situations we needed more expressivity. For example, if someone has published a paper on Machine Learning, this implies that they have an interest in this area, even if they have not explicitly stated it in their research interests. The fact cannot be derived from a class or property hierarchy. While the Jena reasoner API has its own internal form for inference rules, it was important to us to state these derivation rules on the Semantic Web using SWRL. To illustrate, suppose we are interested in deriving the ontology property `has-base-location` for a person, from the rule: "if a person has an affiliation with an organisation, and that organisation has a postal address with a city then this implies that the person has a base location of the same city". In FOL:

```
(∀?p,?u,?a,?c) Person(?p) ∧ Organisation(?u) ∧
  has-affiliation(?p,?u) ∧ has-postal-address(?u,?a) ∧
    address-city(?a,?c) ⇒ has-base-location(?p,?c)
```

In SWRL RDF syntax this looks as shown in Figure 1. Note that the ontology URI is represented by the entity "&akt;" all classes in this reduced version of the AKT Portal Ontology are OWL classes, defined only with OWL Lite constructs.

Reasoning in the AWB: CIF The above example works well because it uses ontology properties of an individual to derive further ontological information about that same individual and uses only the universal quantifier. However, trying to express an existentially quantified sentence (from which we wish to form a constraint) in SWRL is more awkward. For example, if we wanted to say, "every workgroup must contain at least 1 member who is a Professor":

```
(∀?g∈Workgroup)(∃?p∈Professor-In-Academia) has-member(?g,?p)
```

In the SWRL document it is shown how this kind of existentially-quantified statement can be expressed implicitly using the OWL DL `someValuesFrom` construct as part of a class restriction on the `Workgroup` class. As we argued in Section 2, we prefer to express all the quantifiers uniformly and explicitly, and leave the reasoner the option of transforming the constraint expressions to a

```
<swrl:Imp>
 <swrl:body rdf:parseType="Collection">
  <swrl:ClassAtom>
   <swrl:classPredicate rdf:resource="&akt;#Person"/>
   <swrl:argument1 rdf:resource="#p"/>
  </swrl:ClassAtom>
  <swrl:ClassAtom>
   <swrl:classPredicate rdf:resource="&akt;#Organization"/>
   <swrl:argument1 rdf:resource="#u"/>
  </swrl:ClassAtom>
  <swrl:IndividualPropertyAtom>
   <swrl:propertyPredicate rdf:resource="&akt;#has-affiliation"/>
   <swrl:argument1 rdf:resource="#p"/>
   <swrl:argument2 rdf:resource="#u"/>
  </swrl:IndividualPropertyAtom>
  <swrl:IndividualPropertyAtom>
   <swrl:propertyPredicate rdf:resource="&akt;#has-postal-address"/>
   <swrl:argument1 rdf:resource="#u"/>
   <swrl:argument2 rdf:resource="#a"/>
  </swrl:IndividualPropertyAtom>
  <swrl:IndividualPropertyAtom>
   <swrl:propertyPredicate rdf:resource="&akt;#address-city"/>
   <swrl:argument1 rdf:resource="#a"/>
   <swrl:argument2 rdf:resource="#c"/>
  </swrl:IndividualPropertyAtom>
 </swrl:body>
 <swrl:head rdf:parseType="Collection">
  <swrl:IndividualPropertyAtom>
   <swrl:propertyPredicate rdf:resource="&akt;#has-base-location"/>
   <swrl:argument1 rdf:resource="#p"/>
   <swrl:argument2 rdf:resource="#c"/>
  </swrl:IndividualPropertyAtom>
 </swrl:head>
</swrl:Imp>
```

