

Supporting Domain Experts In Formulating Constraint Satisfaction Problems

Derek Sleeman

University of Aberdeen
Kings College, Aberdeen
Scotland, AB24 3UE
d.sleeman@abdn.ac.uk

Stuart Chalmers

University of Aberdeen
Kings College, Aberdeen
Scotland, AB24 3UE
schalmer@csd.abdn.ac.uk

ABSTRACT

Constraint satisfaction is a powerful approach to solving a wide class of problems. However, as many non-experts have difficulties formulating tasks as Constraint Satisfaction Problems (CSPs), we have built a number of interfaces for particular kinds of CSPs, including crypt-arithmetic problems, map-colouring problems, and scheduling tasks, which ask highly focused questions of the user, c.f., the earlier MOLE, MORE, and SALT knowledge acquisition systems. Information from each of these interfaces is then transformed initially into a structured format which is semantic web compliant and is secondly transformed into the format required by the generic constraint satisfaction problem solver. When this problem solver is run, the user is either provided with solution(s) or feedback that the problem is underspecified (when many solutions are feasible) or over-specified (when no solution is possible). The system has 3 distinct phases, namely; information capture, transformation of the information to that used by a standard problem solver, and thirdly the solving and user feedback phase.

Categories and Subject Descriptors

H.5 [Information Interfaces and Presentation]: User centered design; D.3.2 [Language Constructs and Features]: Constraints

General Terms

Constraint Satisfaction Programming, User Modelling, Ontologies

Keywords

Constraint Satisfaction Programming, User Modelling, Ontologies

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. K-CAP 07, October 28–31, 2007, Whistler, British Columbia, Canada. Copyright 2007 ACM...\$5.00

1. INTRODUCTION

Constraint Satisfaction constitutes a powerful approach to problem solving, and over the last two decades a range of tools have been implemented [5, 9]. However, there are distinct skills involved in formulating tasks as CSP problems, the so called modelling problem. The format required by the solvers is deceptively simple: one has to specify a set of relevant variables, their domain values, and constraints between the variables. However the modelling process is still seen as challenging because the problem is often specified in a verbal form (e.g. as brain teasers), and it is the modeller's task to decide the relevant variables, the ranges associated with each variable, and most challengingly the constraints (relationships) which exist between the variables [10].

This paper addresses the modelling problem mentioned early by implementing a series of interfaces which ask for specific information about particular types of CSPs, e.g. map-colouring, cryptarithmic problems and scheduling. This information is then transformed into representational schema currently used on the web, namely an OWL¹ ontology (for the domain knowledge) and CIF/SWRL² rules for the constraints. As we shall see this is further transformed into the representation required by a generic constraint satisfaction solver.

One of the aims of this project is to produce a UI which corresponds to each of the types of CSPs (see section 4.2 for details). We then plan to produce a more generic UI which will handle a number of different CSP task types.

The overall architecture of the system implemented has 3 components. The role of the first is to CAPTURE information about a particular task, the second is to transform that information / knowledge into a form which a generic Constraint Problem Solver can use, and the third solves the task and reports the results to the user. We made a conscious decision to use OWL and SWRL as representational formalisms, as then it is, in principle, possible to augment the information created in the interfaces by other knowledge sources available on the web.

¹www.w3.org/TR/owl-features/

²www.csd.abdn.ac.uk/research/akt/cif/

The rest of the paper is organised as follows: Section 2 gives an introduction to CSPs; Section 3 gives a conceptual overview of the 3-staged system implemented; Section 4 discusses a classification of CSPs and describes Knowledge Acquisition (KA) interfaces built for some of these types/classes; Section 6 describes the implementation of a CSP solver; Section 7 outlines future work and describes some related work.

2. FORMULATING TASKS AS CSPS

CSPs have 3 aspects, namely (i) variables which are associated with (ii) domains (i.e. ranges of values) and (iii) the actual constraint expressions. Below we give examples of all 3 components:

$$a \mapsto D_1\{0..5\}, b \mapsto D_2\{0..5\}, c \mapsto D_3\{0..5\}$$

The above states that the variables a , b and c can be assigned any of the corresponding values given in the domains D_1, D_2, D_3 respectively. The following constraints, C_1 and C_2 , restrict the possible assignments that the variables can take simultaneously.

$$C_1: a-b > c, C_2: a+c < b$$

Given these constraints, the assignments of possible values are now restricted to:

$$a \mapsto D_1\{2..5\}, b \mapsto D_2\{1..4\}, c \mapsto D_3\{0..1\}$$

In general, a constraint satisfaction problem (CSP) is the process of satisfying a given set of statements by restricting the assignments of a given set of variables. More formally, we can define a set of variables $V_1..V_n$, each of which has a non-empty domain $D_1..D_n$ of values. A constraint is a restriction on the values that a subset of the variables can simultaneously take. For the purposes of definition, the set of constraints can be represented as $C_1..C_m$, where each constraint C_j is a pair $\langle J_j, P_p \rangle$, J_j being an ordered subset of $V_1..V_n$, and P_p being a subset of $D_1..D_n$. Therefore, a solution to a CSP is an assignment $V_n \mapsto D_n$ where every constraint $C_1..C_m$ is satisfied. For a more in-depth discussion of CSPs, see [10] or [11].

