

Handling Soft Constraints in the Semantic Web Architecture

Alun Preece

apreece@csd.abdn.ac.uk

Stuart Chalmers

schalmer@csd.abdn.ac.uk

Craig McKenzie

cmckenzie@csd.abdn.ac.uk

Jeff Z. Pan

jpan@csd.abdn.ac.uk

Peter Gray

pgray@csd.abdn.ac.uk

Department of Computer Science, University of Aberdeen, Aberdeen AB24 3UE, UK

<http://www.csd.abdn.ac.uk/research/akt/cif/>

ABSTRACT

In this paper we present a proposal for representing soft CSPs within the Semantic Web architecture. The proposal is motivated by the need for a service-providing agent to reason about its commitments as soft constraints. The two essential requirements addressed are: the need to associate utility values with constraints, to reflect the relative importance of satisfying them, and the need to make statements about which constraints are satisfied and violated by a given solution. The proposal builds upon previous work in defining a Semantic Web Constraint Interchange Format (CIF), which itself builds on the proposed Semantic Web Rule Language (SWRL). The main contribution of this paper is a new ontology for representing soft CSPs; we also extend the previous form of CIF/SWRL. The soft CSP ontology is intended to be used with CIF/SWRL, but is also potentially usable with other constraint and rule representations.

1. INTRODUCTION

Constraint satisfaction is an important type of reasoning, with broad applicability in the Semantic Web context. Examples include extending the definitions of concepts in Web ontologies, in a similar way to rules [8], representing and reasoning about capabilities of Semantic Web services [15], and supporting information integration through the interchange of constraints and data [12].

Constraints are often *soft*: they do not have to be satisfied for a solution to be valid or acceptable [4]. Instead, the goal of the constraint-solving procedure becomes to find an optimal solution that satisfies a maximal subset of the constraints [6]. In this context, constraints often have associated *utility values*, indicating the relative importance of satisfying individual constraints or clauses [2, 7]. Importantly, these utilities are generally not absolute: they are relative to the particular constraint satisfaction problem (CSP) in which the constraint is being applied. In relation to a particular solution, a given constraint may be satisfied or violated, and it is often useful to be able to represent and reason about which constraints are satisfied/violated by a given solution [5]. The ability to make statements about

whether a constraint is satisfied or not in a given context is commonly called constraint reification.

A class of applications where the handling of soft constraints is a key issue is that of commitment management for service provisioning in virtual organisations. In these applications — commonly seen in domains such as e-commerce [14], e-science [16], and e-response¹ — a service-provider manages particular resources, and commits these resources to meet specific goals. Often, the commitment of resources to goals is governed by service-level agreements. The commitments can be modelled as constraints on the resources, and commitments managed as a soft CSP. When a service-provider is presented with a new potential commitment, it must perform reasoning to determine if it can take on this commitment, possibly by dropping (breaking) existing lower-utility commitments.

In this paper we present a proposal for representing soft CSPs within the Semantic Web architecture, using the commitment management scenario as motivation. The proposal builds upon previous work in defining a Semantic Web Constraint Interchange Format (CIF) [13], which itself builds on the proposed Semantic Web Rule Language (SWRL) [10]. This paper extends the previous form of CIF/SWRL, and proposes a new ontology for representing soft CSPs which is intended to complement CIF/SWRL (but is also potentially usable with other constraint and rule representations).

The paper is organised as follows: Section 2 presents an abstract scenario involving an agent reasoning about its commitments using constraint solving, motivating the need to represent utility values and constraint reification; Section 3 describes our SWRL-based Constraint Interchange Format; Section 4 surveys approaches to handling soft and reified constraints in various CSP-solving frameworks; Section 5 introduces an ontology for representing soft CSPs; Section 6 provides discussion and conclusion.

2. MANAGING COMMITMENTS AS CONSTRAINTS

To illustrate the use of soft constraints for modelling and managing commitments, we now present a detailed example. This example is a simplification of the type of problem that occur in the virtual organisation service-provisioning appli-

Copyright is held by the author/owner(s).

WWW2006, May 22–26, 2006, Edinburgh, UK.