Fig. 1. SWRL RDF/XML for the rule: "if a person has an affiliation with an organisation, and that organisation has a postal address with a city then this implies that the person has a base location of the same city".

```
<cif:Constraint>
 <cif:hasQuantifiers rdf:parseType="Collection">
  <cif:Forall>
   <cif:var rdf:resource="#g"/>
   <cif:set rdf:resource="&akt;#Workgroup"/>
  </cif:Forall>
  <cif:Exists>
   <cif:var rdf:resource="#p"/>
   <cif:set rdf:resource="&akt;#Professor-In-Academia"/>
  </cif:Exists>
 </cif:hasQuantifiers>
 <cif:hasImplication>
  <swrl:Imp>
   <swrl:body rdf:parseType="Collection"/>
   <swrl:head rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
     <swrl:classPredicate rdf:resource="&akt;#has-member"/>
     <swrl:argument1 rdf:resource="#g"/>
     <swrl:argument2 rdf:resource="#p"/>
    </swrl:IndividualPropertyAtom>
   </swrl:head>
  </swrl:Imp>
 </cif:hasImplication>
</cif:Constraint>
```

Fig. 2. RDF/XML for the constraint, "every workgroup must contain at least 1 member who is a Professor".

suitable implementation form (for example, a rewriting of the constraint to use an OWL DL reasoner would be possible in this case). So CIF/SWRL makes it possible to represent both the quantifiers in a uniform and explicit way. The CIF/SWRL for this constraint in RDF syntax is shown in Figure 2. Note the empty `body`.

We will now examine a more complex constraint, that would be far more cumbersome and unintuitive to capture in SWRL/OWL DL alone, and also illustrates how solving in CIF can interplay with rule-based reasoning in SWRL. Consider the constraint, "any workgroup with at least 5 members must contain people from different sites". For this we use the derived property `has-base-location` from our SWRL example to indicate a persons "site". The FOL is:

$$(\forall?g \in \text{Workgroup}) \text{ has-size}(?g,?s) \land (?g{\geq}5) \Rightarrow$$
$$(\exists?p1,?p2 \in \text{Person}) \text{ has-member}(?g,?p1) \land \text{has-base-location}(?p1,?b1) \land$$
$$\text{has-member}(?g,?p2) \land \text{has-base-location}(?p2,?b2) \land (?b1{\neq}?b2)$$

What makes this example interesting is that ordering and positioning of the quantifiers can be retained and the meaning does not need to be compromised to be able to write this in CIF/SWRL. The RDF/XML for this example is shown in Figure 3; note the nested implication where the recursive nesting of `Constraint` structures terminates with an implication where the `body` is empty.

```
<cif:Constraint>
 <cif:hasQuantifiers rdf:parseType="Collection">
  <cif:Forall>
   <cif:var rdf:resource="#g"/>
   <cif:set rdf:resource="&akt;#Workgroup"/>
  </cif:Forall>
 </cif:hasQuantifiers>
 <cif:hasImplication>
  <swrl:Imp>
   <swrl:body rdf:parseType="Collection">
    <swrl:IndividualPropertyAtom>
     <swrl:classPredicate rdf:resource="&akt;#has-size"/>
     <swrl:argument1 rdf:resource="#g"/>
     <swrl:argument2 rdf:resource="#s"/>
    </swrl:IndividualPropertyAtom>
    <swrl:DatavaluedPropertyAtom>
     <swrl:propertyPredicate rdf:resource="&swrlb;#greaterThanOrEqual"/>
     <swrl:argument1 rdf:resource="#s"/>
     <swrl:argument2 rdf:datatype="&xsd;#int">5</swrl:argument2>
    </swrl:DatavaluedPropertyAtom>
   </swrl:body>
   <swrl:head rdf:parseType="Collection">
    <cif:Constraint>
     <cif:hasQuantifiers rdf:parseType="Collection">
      <cif:Exists>
       <cif:var rdf:resource="#p1"/>
       <cif:set rdf:resource="&akt;#Person"/>
      </cif:Exists>
      <cif:Exists>
       <cif:var rdf:resource="#p2"/>
       <cif:set rdf:resource="&akt;#Person"/>
      </cif:Exists>
     </cif:hasQuantifiers>
     <cif:hasImplication>
      <swrl:Imp>
       <swrl:body rdf:parseType="Collection"/>
       <swrl:head rdf:parseType="Collection">
        . . . details of body omitted: simply a list of </swrl:IndividualPropertyAtom>s . . .
       </swrl:head>
      </swrl:Imp>
     </cif:hasImplication>
    </cif:Constraint>
   </swrl:head>
  </swrl:Imp>
 </cif:hasImplication>
</cif:Constraint>
```

Fig. 3. RDF/XML for the constraint, "any workgroup with at least 5 members must contain people from different sites".

