A Semantic Web Approach to Handling Soft Constraints in Virtual Organisations

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 constraint satisfaction problems (CSPs) within the Semantic Web architecture. The proposal is motivated by the need for a service-providing agent in a virtual organisation to reason about its commitments as soft constraints. The three essential requirements addressed are: (1) the need to have constraints express commitments in terms of Semantic Web services, (2) the need to associate utility values with constraints, to reflect the relative importance of satisfying them, and (3) 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 paper describes an ontology for representing soft CSPs and their solutions, allowing an agent's set of commitments to be expressed as a collection of soft constraints. The ontology is an open interchange format for soft CSPs, allowing commitment to be communicated and exchanged among the members of a virtual organisation.

1. INTRODUCTION

Constraint satisfaction is an important type of reasoning, with broad applicability in the Semantic Web context. Examples include representing and reasoning about capabilities of Semantic Web services [16], supporting information integration through the interchange of constraints and data [13], and extending the definitions of concepts in Web ontologies, in a similar way to rules [9].

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, 8]. 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 [15], e-science [17], and e-response[1] — a service-provider manages particular resources, and commits these resources to satisfying 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) [14], which itself builds on the proposed Semantic Web Rule Language (SWRL) [11]. 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). In the context of service provisioning in virtual organisations, the primary motivation for representing soft CSPs within the Semantic Web architecture is to make the constraint representation compatible with ontologies for Semantic Web services (SWS); it is our expectation that services over which commitments are being made are instances of existing SWS

[1] http://e-response.org/

ontologies, normally expressed in OWL or RDFS.[2]

The paper is organised as follows: Section 2 presents an abstract scenario involving an agent reasoning about it's 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 describes our virtual organisation demonstrator implementation; Section 7 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 application domains alluded to in the introduction (see also [15]).

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`: 5`x` from time 0→5 on `a1`
- `c2`: 3`x` from time 6→10 on `a1`
- `c3`: 5`x` 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 15`x` 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 7`x` from `a1` and 8`x` from `a2` rather than 5`x` and 10`x` 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.

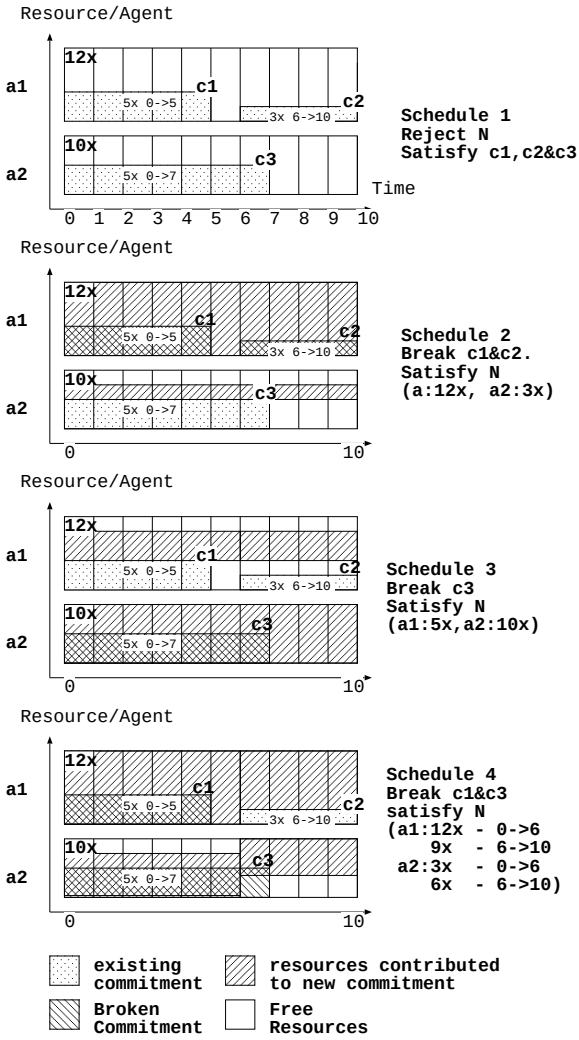

Figure 1: Agent `a1` & `a2`'s options for providing new commitment `N`

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], and further discussion our virtual organisation demonstrator implementation appears in Section 6.