¹<http://e-response.org/>

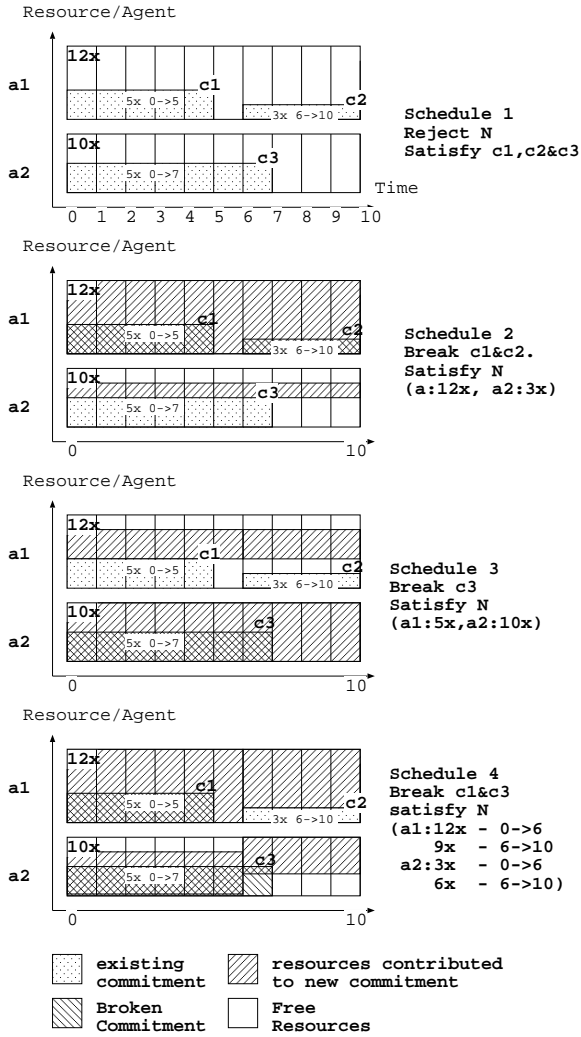


Figure 1: Agent a1 & a2’s options for providing new commitment N

cation domains alluded to in the introduction (see also [14]).

Consider two service-providing agents, a1 and a2. Each agent can provide a certain amount of resource x (12 units from a1 and 10 from a2). The agents have existing commitments — c1, c2 and c3 on those resources, as shown in the first schedule in Figure 1:

- c1: 5x from time 0→5 on a1
- c2: 3x from time 6→10 on a1
- c3: 5x from time 0→7 on a2

Note that in this simple example we only look at a single type of resource (x). However, the solution to the commitment management problem presented here generalises to any number of resource types and combination [5]. We restrict ourselves to a single resource type here only for the sake of clarity.

If a new request, N is received by the agents to provide 15x from time 0→10, then the agent has four main choices:

- Reject N and satisfy existing commitments c1, c2 & c3 (Schedule 1 in Figure 1)

- Accept N and break c1 & c2 (Schedule 2)
- Accept N and break c3 (Schedule 3)
- Accept N and break c1 & c3 (Schedule 4)

(Note that there are many permutations of the exact amounts of the resource x , but in terms of commitments satisfied or broken these are the four main choices.)

As the number of agents and commitments increases the number of possible combinations of solutions that satisfy all the commitments (and solutions that break commitments) grows exponentially. Also the number of trivial solutions (i.e. solutions that vary in extremely small detail) increases (e.g. schedule 3 could take 7x from a1 and 8x from a2 rather than 5x and 10x which would not affect the commitments broken). The main emphasis behind the CSP-solving procedure is to find solutions that break commitments (i.e. solutions that are different enough in outcome that they break different commitments). As a result of this we need to equip the CSP solver with a method for differentiating between solutions. We also need a way to prioritise commitments so that we can rule out solutions that break commitments that have been specified *a priori* as ‘must-complete’ tasks.

The commitment management system is implemented as a reification extension to a cumulative scheduling CSP solver that uses a combination of reification and constraint value labeling to provide the required commitment management and prioritisation — details are provided in [5].