5   Discussion

In our previous work, we have shown how the solving of CIF constraints can be implemented by dynamically composing the constraints and available data instances into a constraint satisfaction problem, code-generated for use with a particular finite domain solver [7, 6][8]. This approach works well with the CIF constraints, which are range-restricted FOL. Of course we are making a closed world assumption here, at the time the finite domain CSP is composed, and this might seem contradictory to the general vision of an open world Semantic Web (and specifically the open world assumption underpinning OWL DL). In practice, there is no contradiction: a finite number of candidate instances are always available at run-time, whether gathered from a local cache (as in the current AWB) or acquired through some wider search (which is always "best-effort" on the Web). As explained in Section 2, this is because we are essentially doing A-Box reasoning, relying on the presence of assertions and instances, rather than T-Box reasoning without them.

As we said before, our approach is not incompatible with the use of other reasoning mechanisms (in fact, part of our motivation for doing this work is to explore the interplay of multiple reasoners — see Section 6). For example, OWL DL class restrictions can usefully be employed in CIF expressions to specify the domains of variables, both in the quantifier expressions (as the value of a `cif:set` property) and within the heads and bodies of the implications (allowed by the abstract syntax in Section 3 as unary-predicate `atom`s). We have yet to properly explore the computational complexities arising from this usage, however, or to come to a point where we can recommend "best practice".

However, an important point about the original design of CIF, which is retained in the reformulation presented here, is that it is perfectly feasible to use CIF with only RDFS data models. This is true of SWRL as well, although of course SWRL has no way to handle existential quantification without OWL DL constructs — not a problem for CIF. Given the relatively much wider use of RDFS than OWL on the current Semantic Web (Dublin Core, RSS, vCards, and FOAF[9] are among the most widely-instantiated Semantic Web schemas) we feel this makes CIF immediately useful for practical applications.

One part of the proposed CIF syntax that we have not explored in this paper, chiefly for reasons of space, is the representation of disjunction and negation. These are allowed in CIF implication heads and bodies, although we are still experimenting with various forms to avoid over-complexity in the RDF/XML syntax. Again, we also aim ultimately to move to a point where we can recom-

---

[8] Solvers used to date include ECLiPSe (`http://www.icparc.ic.ac.uk/eclipse/`) and the Sicstus Prolog FD library (`http://www.sics.se/isl/sicstus/`)

[9] Technically FOAF (Friend-of-a-Friend) is an OWL Full ontology; however, there is only very limited use of OWL constructs in FOAF term definitions, chiefly to indicate which properties can be used a unique identifiers for instances. Users of FOAF need little or no real understanding of OWL to use it, which perhaps explains its rapid uptake; see, for example: `http://www.plink.org`

mend "best practice" in the use of these more elaborate forms of constraint; we believe more research is needed generally in this area.

As a final point, it is worth noting that the `URIreference` and `annotation` features from OWL/SWRL allow statements to be made about constraints. This supports the usual authorship and provenance information to be attached to constraints, which is always useful, but also allows other kinds of metadata specific to the usage of constraints. For example, we may attach properties indicating the "strength" of the constraint — is it *hard* or *soft* (that is, can it be relaxed?), or are there particular exceptional conditions under which it may become hard or soft? In general, we have an interest in using constraint reification in the solving process [2], where it becomes useful to reason about which constraints are currently satisfied and which are not, and to use techniques such as negotiation and argumentation to relax (or in some cases harden) constraints. We hope that the the `URIreference` and `annotation` features will be a useful mechanism in constraint reification.

