A Reusable Commitment Management Service using Semantic Web Technology

Alun Preece, Stuart Chalmers, and Craig McKenzie

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

Abstract. Commitment management is a key issue in service-provisioning in the context of virtual organisations (VOs). A service-provider — which may be a single agent acting within an organisation, or the VO acting as a collective whole — manages particular resources, and commits these resources to meet specific goals. Commitments can be modelled as constraints on resources. Such constraints are often soft: they can be broken if necessary. The goal of the work described in this paper is to create an open, reusable commitment management service (CMS) based on Semantic Web standards. The chief requirement is that the CMS should be reusable in different domains, able to manage commitments over services described in a wide range of domain-specific service ontologies. This paper presents open Semantic Web representations for (1) expressing individual commitments as constraints over service descriptions, (2) capturing a set of commitments as a soft constraint satisfaction problem, and (3) representing and communicating the solution to a soft CSP. A reference implementation of a constraint solver able to operate on (1) and (2) to produce (3) is described, and its reuse is demonstrated in two distinct domains: e-commerce and e-response.

1 Introduction

Commitment management is a key issue in service-provisioning in the context of virtual organisations (VOs). A service-provider — which may be a single agent acting within an organisation, or the VO acting as a collective whole — manages particular resources, and commits these resources to meet specific goals. The issue of commitment management appears throughout the lifecycle of such organisations [14]: when a partner is deliberating whether to bid to join a VO, it must consider its existing commitments, and construct a bid that is compatible with its commitments; when a VO is operating, it must manage its commitments over its collective resources and — when perturbations inevitably occur — the VO must adapt by revising its commitments; finally, when a VO's job is done and it disbands, commitments must be released and cleaned-up. Although the types of services managed in each case are very diverse, commitment management issues arise in VOs in all domains, including e-commerce [13], e-science [6], and e-response[1].

Often, the commitment of resources to goals is governed by service-level agreements. The commitments can be modelled as *constraints* on the resources. Such constraints are often *soft*: they can be broken if necessary [4]. When a service-provider is

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

presented with a new potential commitment, it must perform reasoning to determine if it can take on this commitment, possibly by dropping (breaking) existing commitments (constraints). Hence, the goal of the agent's deliberation procedure becomes to find an optimal solution that satisfies a maximal subset of the constraints [7]. 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*.

The goal of the work described in this paper is to create an open, reusable commitment management service (CMS) based on Semantic Web standards. The chief requirement is that the CMS should be reusable in different domains, able to manage commitments over services described in a wide range of domain-specific service ontologies. This requirement motivates the Semantic Web approach: it is our expectation that the majority of service ontologies will be defined in a SW-based representation, currently OWL or RDFS.[2] By rooting our CMS in the Web and Semantic Web architectures, we also exploit existing XML-based interchange formats (including RDF syntax), transport protocols (HTTP, SOAP, etc) and logical foundations (including description logic and rules). Building on these foundations, the requirements for our CMS are:

1. an open format for expressing individual commitments as constraints over service descriptions;
2. an open format for capturing a set of commitments as a soft constraint satisfaction problem;
3. an open format for representing and communicating the solution to a soft CSP;
4. a reference implementation of a constraint solver able to operate on (1) and (2) to produce (3);
5. demonstrations of the CMS working in at least two distinct domains, to provide proof-of-concept of reusability.

In light of these requirements, this paper offers the following:

- We review our Semantic Web Constraint Interchange Format (CIF) [12], which builds on the proposed Semantic Web Rule Language (SWRL) [9]. This format (CIF/SWRL) provides an open representation for expressing individual commitments as quantified constraints over service descriptions defined in terms of OWL or RDFS ontologies.
- We present an ontology for representing soft CSPs and their solutions. The ontology — which is intended to complement CIF/SWRL but is also potentially usable with other constraint and rule representations — allows utility values to be associated with constraint expressions. The solution format allows constraints to be labelled to indicate whether they are satisfied or not in a particular solution.

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

– We describe our reference implementation of a constraint solver based on the Java Constraint Library (JCL)[3], and present two demonstration systems using the CMS, one in an e-commerce domain (multimedia service provisioning) and the other in an e-response domain (disaster management).

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 the CIF/SWRL constraint interchange format; Section 4 surveys approaches to handling soft and reified constraints in various CSP-solving frameworks, and describes our ontology for representing soft CSPs; Section 5 describes the two virtual organisation demonstrator implementations; 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 occurs in virtual organisation service-provisioning application domains.