3. A CONSTRAINT INTERCHANGE FORMAT BASED ON SWRL

Our Constraint Interchange Format (CIF) is based on the Colan [1] constraint language, which is based on range restricted first order logic. (The term “constraint” is often used rather freely; in this paper we use the term for logical expressions within the scope of Colan — see [13] for broader discussion of the relationship between rules and constraints.) Earlier versions of the language were aligned with RDF [11] and SWRL [13]. CIF 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. An example CIF constraint is shown in human-readable SWRL-style syntax below:

$$\begin{aligned}
 (\forall ?x \in X, ?y \in Y) p(?x, ?y) \wedge Q(?x) \Rightarrow \\
 (\forall ?z \in Z) q(?x, ?z) \wedge R(?z) \Rightarrow \\
 (\exists ?v \in V) s(?y, ?v)
 \end{aligned}$$

Commitment c2 from the example in Section 2 can be written in this syntax as follows:

$$\begin{aligned}
 (\forall ?t \in \text{Time}) ?t \geq 6 \wedge ?t \leq 10 \Rightarrow \\
 (\exists ?c \in \text{Commitment}) \text{hasService}(?c, ?s) \wedge \\
 \text{hasServiceType}(?s, 'x') \wedge \text{hasAmount}(?s, 3)
 \end{aligned}$$

Compared to the SWRL syntax in [10], this simply adds the quantifiers and supports nested implications. Note that the innermost-nested implication has an empty body as it is always of the form “*true* \Rightarrow ...”. In the above syntax this is implicit; the following abstract and RDF syntaxes make this explicit.

Figure 2 shows the CIF extensions to the abstract syntax given in SWRL and OWL documentation [10], using

```

constraint ::= 'Implies(' [ URIreference ] { annotation }
            quantifiers antecedent consequent ')'
antecedent ::= 'Antecedent(' { expr } ')'
consequent ::= 'Consequent(' consequent ')'
consexpr  ::= constraint | { atom }
expr      ::= atom | disjunct | conjunct | negation
disjunct  ::= 'Or(' { expr } ')'
conjunct  ::= 'And(' { expr } ')'
negation  ::= 'Not(' expr ')'
quantifiers ::= 'Quantifiers(' { q-atom } ')'
q-atom    ::= quantifier '(' q-var q-set ')'
quantifier ::= 'forall' | 'exists'
q-var     ::= I-variable
q-set     ::= description

```

Figure 2: CIF/SWRL abstract syntax in EBNF

the same EBNF syntax. We refer to this form of CIF as CIF/SWRL. A `constraint` retains the `URIreference` and `annotation` syntax features from 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 extended from SWRL to allow combinations of disjunction, conjunction, and negation expressions. In the simplest case where an antecedent is a conjunction of atoms, the syntax allows omission of an explicit `And` structure — the “and” is implicit (as in the SWRL syntax). However, disjunctions and negations are always explicit, as are any conjunctions within them. It is worth noting that a consequent can be only a conjunction — CIF/SWRL does not allow disjunction or negation here.

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 description. So, in the informal expression “ $?x \in X$ ” `x` is an `I-variable` and `X` is an OWL/RDFS class identifier. In more complex cases, the OWL description may be a restriction or other more elaborate expression allowed by the OWL syntax.

This new syntax also differs from an earlier version published in [13] by allowing disjunction and negation in the antecedents, and any OWL description as the value of a `q-set`.

The informal example re-cast into the abstract syntax is shown in Figure 3. Note the empty antecedent in the innermost-nested implication.

3.1 CIF/SWRL RDF Syntax Summary

To support publishing and interchange of CIF constraints in the Semantic Web context, we provide an RDF/XML syntax as an extension to the one given for SWRL. The full RDF Schema for the CIF/SWRL syntax is available at the project website²; here we merely summarise the necessary extensions to the SWRL RDF syntax:

- We define a new `rdfs:Class Constraint`, with properties `hasQuantifiers` and `hasImplication`. The range

²<http://www.csd.abdn.ac.uk/research/akt/cif/>

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 `Quantifier` with two subclasses: `Forall` and `Exists`. Two properties `var` and `set` complete the implementation of the `q-atom` from the abstract syntax. The range of both is an RDF resource: in the case of `var` this will be a `URIref` to a SWRL variable, while for `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 `Constraint`.
- Finally, we define `OrExpression` and `AndExpression` as sub-classes of `rdfs:List`, and a `Negation` class that has a `swrl:argument1` property to point to the negated atom.

The RDF/XML for the constraint `c2` is shown in Figure 4.