6   Conclusion & Future Directions

In this paper, we have proposed a representation for fully-quantified constraints at the Semantic Web logic layer, in the form of an extension to the implication constructs available in the SWRL proposal. We illustrated the use of the CIF/SWRL constraints alongside SWRL derivation rules in a practical application: the AKTive Workgroup Builder. Work on the AWB is ongoing; currently we are employing three forms of reasoning:

– Jena is used to perform OWL Lite reasoning at the ontology level, as part of the task of assembling candidate instances for the solving process.
– We are experimenting with the use of Hoolet[10] to implement the SWRL derivation rules.
– Through a PrologBeans interface we are harnessing the SICStus Prolog finite domain constraint solver.

Looking ahead, our interest lies in combining these various mechanisms into a practical hybrid reasoning suite, and in exploring complexity and scalability trade-offs.

Related work within the AKT project at Aberdeen covers other constraint-solving approaches and support tools. For example, we are applying constraint-satisfaction to configuration design, and are developing a constraint editor to allow end-users to express CIF constraints.[11]

---

[10] `http://owl.man.ac.uk/hoolet/`
[11] Details are available at: `http://www.csd.abdn.ac.uk/research/akt/`

References

1. N. Bassiliades and P.M.D Gray. CoLan: a Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering*, 14:203–249, 1994.
2. S. Chalmers, A. D. Preece, T. J. Norman, and P. Gray. Commitment management through constraint reification. In *3rd International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004)*, 2004.
3. S.M. Embury and P.M.D. Gray. Compiling a Declarative, High-Level Language for Semantic Integrity Constraints. In R. Meersman and L. Mark, editors, *Database Application Semantics: Proceedings of 6th IFIP TC-2 Working Conference on Data Semantics*, pages 188–226, Atlanta, USA, May 1995. Chapman and Hall.
4. P. M. D. Gray, S. M. Embury, K. Hui, and G. J. L. Kemp. The evolving role of constraints in the functional data model. *Journal of Intelligent Information Systems*, 12:113–137, 1999.
5. B. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proceedings of the Twelfth International World Wide Web Conference*, pages 48–57. ACM, 2003.
6. K. Hui, S. Chalmers, P. Gray, and A. Preece. Experience in using RDF in agent-mediated knowledge architectures. In L. van Elst, V. Dignum, and A. Abecker, editors, *Agent-Mediated Knowledge Management (LNAI 2926)*, pages 177–192. Springer-Verlag, 2004.
7. K. Hui, P. Gray, G. Kemp, and A. Preece. Constraints as mobile specifications in e-commerce applications. In R. Meersman, K. Aberer, and T. Dillon, editors, *Semantic Issues in e-Commerce Systems*, pages 327–341. Kluwer, 2003.
8. G. Kemp, P. Gray, and A. Sjöstedt. Rewrite rules for quantified subqueries in a federated database. In L. Kerschberg and M. Kafatos, editors, *Proceedings of the Thirteenth International Conference on Scientific and Statistical Database Management*, pages 134–143. IEEE Computer Society Press, 2001.
9. J.-M. Nicolas. Logic for Improving Integrity Checking in Relational Databases. *Acta Informatica*, 18:227–253, 1982.
10. T. J. Norman, A. D. Preece, S. Chalmers, N. R. Jennings, M. M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian. CONOISE: Agent-based formation of virtual organisations. In *Research and Development in Intelligent Systems XX*, pages 353–366. Springer-Verlag, 2003.
11. A. Preece, K. Hui, A. Gray, P. Marti, T. Bench-Capon, Z. Cui, and D. Jones. KRAFT: An agent architecture for knowledge fusion. *Inernational Journal of Cooperative Information Systems*, 10(1 & 2):171–195, 2001.
12. A. Preece, K. Hui, and P. Gray. An FDM-based constraint language for semantic web applications. In P. Gray, L. Kerschberg, P. King, and A. Poulovassilis, editors, *Agent-Mediated Knowledge Management (LNAI 2926)*, pages 417–434. Springer-Verlag, 2004.
13. N. Shadbolt, N. Gibbins, H. Glaser, S. Harris, and m. c. schraefel. CS AKTive Space, or how we learned to stop worrying and love the semantic web. *IEEE Intelligent Systems*, pages 41–47, May/June 2004.