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 `7x` from `a1` and

[3] http://liawww.epfl.ch/JCL/

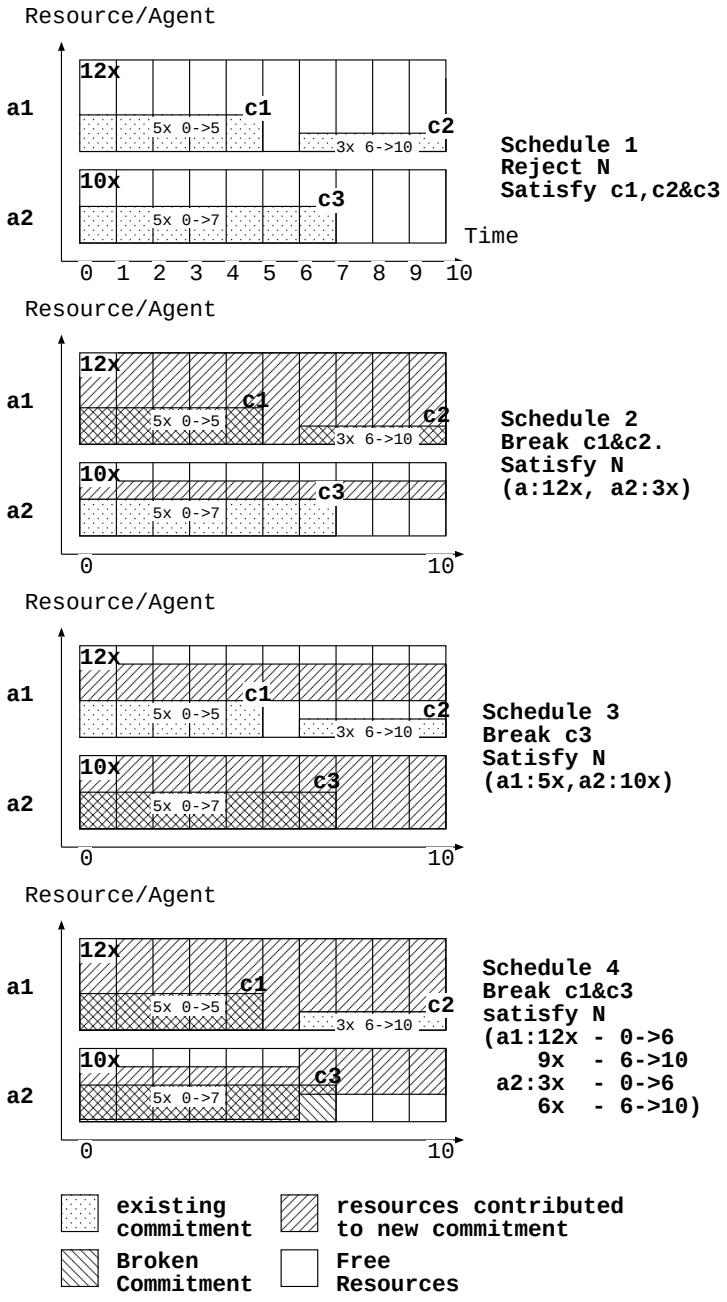

Fig. 1. Agent `a1` & `a2`'s options for providing new commitment `N`

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.

This kind of commitment management mechanism can be 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 of our virtual organisation demonstrator implementations appears in Section 5.

3 A Constraint Interchange Format Based on SWRL

Our Constraint Interchange Format (CIF) is derived from the Colan [1] constraint language, which is based on range restricted first order logic.[4] Earlier versions of the CIF language were aligned with RDF [11] and SWRL [12]. CIF constraints are essentially defined as quantified implications, for example:

```
(∀?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)
```

Unary predicates and named sets in these expressions (P, Q, X, Y, Commitment, Time, etc) are RDFS or OWL classes, while binary predicates (p, q, hasService, hasAmount, etc) are RDFS or OWL properties. When CIF is used to express commitments, most of these terms will come from domain-specific service ontologies — examples are given in Section 5. Due mainly to the addition of explicit universal and existential quantifiers, and nested implications, CIF constraints are not expressible in SWRL as it stands, so we have defined CIF/SWRL as an extension of SWRL: we reuse the implication structure from SWRL, but allow for nested quantified implications within the consequent of an implication. Compared to the SWRL syntax in [9], 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 syntax, and the RDF syntax given in the appendix make this explicit.)