4. REPRESENTING SOFT CONSTRAINTS IN CSP SYSTEMS

Before presenting our soft CSP ontology, we examine common features of soft CSPs in the literature and in practical implementations, in order to identify the minimal features required of the ontology.

Soft constraints can be represented and implemented in a variety of ways, depending on language and system used. In this section we look at a number of CSP-solving frameworks (based on Prolog and Java), and describe ways in which we can model soft constraints using the features available in those frameworks. We then give a brief overview of some of the soft constraint literature.

4.1 Prolog Implementations

In many Prolog implementations, the issue of soft constraints can be modelled with reification. Reification is the attachment of a boolean value to each constraint. If a constraint is satisfied, then the boolean value is set to true, otherwise it is set to false. This means that it is possible to reason about the constraints, by reasoning about these boolean values.

Given an unsatisfiable problem, the aim then is to find the best subset of simultaneously satisfiable constraints (i.e. true values), by utilising the attached boolean values.

These values themselves can then form the basis for a meta-level CSP, the solution to which is an assignment of reification values to constraints at the lower level. SICS-tus³, GNU Prolog⁴ and SWI Prolog⁵ all provide a system of reification.

4.2 Java Implementations

In Java, two dominant constraint libraries are Java Constraint Library (JCL)⁶ and Choco⁷.

The JCL attaches a floating point number to each tuple of a constraint rated from 0.0 (important) to 1.0 (not important), so each outcome pairing is given a value showing

³<http://www.sics.se>

⁴<http://gnu-prolog.inria.fr/>

⁵<http://www.swi-prolog.org/>

⁶<http://liawww.epfl.ch/JCL/>

⁷<http://choco.sourceforge.net/>

```

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)))))))))

```

Figure 3: Example constraint shown in the CIF/SWRL abstract syntax

```

<cf:Constraint rdf:about="#c2" />
<cf:hasQuantifiers rdf:parseType="Collection" >
  <cf:Forall>
    <cf:var rdf:resource="#t" />
    <cf:set rdf:resource="&schedule;#Time" />
  </cf:Forall>
</cf:hasQuantifiers>
<cf:hasImplication>
  <swrl:Imp>
    <swrl:body rdf:parseType="Collection" />
    <swrl:DatavaluedPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&swrlb;#greaterThanOrEqual" />
      <swrl:argument1 rdf:resource="#t" />
      <swrl:argument2 rdf:datatype="&xsd;#int" />6</swrl:argument2 />
    </swrl:DatavaluedPropertyAtom>
    <swrl:DatavaluedPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&swrlb;#lessThanOrEqual" />
      <swrl:argument1 rdf:resource="#t" />
      <swrl:argument2 rdf:datatype="&xsd;#int" />10</swrl:argument2 />
    </swrl:DatavaluedPropertyAtom>
  </swrl:body>
  <swrl:head rdf:parseType="Collection" >
    <cf:Constraint>
      <cf:hasQuantifiers rdf:parseType="Collection" >
        <cf:Exists>
          <cf:var rdf:resource="#c" />
          <cf:set rdf:resource="&schedule;#Commitment" />
        </cf:Exists>
      </cf:hasQuantifiers>
    <cf:hasImplication>
      <swrl:Imp>
        <swrl:body />
        <swrl:head rdf:parseType="Collection" >
          <swrl:IndividualPropertyAtom>
            <swrl:classPredicate rdf:resource="&schedule;#hasService" />
            <swrl:argument1 rdf:resource="#c" />
            <swrl:argument2 rdf:resource="#s" />
          </swrl:IndividualPropertyAtom>
          <swrl:IndividualPropertyAtom>
            <swrl:classPredicate rdf:resource="&schedule;#hasServiceType" />
            <swrl:argument1 rdf:resource="#s" />
            <swrl:argument2 rdf:resource="&service;#x" />
          </swrl:IndividualPropertyAtom>
          <swrl:IndividualPropertyAtom>
            <swrl:classPredicate rdf:resource="&schedule;#hasAmount" />
            <swrl:argument1 rdf:resource="#c" />
            <swrl:argument2 rdf:datatype="&xsd;#int" />3</swrl:argument2 />
          </swrl:IndividualPropertyAtom>
        </swrl:head>
      </swrl:Imp>
    </cf:hasImplication>
  </cf:Constraint>
</swrl:head>
</swrl:Imp>
</cf:hasImplication>
</cf:Constraint>

```

Figure 4: RDF/XML for the constraint (commitment) c2

its preference as a solution. When solutions are returned from the solver they are given a ‘score’ dependent on what tuple has been chosen. These may be used to prioritise the solutions dependent on preferences.