3. SYSTEM ARCHITECTURE

Previous projects in constraint modelling (see section 7) have focused on creating new meta-languages for specifying problems [5] or have required the user to model the CSP task using an existing representation formalism e.g. UML and OCL [9]. Here we are focusing on classifying the types of CSP problems which exist, with a view to creating an interface for each type to enable the non-CSP specialist to communicate the essence of their task. For some CSP tasks it is clear the nature of the interface required, e.g. Map-colouring problems or Cryptarithmic problems (of the form SEND +

MORE = MONEY) whereas the differences between Scheduling / Configuration / Assignment / Constraints / Positioning are in general much more subtle, and will need some further analysis before helpful distinctions can be made. (In fact, as noted in section 4.2, the Cryptarithmic and Map-colouring tasks are both members of the Assignment class).

The general inspiration for this approach is the Knowledge Acquisition (KA) work done in the early 80's when several groups realised that KA could be made more focused if one acquires knowledge for a particular purpose. The MOLE [3] and MORE [6] systems, for instance, acquired knowledge which would support only classification/diagnosis and thus only needed to capture the several diagnostic classes and the corresponding diagnostic rules. Similarly, the SALT KA system [7] was designed to acquire knowledge to support the propose-and-revise algorithm and so elicited 3 types of knowledge/information from the domain expert namely:

- Procedural knowledge to specify how an existing entity (such as a motor) could be enhanced/changed
- Constraints to specify relationships which must or must not hold between variables
- Fixes: what to do if a particular constraint is violated.

So the argument then is that a KA interface will be implemented for each of the classes of CSPs which will ask relevant focused questions. (We will address later the taxing question of how a non-CSP expert can choose the relevant KA interface for a particular problem they wish to solve.) In fact as figure 1 shows, we have conceptualised the task of formulating and solving CSP as three phases:

- CAPTURING the essence of the task (outlined above & discussed in section 4)
- TRANSFORMING the task from the information collected from each interface to a common formalism. In fact, we have chosen SWRL and OWL, emerging WWW standards, so that the information acquired can, in principle, be enhanced by other Web based Knowledge Sources.
- SOLVING & Providing User Feedback. The final phase attempts to solve the task and either provides the user with a result or information which indicates that the task is over (or under) specified.

4. CAPTURING USER INFORMATION

4.1 Model Generation vs. Model Selection

Systems such as a CONJURE [5] provide the non-CSP expert with a high-level language in which to formulate the task s/he wishes to solve. This approach has the advantage of allowing a wide range of problems to be specified (in principle) but the disadvantage that it gives the user little guidance. We refer to this as the Model Generation approach.