[2]For example, OWL-S (http://www.daml.org/services) or WSML (http://www.wsmo.org).

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 [14] for broader discussion of the relationship between rules and constraints.) Colan expressions allow explicit universal and existential quantifiers, and nested implications. Earlier versions of the language were aligned with RDF [12] and SWRL [14]. An example CIF constraint is shown in human-readable SWRL-style syntax below:

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

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

```
(∀?t∈Time) ?t≥6 ∧ ?t≤10 ⇒
  (∃?c∈Commitment) hasService(?c,?s) ∧
    hasServiceType(?s,'x') ∧ hasAmount(?s,3)
```

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. Compared to the SWRL syntax in [11], 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* ⇒ ...". In the above syntax this is implicit; the following abstract and RDF syntaxes make this explicit. (Comparison with the SWRL FOL language is made in Section 7.)

Figure 2 shows the CIF extensions to the abstract syntax given in SWRL and OWL documentation [11], using 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

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

quantifier expressions, each of which contains a reference to a SWRL `I-variable` and an OWL description. So, in the informal expression "`?x` ∈ `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 [14] 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 website[3]; 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 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 `rdf:List`, and a `Negation` class that has a `swrl:argument1` property to point to the negated atom.

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

```
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

```
<cif:Constraint rdf:about="#c2" />
 <cif:hasQuantifiers rdf:parseType="Collection">
  <cif:Forall>
   <cif:var rdf:resource="#t" />
   <cif:set rdf:resource="&schedule;#Time" />
  </cif:Forall>
 </cif:hasQuantifiers>
 <cif: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">
    <cif:Constraint>
     <cif:hasQuantifiers rdf:parseType="Collection">
      <cif:Exists>
       <cif:var rdf:resource="#c" />
       <cif:set rdf:resource="&schedule;#Commitment" />
      </cif:Exists>
     </cif:hasQuantifiers>
     <cif: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>
     </cif:hasImplication>
    </cif:Constraint>
   </swrl:head>
  </swrl:Imp>
 </cif:hasImplication>
</cif:Constraint>
```

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

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. SICStus[4], GNU Prolog[5] and SWI Prolog[6] all provide a system of reification.

4.2 Java Implementations

In Java, two dominant constraint libraries are Java Constraint Library (JCL)[7] and Choco[8].

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 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 to be 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 constriants, 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 solving[9]. Using this method, a constraint program can describe why certain decision 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 [8] 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.

[4]http://www.sics.se
[5]http://gnu-prolog.inria.fr/
[6]http://www.swi-prolog.org/
[7]http://liawww.epfl.ch/JCL/
[8]http://choco.sourceforge.net/
[9]http://www.e-constraints.net/

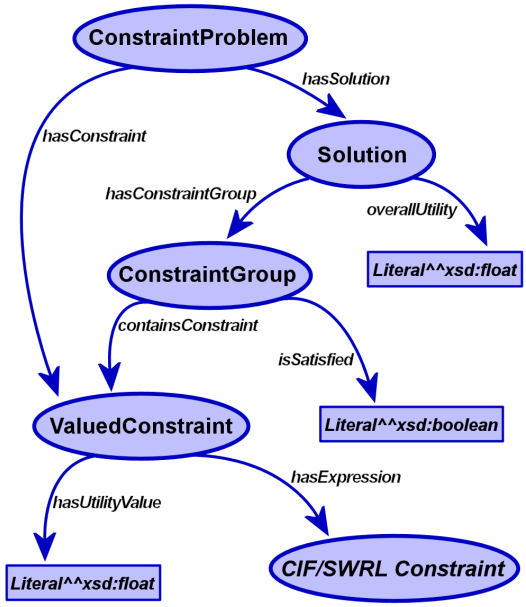

Figure 5: Graph of the OWL DL CSP ontology

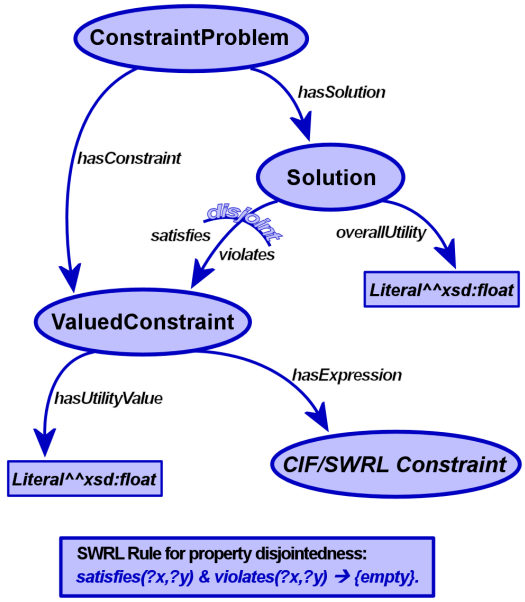

Figure 6: Graph of the CSP ontology using a SWRL rule to enforce the disjunct properties `satisfies` and `violates`

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.

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, independent 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 constraint can only be satisfied or violated with respect to a given solution. OWL DL does not enable us to enforce this check[10], so we define a rule to enforce data integrity in this case. Using SWRL:

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

($\bot$ denotes the empty consequent i.e. trivially false [11].)

[10]Disjoint 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

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 [10], this particular rule is within the DL-safe subset of SWRL (as the disjointness is imposed on named `ValueConstraint`s rather than any possible ones). Therefore, it is still possible to have decidable reasoning support for our OWL DL + SWRL version of the CSP ontolgy.

6. DEMONSTRATOR SYSTEM: MULTIMEDIA SERVICE PROVISIONING

This section presents the CSP-based commitment management system (CMS) in the context of the multimedia service provisioning demonstrator system developed as part of the Conoise-G project [17]. The decision task with which a service-providing agent is faced is shown in Figure 7. In response to a call for bids, the agent reasons about its available resources and commitments on those resources, and decides whether and what to bid. This is the core decision-making task that was detailed in Section 2. If it is already representing a virtual organisation, the available resources and existing commitments are the combined resources and commitments of the organisation. In making its deliberations, the agent has the option to seek to recruit other service providers to extend the resources it can provide; it can also opt to free-up resources by breaking existing commitments as described in Section 2.

The Conoise-G representation is built on the FIPA standard agent platform[11]. Following experience gained in previous related work [7] we employ formalisms from the Semantic Web to provide the information representations for communication among agents in Conoise-G. We chose these representations in preference to the more conventional use of SL in the content of FIPA messages for a number of reasons. Firstly, the Semantic Web representations are far more widely used than FIPA-SL, so while in principle both SL and the Semantic Web formats are equally open, in practice Conoise-G is lent greater interoperability with the rest of the distributed systems world by aligning with World Wide Web consortium recommendations. Secondly, by adopting Semantic Web representations we benefit from being able to reuse existing schemas and ontologies; for example, we borrowed heavily from the DAML-S service ontology. In creating a Conoise-G system for a particular application domain, we would be in a position to exploit any existing schemas or ontologies in that area. Thirdly, particularly at the lower (RDF) layers of the Semantic Web formalism stack, the semantics of the data model are much simpler than FIPA-SL (while still adequate for operational use), so there is less of a learning curve for designers and implementors of Conoise-G agents. (As a related point, there is also a substantial amount of well-tested software for processing RDF, unlike FIPA-SL.)

[11]http://www.fipa.org/

In the current version of the Conoise-G architecture, we have created a set of interrelated ontologies expressed in a relatively lightweight manner as RDF schemas. For now, RDFS is sufficiently expressive to capture usable structures, and has allowed us to rapidly develop the necessary message formats for inter-agent communication in the testbed application. We would envisage the definitions in the ontologies becoming tightened up with the addition of OWL statements once the formats have stabilised through further testing and refinement. Two sample RDF messages expressed using a number of the ontologies are shown in Figures 8 and 9. The first shows a sample call for bids, as issues to provider agents. This consists of an instance of a user **Requirement** structure, stating a number of services that the user's requirement *consistsOf*, and also a *qualityPreference* property, indicating that the most important thing for this user is lowest cost. The descriptions of each required service are adorned with service-specific properties, for example the **MovieContent** requirement specifies a number of movies (per month), a subscription preference, and a genre type. This example illustrates the use of terms from three of the Conoise-G ontologies:

- the `package` ontology describes service packages, defining terms such as the class **Requirement** and the property *consistsOf*;
- the `quality` ontology describes domain-independent quality-of-service terms such as the *qualityPreference* property, and its various settings such as "minCost";
- the `media` ontology defines all application domain-specific terms for the multimedia service provisioning scenario, including the service classes **MovieContent**, **HtmlContent**, **PhoneCalls**, and **TextMessaging**, all of which the ontology defines to be (indirect) sub-classes of the generic Conoise-G **ServiceProfile** class (closely based on DAML-S).

The second sample message in Figure 9 shows a bid issued by one of the service provider agents in response to the call shown in Figure 8. The bid is for just one of the required services (the **HtmlContent** part); the **Bid** structure is similar to the **Requirement** structure in that it also employs the *consistsOf* property, but here there is also an identified instance of a **Provider**, whose properties are defined using terms from the `profile` ontology (that also defines the **ServiceProfile** class mentioned above). This information will allow the user to access the service if this bid is ultimately accepted as part of the winning package. Note also that the services offered in bids have **Price** structures attached, which are rich enough to identify different price "bands" depending on the volume the user might wish to consume.

The elements of these ontologies are used in the commitment expressions. Specifically, the service types (for exam-

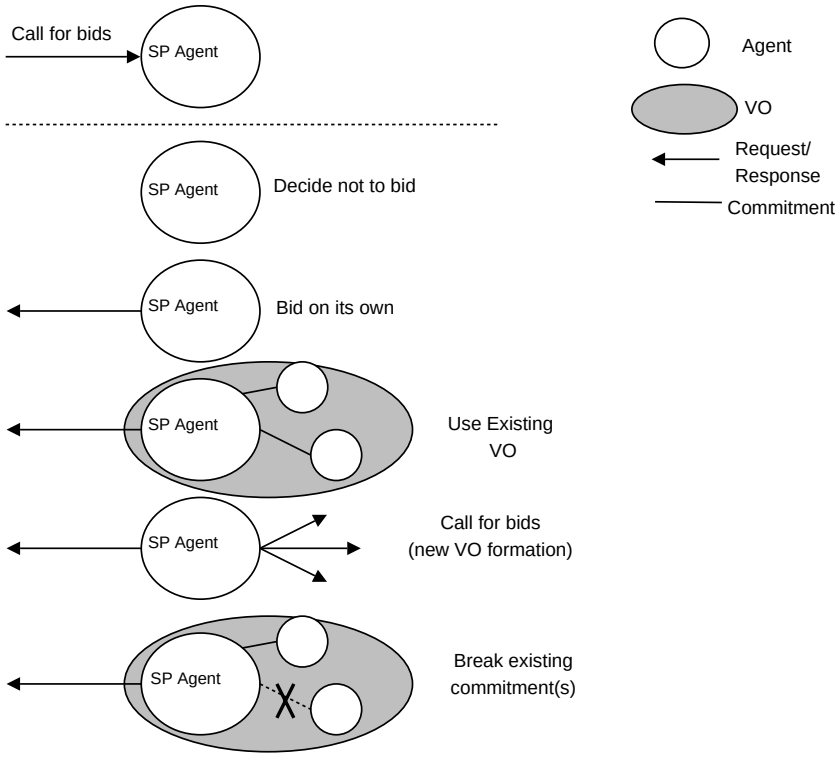

Figure 7: Decision task tackled by a service-providing agent in Conoise-G

ple, **MovieContent** or **HtmlContent**) are used to define the types of service in a commitment (in place of the abstract service `x` in our examples in Section 2 and 3), and service properties (for example, *subscriptionType* or *mediaStyle*) can be constrained in a commitment expression.

A screenshot showing the running Conoise-G demonstrator is shown in Figure 10. The composition of the virtual organisation is shown in the top-right portion of the figure. The foreground shows a simulated PDA delivering the provided package of services.

7. 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 in a virtual organisation to 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 languages[12], we will consider suitable extensions to the CSP ontology.

The SWRL FOL proposal to extend SWRL to full first-order logic[13] 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 (soft-

[12]http://www.w3.org/2005/rules/
[13]http://www.daml.org/2004/11/fol/

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:quality="http://conoise.org/ontologies/quality#"
  xmlns:media="http://conoise.org/ontologies/media#"
  xmlns:package="http://conoise.org/ontologies/package#">
  <package:Requirement rdf:about="http://conoise.org/samples/request">
    <quality:qualityPreference rdf:resource="http://conoise.org/ontologies/quality#minCost"/>
    <package:consistsOf
      rdf:type="http://conoise.org/ontologies/media#PhoneCalls"
      media:numberOfMinutes="25"/>
    <package:consistsOf>
      <media:MovieContent media:numberOfMovies="72">
        <media:subscriptionType rdf:resource="http://conoise.org/ontologies/media#monthly"/>
        <media:mediaStyle rdf:resource="http://conoise.org/ontologies/media#scienceFiction"/>
      </media:MovieContent>
    </package:consistsOf>
    <package:consistsOf>
      <media:HtmlContent media:updateFrequency="24">
        <media:mediaStyle rdf:resource="http://conoise.org/ontologies/media#news"/>
      </media:HtmlContent>
    </package:consistsOf>
    <package:consistsOf
      rdf:type="http://conoise.org/ontologies/media#TextMessaging"
      media:numberOfMessages="100"/>
  </package:Requirement>
</rdf:RDF>
```