This method can easily model the reification described in the Prolog systems. If we add ‘0’ to each domain of possible values for each variable, we can class this as a ‘not applied’ value for that variable (i.e. if the variable is assigned to 0, we take it to be not satisfied). We can mark a constraint tuple where an assigned value is 0 as 1.0 (i.e. not important), and other possible values as anywhere between 0.0 to 0.9; therefore the preference will be to find a value other than 0 for that constraint (i.e. satisfy the constraint). Obviously this requires some work-arounds when zero value assignments are required for specific values, but in the case of the commitment management examples we have been investigating, this method has proved satisfactory.

Choco is a system for solving constraints, also written in Java. It is a library for constraint satisfaction problems (CSPs), constraint programming (CP) and explanation-based constraint solving that is built upon an event-based propagation mechanism. The type of constraints that can be handled by Choco are arithmetic constraints (equality, difference, comparisons and linear combination), boolean and user-defined N-ary constraints. The propagation engine maintains arc-consistency for binary constraints throughout the solving process, while for n-ary constraints, it uses a weaker propagation mechanism with a forward checking algorithm. Choco uses a system of explanation based solving⁸. Using this method, a constraint program can describe why certain decisions were taken (i.e. why variable x cannot take the value a) and so show why a problem fails. This information can then be used to find subsets of satisfiable constraints within the given set.

4.3 Soft Constraints In the Literature

A number of people in the literature look at the scoring, or ordering, of constraints in CSP solving in two main ways [2]:

- Assigning values to each possible tuple in a constraint.
- Assigning a value to the actual constraint itself.

There are a number of ways that these two methods are modelled. Fuzzy CSPs [7] allow constraint tuples to have an associated preference (1 = best, 0 = worst). Again, as described in the Java Constraint Library section, we can still model (and have modelled) partial CSPs using this method, by adding a tuple with 0 values to the domain of possible values, and assigning this a ‘1.0’ preference (i.e. worst outcome). Similarly weighted CSPs [4] assign preference, but the value given with each tuple is associated with a cost. The main factor in these types of CSP is that a value is associated with the individual tuples in a constraint, not the actual general constraint itself.

Freuder and Wallace [6] talk more in terms of the actual constraints themselves, and relaxing them. They talk about sets of solutions, rather than the actual individual solutions to each variable. They then talk about a partial ordering of solutions, where the solutions are ordered by a given distance metric.

5. DEVELOPING THE CSP ONTOLOGY

We were interested in developing a well formed means of representing a set of one (or more) constraints that, when combined, form a single (soft) CSP, the ultimate goal being to facilitate interchange of information between a CSP problem constructor and an appropriate solver. The solver would process the problem and return to the constructor zero or more solutions, each solution identifying those constraints that are satisfied and those that are violated by that solution. This would then allow the CSP constructor to decide itself which solution to select.

As discussed in Section 4, soft CSP solvers typically allow each constraint to be assigned a *utility value*, defined as a floating point number with a value ranging from 0 to 1 inclusive. These values represent the significance, or importance, of that constraint with respect to the other constraints comprising the CSP. Essentially, this value represents the degree of softness of each constraint, with a higher number implying a lower softness, and therefore a greater desirability to satisfy that constraint. However, depending upon the strategy employed by the CSP solver, a constraint with a lower utility value may still be satisfied in preference to violating another constraint with a higher utility value.