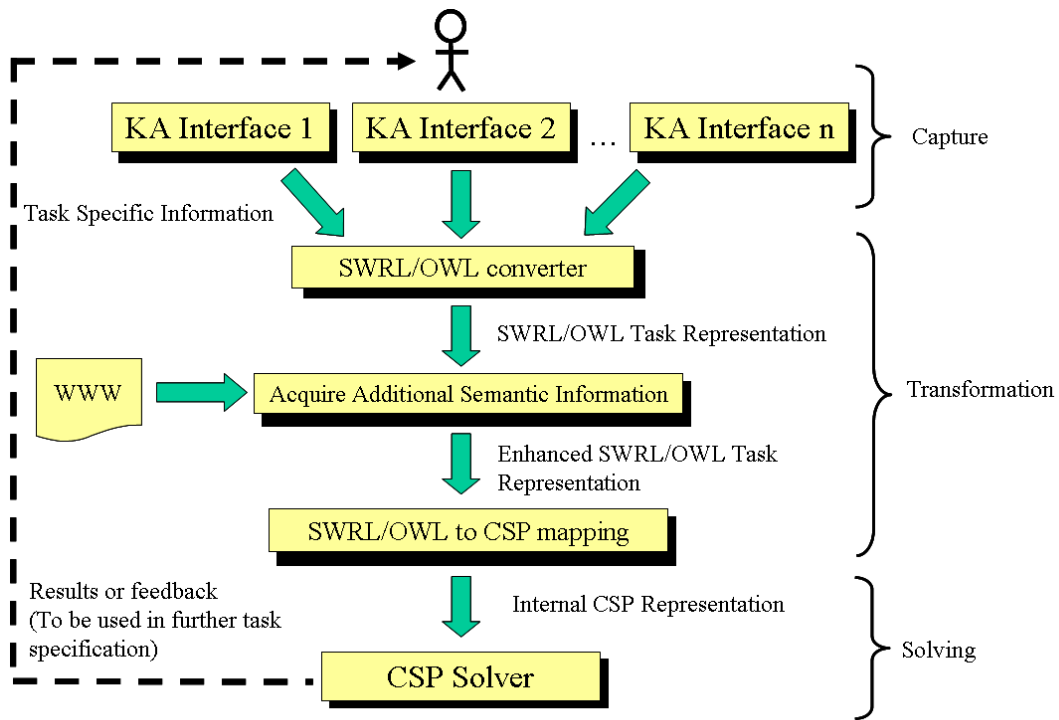


Figure 1: Conceptual System Design showing 3 principal phases

By contrast we refer to the approach being followed here as the model selection approach. Conceptually, at least we can think of this approach as providing the user with a series of templates (one corresponding to each of the CSP classes or perhaps subclasses). Each template has a number of slots which the user has to complete. If a user is able to provide a consistent set of responses to all the slots associated with a template, then we shall say that this task corresponds to that CSP class (e.g. Assignment). In practice we believe that this part of the system will be implemented as a series of “linked templates”, so that at any stage of the specification of a task, it will be possible to specify:

- Which classes have been ruled out (because certain information is *not* available)
- Which classes are currently completely satisfied (could be null)
- Which information is needed, to satisfy the remaining possible classes

4.2 Classes of CSP tasks

As mentioned in section 3, we are interested in describing the number of CSPs in sufficient detail, so that we can identify distinctive classes and sub-classes. Miguel et. al³, in their work with ESSENCE and CONJURE, have introduced a classification and suggest the following definitions:

³<http://www.cs.york.ac.uk/aig/constraints/AutoModel/Essence/Tree/>

- Scheduling - characterised by assigning start times to a series of tasks that have to be performed by some deadline with the possibility of precedence constraints between them (e.g. process A must be completed before process B can start).
- Configuration - where the problem involves assigning a unique value to a variable according to constraints between the values and their variables.
- Assignment - similar to configuration problems, but the assignment of values to variables is not a one-to-one relationship. Subsets of this class include permutation and partitioning problems.
- Construction - the object is to construct a set of variables according to a goal (such as maximising the values assigned to the set of variables). Constraints here can be on the membership of this set, and on the position of the variable in that set.
- Positioning - involve arranging objects according to spatial/geometric constraints. Typically all objects must lie within a boundary and objects are not permitted to overlap (where these restrictions are specified as a series of constraints).

Generally, we can classify each problem type by the relationship between the Variables (described as Objects by Miguel) and Assigned Domain Values (defined as Labels). For instance, in permutation problems each label is used only once



Figure 2: The Map-colouring KA Interface

(effectively saying that the values assigned to a set of variables must all be different), whereas with partitioning problems the actual assignment of the label value is unimportant, but what is important are the groups of variables which have the same values.

To this end, we have attempted to create an ontology to model $\text{Variable} \mapsto \text{Domain}$ pairs and describe the relationships which hold between them for the various classes (described in section 5.2). We then attempt (through the various interfaces) to instantiate this ontology with information relating to a particular task.

4.3 Implemented KA interfaces

In previous sections we discussed in some detail a classification for CSPs. Here we outline two Interfaces we have recently developed, namely:

- A Mapping task (a subset of the assignment class)
- A Cryptarithmic Problem (a particular kind of assignment task)

A screen shot for the Mapping interface is given in Figure 2. The map colouring task specifies a number of physical areas (e.g. countries, counties, areas of a town) and specifies that adjacent areas should have **different** colours. So the task involves specifying the set of colours available, and the adjacency relationships between the several areas. A solution to the problem is one where all the adjacent areas have different colours. So for example if the objects to be assigned a colour are the 4 counties of the SouthWest of England, namely: Cornwall, Devon, Somerset & Dorset, and the relationships between the 4 objects are as given in the left hand diagram in figure 3, and the 3 colours to be assigned are white, grey & black, then the right hand figure would be an acceptable solution. The UI in figure 2 used to collect information about the task, initially asks the user to provide the names of the 4 objects (top left hand corner), then the user is asked to indicate the spatial relationships between the objects (this is done in the window called Positions), at which point the system allows the user to revise the relationships

provided until they are happy with the resulting model. Finally, the system asks the user for the available colours, and then the complete task is passed to the problem solver.