Figure 2 shows the CIF extensions to the abstract syntax given in SWRL and OWL documentation [9], using the same EBNF syntax. A constraint expression retains

[4] The term "constraint" is often used rather freely; in this paper we use the term for logical expressions within the scope of Colan — see [12] for broader discussion of the relationship between rules and constraints.

the URIreference and annotation syntax features from SWRL so as to allow statements to be made about the constraints themselves (see Section 4.1). Note that nesting is handled by extending the original SWRL grammar, allowing a constraint to appear recursively inside a consequent.

```
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
```

Fig. 2. CIF/SWRL abstract syntax in EBNF

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 predicate (for example, P(I-variable(x)) or a binary predicate (for example, q(I-variable(y) I-variable(z))). The only other notable additional 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 ∈ X" x is an I-variable and X is an OWL/RDFS class identifier.

The example commitment c2 re-cast into the abstract syntax is shown in Figure 3. Note the empty antecedent in the innermost-nested implication.

The RDF syntax for CIF/SWRL is summarised in the appendix.

4 An Ontology for Representing Soft CSPs

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.

```
Implies(
    Quantifiers(forall(I-variable(t) Time))
    Antecedent(
        greaterThanOrEqual(I-variable(t) 6)
        lessThanOrEqual(I-variable(t) 10))
    Consequent(
        Implies(
            Quantifiers(exists(I-variable(c) Commitment))
            Antecedent()
            Consequent(
                hasService(I-variable(c) I-variable(s))
                hasService(I-variable(s) 'x')
                hasAmount(I-variable(s) 3)))))
```

Fig. 3. Example constraint shown in the CIF/SWRL abstract syntax

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.

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[5], GNU Prolog[6] and SWI Prolog[7] all provide a system of reification.

Java Implementations In Java, two dominant constraint libraries are Java Constraint Library (JCL)[8] and Choco[9].

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.

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

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[10]. 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.

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 [7] 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.

4.1 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

[10] http://www.e-constraints.net/

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.

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. While for our practical purposes the ontology would be mainly used to annotate CIF/SWRL constraints, in principle it should be usable with other constraint and rule representations.

Figure 4 is a graphical depiction of the OWL CSP ontology, which is expressed in OWL DL and SWRL (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).

The properties `satisfies` and `violates` are used to represent the fact that a particular solution instance satisfies a particular constraint, or not. 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[11], so we define a rule to enforce data integrity in this case.

The first two solutions from Figure 1 are represented in triple form as follows using the CSP ontology:

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

[11] 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

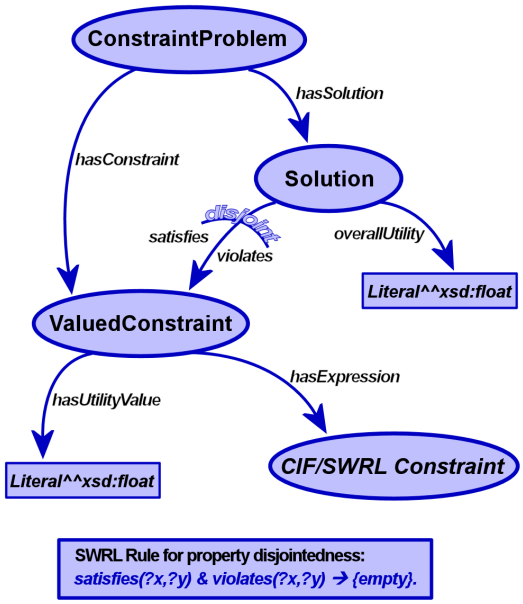

Fig. 4. Graph of the CSP ontology

```
<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 `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 ontolgy.

5 Demonstrator Systems

To demonstrate reuse of the commitment management service it has been applied in two distinct domains: e-commerce and e-response. In the first domain, the CMS has been used in the context of a multimedia service provisioning demonstrator system developed as part of the Conoise-G project [14]. A customer wishes to subscribe to a package of multimedia services for their mobile device — a screenshot from the demonstrator, including a PDA simulator, is shown in Figure 5. A service ontology defines available service types and characteristics, from which the user can select their requirements via their user-agent. These requirements are then posted to the network of service-providing agents via a yellow pages, inviting agents to bid to provide the elements of the required package.

Fig. 5. The Conoise-G virtual organisation demonstrator system.

Here, a virtual organisation is entirely agent-mediated: in response to a call for bids, an agent reasons about its available resources and commitments on those resources, and decides whether and what to bid. 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.

Commitments on resources are expressed as constraints on classes defined in a Conoise-G `media` ontology, which defines all application domain-specific terms for the multimedia service provisioning scenario. These include 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 (based on DAML-S). The Conoise-G demonstrator is built on the FIPA standard agent platform[12]; the content of all inter-agent communication is RDF. The CMS is implemented using the Java Constraint Library; RDF processing is done using Jena2[13], by means of which the RDF transport format of the CSPs is converted into the JCL native CSP format for solving.

The scenario for our second application domain — e-response — is a fictitious disaster in the city of London, UK.[14] Here, the services upon which commitments need

[12] http://www.fipa.org/
[13] http://www.hpl.hp.com/semweb/jena2.htm
[14] Details are available at: http://e-response.org/

to be managed are physical entities such as fire engines, ambulances, police units, etc. Like the multimedia scenario, these are defined in a domain-specific service ontology, and CIF/SWRL constraints express commitments over them (for example, "commit 10 fire engines to a fire incident at Bartholemew's Hospital, from 10am, for an estimated duration of 5 hours"). A key difference to the e-commerce scenario is that this is human-mediated: human decision-makers need to be presented with possible commitment management solutions for them to make informed choices. This requires that the CMS be interfaced with the Compendium issue-mapping software that provides the main user interface, illustrated in Figure 6.

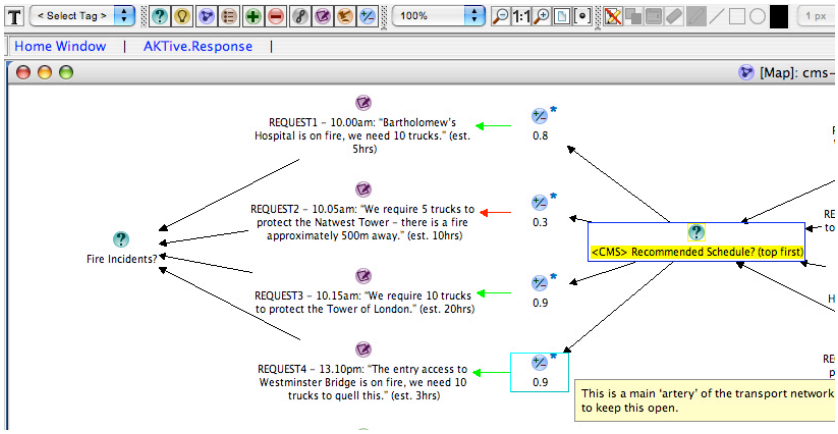

Fig. 6. The e-response virtual organisation demonstrator system.

Together, these two demonstrators illustrate a range of application dimensions for the reuse of the CMS, in managing commitment knowledge in both autonomous, agent-mediated virtual organisations, and human-mediated decision-making. In both cases, commitments are expressed using CIF/SWRL against pre-existing service ontologies.

6 Discussion and Conclusion

In this paper we presented a set of components comprising a reusable commitment management service for agents operating in virtual organisations. The components build on the Semantic Web architecture, so allowing the management of commitments over Semantic Web services (and indeed any service defined using OWL or RDFS). Some of the components have more general applicability than commitment management: CIF/SWRL and the soft CSP ontology are reusable for any application of CSP and soft CSP-solving in a Semantic Web context (one such application is described in [12]). 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.

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[15], we will consider suitable extensions to the CSP ontology.

The SWRL FOL proposal to extend SWRL to full first-order logic[16] 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).

Currently, work on the e-response scenario is ongoing, and our focus is moving onto effective integration of human-mediated and agent-mediated decision processes.

Acknowledgments This work is supported under the Advanced Knowledge 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, 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

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. Stefano Bistarelli, Hélène Fargier, Ugo Montanari, Francesca Rossi, Thomas Schiex, and Gérard Verfaillie. Semiring-based CSPs and Valued CSPs: Basic properties and comparison. In Michael Jampel, Eugene Freuder, and Michael Maher, editors, *Over-Constrained Systems*, pages 111–150. Springer-Verlag LNCS 1106, August 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. Ken 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. I. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why Grid and agents need each other. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 8–15, 2004.

[15] http://www.w3.org/2005/rules/

[16] http://www.daml.org/2004/11/fol/

7. Eugene 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.
8. Hans W. Guesgen and Anne 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. 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/.
10. Ian Horrocks and Peter Patel-Schneider. A proposal for an OWL Rules Language. In *Thirteenth International World Wide Web Conference (WWW 2004)*. ACM, 2004.
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. 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.
13. 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.
14. J. Patel, L. Teacy, M. Luck, 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.

Appendix: CIF/SWRL RDF Syntax

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[17]; here we merely summarise the necessary extensions to the SWRL RDF syntax:

- We define a new `rdfs:Class Constraint`, with two associated properties: `hasQuantifiers` and `hasImplication`. The range of the former is an RDF list (of quantifier structures) and the range of the latter is a `ruleml:Imp`.
- We define the parent class `Quantifier` with sub-classes `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`.
- 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.

RDF/XML for the constraint `c2` is shown in Figure 7.

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

```
<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>
```

Fig. 7. RDF/XML for the constraint (commitment) $c2$