To help clarify these requirements, consider the following scenario from an e-response domain. A controller within an ambulance dispatch centre is trying to solve the CSP of assigning an individual ambulance to a series of emergencies (each emergency can be regarded as a constraint). The utility value of each constraint is a combined representation of the time taken for the ambulance to respond and the gravity of the situation. There are three quick trips involving transporting patients from one local hospital to another. Each of these constraints is assigned a utility value of 0.2 as they are not urgent but still desirable to free up beds. There has also been a traffic accident where someone is injured but not critically — this incident is therefore assigned a utility value of 0.5. Finally, someone has been badly burned in a house fire and is needing emergency treatment, unfortunately the location is difficult to get to and will tie up the ambulance for some time — this emergency gets assigned a utility value of 0.9. These are then given to the CSP solver to produce possible solutions of how the ambulance should respond, taking into account the utility values of each constraint, and attempting to produce the highest combined overall utility. One solution may be to only satisfy one constraint and attend to the burns victim (0.9). Another solution may lean toward satisfying more constraints and helping more people by attending the traffic accident and moving two of the three patients ($0.5 + 0.2 + 0.2 = 0.9$). The controller must then make the final decision of which solution is preferable. The key issue is that these assigned utility values are all subjective. A second controller may deem the traffic accident as more important and assign it a higher utility value, likewise he might regard the patient transfers as being more trivial and lower their utility values.

From the preceding discussion, it is clear that a utility value is not an intrinsic part of a constraint itself, rather it can be viewed as a kind of annotation on a constraint, with respect to a particular CSP (set of constraints). Similarly, the status of a constraint in terms of whether it is satisfied or not can be seen as an annotation of that constraint with respect to a particular solution. Therefore, we decided to create a separate ontology to represent a CSPs, indepen-

⁸<http://www.e-constraints.net/>

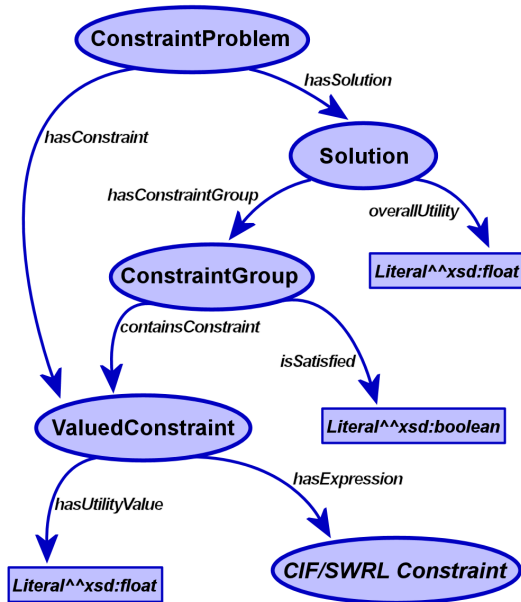


Figure 5: Graph of the OWL DL CSP ontology

dent of the (CIF/SWRL) representation of the individual constraints themselves. The following sections describe two variant representations of the CSP ontology.

5.1 CSP Ontology in OWL DL

Figure 5 is a graphical depiction of the OWL DL version of the CSP ontology (classes are drawn as ovals, primitive data types as rectangles, and properties are arcs going from the domain and pointing to the range of that property). Initially, a CSP constructor would create an instance of a **ConstraintProblem** with one, or more, instances of **ValuedConstraint**. Each **ValuedConstraint** is assigned a utility value (real number) with the actual constraint expressed using CIF/SWRL. At this point the constructor would have only a representation of the CSP itself; there would be no instances of the **Solution** class. Only once the CSP has been passed onto a solver will any instances of **Solution** be created (or not, if no solution can be found).

In order to describe the state of a **ValuedConstraint** (i.e. whether it is satisfied or not with respect to a particular **Solution** instance), this version of the ontology features a sub-class of **Solution** called **ConstraintGroup**. This class acts as a container for a set of **ValuedConstraints** involved with a particular **Solution** instance and attaches a single boolean-valued property **isSatisfied** to each member of the **ConstraintGroup**. (Enforcing this restriction of a single **isSatisfied** property is the main reason OWL DL was used to define the soft CSP ontology.)