Figure 8: Sample call for bids in RDF format, as sent to provider agents

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:media="http://conoise.org/ontologies/media#"
  xmlns:profile="http://conoise.org/ontologies/profile#"
  xmlns:package="http://conoise.org/ontologies/package#">
  <package:Bid rdf:about="http://conoise.org/samples/pa2bid#bid2">
    <package:providedBy>
      <package:Provider
        profile:fipaAddress="pa2@conoise.org:15551/JADE">
        <profile:name>Provider Agent 2</profile:name>
      </package:Provider>
    </package:providedBy>
    <package:consistsOf
      <media:HtmlContent rdf:about="http://conoise.org/samples/pa2bid#pa2news"
        media:updateFrequency="72">
        <media:mediaStyle rdf:resource="http://conoise.org/ontologies/media#news"/>
        <package:hasPriceStructure
          rdf:type="http://conoise.org/ontologies/package#Price"
          package:min="0" package:max="10" package:unitPrice="3"/>
        <package:hasPriceStructure
          rdf:type="http://conoise.org/ontologies/package#Price"
          package:min="10" package:max="50" package:unitPrice="2"/>
        <package:hasPriceStructure
          rdf:type="http://conoise.org/ontologies/package#Price"
          package:min="50" package:max="1000" package:unitPrice="1"/>
      </media:HtmlContent>
    </package:consistsOf>
  </package:Bid>
</rdf:RDF>
```

Figure 9: Sample bid in RDF format, as issued by a provider agent

Figure 10: The Conoise-G virtual organisation demonstrator system.

est)...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 [13, 12].

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 domain[14] 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.

8. ACKNOWLEDGMENTS

This work is supported by the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), funded by the UK Engineering and Physical Sciences Research Council (grant number GR/N15764/01). The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffeld, 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. We are grateful to Gareth Shercliffe and Patrick Stockreisser for their work on the Conoise-G demonstrator user interface. See also: http://www.conoise.org

9. 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 csps. 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] G. A. Grimnes, S. Chalmers, P. Edwards, and A. Preece. Granitenights: A multi-agent visit scheduler utilising semantic web technology. In *Cooperative Information Agents VI (LNAI 2782)*, pages 137–151. Springer, 2003.

[8] 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.

[9] S. Hawke, editor. *W3C Workshop on Rule Languages for Interoperability*, 2005. http://www.w3.org/2004/12/rules-ws/.

[10] I. Horrocks and P. Patel-Schneider. A proposal for an OWL Rules Language. In *Thirteenth International World Wide Web Conference (WWW 2004)*. ACM, 2004.

[11] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web rule language combining OWL and RuleML. Technical report, W3C, 2004. http://www.w3.org/Submission/SWRL/.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] M. Nottingham and P. Le Hégaret, editors. *W3C Workshop on Constraints and Capabilities for Web Services*, 2004. http://www.w3.org/2004/09/ws-cc-program.html.

[17] 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.

[14] http://e-response.org/