5.2 CSP Ontology in OWL DL + SWRL

The second variation of our CSP ontology is shown in Figure 6. The motivation for this variant is to capture the direct relationships between individual solutions and constraint instances. The **ConstraintGroup** class and its boolean property **isSatisfied** are replaced by the properties **satisfies** and **violates**, both of which have domain **Solution** and range **ValuedConstraint**. Clearly, the use of these properties must be disjoint between the same instances: a given

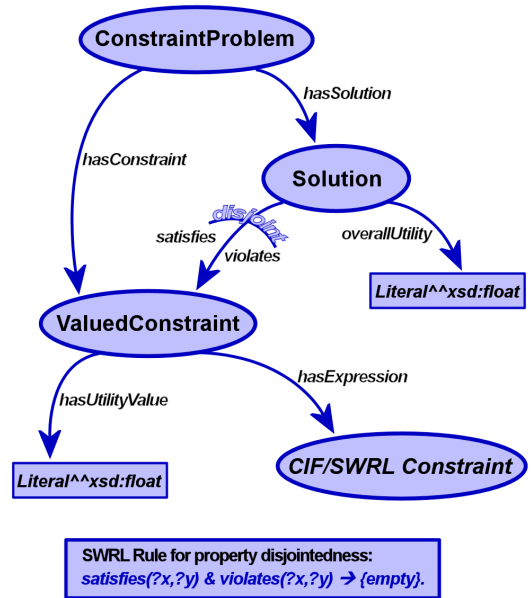


Figure 6: Graph of the CSP ontology using a SWRL rule to enforce the disjoint properties **satisfies** and **violates**

constraint can only be satisfied or violated with respect to a given solution. OWL DL does not enable us to enforce this check⁹, so we define a rule to enforce data integrity in this case. Using SWRL:

$$\text{csp:satisfies}(?x,?y) \wedge \text{csp:violates}(?x,?y) \Rightarrow \perp$$

(\perp denotes the empty consequent i.e. trivially false [10].)

In principle, both variants of the ontology can be used; however, we prefer the simplicity and directness of this second variant. In this representation, the first two solutions from Figure 1 are as follows:

```

<ex:soln1> <csp:satisfies> <ex:c1> .
<ex:soln1> <csp:satisfies> <ex:c2> .
<ex:soln1> <csp:satisfies> <ex:c3> .
<ex:soln1> <csp:violates> <ex:N> .

<ex:soln2> <csp:violates> <ex:c1> .
<ex:soln2> <csp:violates> <ex:c2> .
<ex:soln2> <csp:satisfies> <ex:c3> .
<ex:soln2> <csp:satisfies> <ex:N> .

```

While adding SWRL rules to a DL knowledge base can make inference undecidable [9], this particular rule is within the DL-safe subset of SWRL (as the disjointness is imposed on named **ValueConstraints** rather than any possible ones). Therefore, it is still possible to have decidable reasoning support for our OWL DL + SWRL version of the CSP ontology.

6. DISCUSSION AND CONCLUSION

In this paper we presented a proposal for representing soft CSPs within the Semantic Web architecture. The proposal was motivated by the need for a service-providing agent to

⁹Disjoint property axioms are expected to be available in OWL 1.1, which is still decidable: <http://www-db.research.bell-labs.com/user/pfps/owl/overview.html>

reason about its commitments as soft constraints. The two essential requirements addressed were: the need to associate utility values with constraints, to reflect the relative importance of satisfying them, and the need to make statements about which constraints are satisfied and violated by a given solution. While there exists an XML-based proposal for representing CSPs [3], to the best of our knowledge our proposal is the first CSP interchange format founded on RDF and OWL.

The proposal built upon previous work in defining a Semantic Web Constraint Interchange Format (CIF), which itself built on the proposed Semantic Web Rule Language (SWRL). This paper extended the previous definition of CIF/SWRL to allow disjunction and negation in implication antecedents, and the ability to use OWL descriptions in the scope of quantifiers. Note that, while the CSP ontology is designed to work with CIF as the constraint representation, it is conceivable that other constraint and rule representations could be used as the values of the *expression* properties of `ValueConstraints`. As work continues on standardising Semantic Web rule and constraint languages¹⁰, we will consider suitable extensions to the CSP ontology.

The SWRL FOL proposal to extend SWRL to full first-order logic¹¹ shares many of the features we earlier proposed for CIF/SWRL. While, at the time of writing, the SWRL FOL proposal lacks an RDF syntax, we anticipate it would not be hard to fully align CIF/SWRL with SWRL FOL. The main differences are in the syntactic form for the quantifier parts of expressions, a more expressive consequent (SWRL FOL allows disjunction and negation here), and a more complex syntax for simple conjunctions (SWRL FOL opts not to follow the SWRL “list format” for these).

It is worth emphasising that the CSP ontology and CIF are intended to be *interchange formats* — to solve a CSP it is necessary to translate the CSP and the individual CIF/SWRL constraints to a native format, such as one of those surveyed in Section 4. This will involve some degree of (often non-trivial) conversion and mapping of the various elements of the CSP. For example, as mentioned in Section 2 our current implementation of the commitment management system uses the Java Constraint Library and, as a result, utility values need to be mapped from the 0 (softest)...1 (hardest) scale used in the ontology (Section 5) to the 1 (softest)...0 (hardest) scale used in the JCL (Section 4). In previous work, CIF has also been translated to the native formats of the Sicstus Prolog FD library and ECLiPSe [12, 11].

Our immediate future plans lie in developing the interaction between constraint solving and the user using our two interchange formats. Using an emergency response application domain¹² as a test-bed scenario for our commitment management system, we aim to experiment with various styles of user interface to allow a user to pose a constraint satisfaction problem to a solver, receive a number of solutions with various overall utilities, and choose a preferred set of commitments.

7. ACKNOWLEDGMENTS

This work is supported under the Advanced Knowledge

¹⁰<http://www.w3.org/2005/rules/>

¹¹<http://www.daml.org/2004/11/fol/>

¹²<http://e-response.org/>

Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is funded 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. See also: <http://www.aktors.org>

The commitment management service was developed in the context of the CONOISE and CONOISE-G projects, involving the Universities of Aberdeen, Cardiff, and Southampton, and British Telecom, and funded by the DTI/Welsh e-Science Centre, and BT. See also: <http://www.conoise.org>

8. REFERENCES

- [1] N. Bassiliades and P. Gray. CoLan: a Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering*, 14:203–249, 1994.
- [2] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and Valued CSPs: Basic properties and comparison. In M. Jampel, E. Freuder, and M. Maher, editors, *Over-Constrained Systems*, pages 111–150. Springer-Verlag LNCS 1106, Aug. 1996.
- [3] F. Boussemart, F. Hemery, and C. Lecoutre. Description and representation of the problems selected for the first international constraint satisfaction solver competition. Technical report, CRIL, Université d’Artois, 2005.
- [4] K. Brown. Soft consistencies for weighted csp. In *Proceedings of Soft’03: 5th International Workshop on Soft Constraints*, Kinsale, Ireland, September 2003.
- [5] 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)*, pages 430–437, 2004.
- [6] E. C. Freuder. Partial Constraint Satisfaction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89, Detroit, Michigan, USA*, pages 278–283, 1989.
- [7] H. W. Guesgen and A. Philpott. Heuristics for solving fuzzy constraint satisfaction problems. In *2nd New Zealand Two-Stream International Conference on Artificial Neural Networks and Expert Systems (ANNES ’95), 1995.*, 1995.
- [8] S. Hawke, editor. *W3C Workshop on Rule Languages for Interoperability*, 2005. <http://www.w3.org/2004/12/rules-ws/>.
- [9] I. Horrocks and P. Patel-Schneider. A proposal for an OWL Rules Language. In *Thirteenth International World Wide Web Conference (WWW 2004)*. ACM, 2004.
- [10] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Groszof, and M. Dean. SWRL: A Semantic Web rule language combining OWL and RuleML. Technical report, W3C, 2004. <http://www.w3.org/Submission/SWRL/>.
- [11] 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.

- [12] 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.
- [13] C. McKenzie, P. Gray, and A. Preece. Expressing fully quantified constraints in CIF/SWRL. In G. Antoniou and H. Boley, editors, *Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, pages 139–154. Springer-Verlag, 2004.
- [14] 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. *Knowledge-Based Systems*, 17:103–111, 2004.
- [15] M. Nottingham and P. Le Hégarret, editors. *W3C Workshop on Constraints and Capabilities for Web Services*, 2004.
<http://www.w3.org/2004/09/ws-cc-program.html>.
- [16] J. Patel, . M. L. L. Teacy, N. R. Jennings, S. Chalmers, N. Oren, T. J. Norman, A. Preece, P. M. D. Gray, P. J. Stockreisser, G. Shercliff, J. Shao, W. A. Gray, N. J. Fiddian, and S. Thompson. Agent-based virtual organisations for the grid. In *Proc 1st International Workshop on Smart Grid Technologies*, 2005.