Cryptarithmic problems are of the form:

$$\begin{array}{r} \text{CROSS} + \\ \text{ROADS} \\ \hline \text{DANGER} \end{array}$$

Where one is told that each of the letters is in the range $0..9$, each letter has a distinct value, and $D \neq 0$. So it is usual to formulate the problem as a series of variables (in the case of this task: $C, R, O, S, A, D, N, G \& E$). Note in this case the 2 entities are to be **added** to give the result **DANGER**. In formulating the main constraint for this problem we need to remember the significance/semantics of a column in an arithmetic task. The right hand column consists of units, the adjacent column represents units of 10s, and the next column represents 100s, etc. Bearing this in mind the main constraint can be expressed as:

$$10000 * C + 10000 * R + 1000 * R + 1000 * O + 100 * O + 100 * A + 10 * S + 10 * D + S + S = 100000 * D + 10000 * A + 1000 * N + 100 * G + 10 * E + R$$

or as:

$$10000 * (C + R) + 1000 * (R + O) + 100 * (O + A) + 10 * (S + D) + 1 * (S + S) = 100000 * D + 10000 * A + 1000 * N + 100 * G + 10 * E + R$$

Using some simple knowledge of arithmetic enables one to formulate several additional constraints which will help further reduce the size of the resulting search space. Because the result (**DANGER**) has a figure in the 100000 column this implies that both $C \& R$ in the 10000 column can not be 0. This information is formulated as two further constraints:

$$\begin{array}{l} C > 0 \\ R > 0 \end{array}$$

If we have a powerful constraint specification language, we could formulate a further constraint. Namely that R must be even. This can be deduced as the first column (into which no carries are possible) has 2 S 's, and so R must be even including 0 (as the S 's could both be 5 this would mean that R would then be 0 and a carry would be propagated into the next column).

Section 7 details planned user studies and how we expect to develop more generalised interfaces for user advice on additional information they will need to add to make tasks fit particular classifications.

5. TRANSFORMATION: OWL & CIF/SWRL

For our implementation, we have followed the 3 phases of the design (Capturing, Transforming, Solving) described in

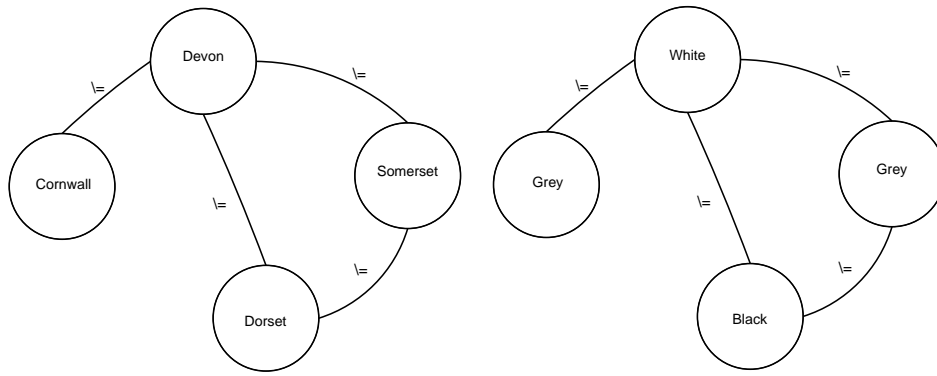


Figure 3: An example Map Colouring Specification and solution

section 3. To this end, our architecture is split into 3 distinct sections (driven also by the design decision to deploy the solver as an openly available web service).

While we have discussed the ideas behind the KA Interfaces in section 4, here we discuss the representation of the constraints using CIF/SWRL [8], a SWRL extension, plus the design of the ontology for variable and domain elements.

5.1 CIF/SWRL Constraint Representation

SWRL (Semantic Web Rule Language) is currently the leading proposal for representing the Semantic Web Logic layer. While essentially a rule language, quantified constraints are also easily representable using such constructs. CIF/SWRL (Constraint Interchange Format) models this using the implication structure from SWRL, with the addition of nested quantifiers.

As an example, we will represent the constraints $a-b > c$ and $a * c < b$ from section 2 in the SWRL syntax:

```

implies (
  quantifiers (
    forall (i-variable (avar) variable)
      forall (i-variable (bvar) variable)
        forall (i-variable (cvar) variable)
  )
  Antecedent (true)
  Consequent (
    hasName (avar, A)
    hasName (bvar, B)
    hasName (cvar, C)
    hasType (avar, integer)
    hasType (bvar, integer)
    hasType (cvar, integer)
    avar - bvar > cvar
    avar * cvar < bvar
  )
)

```

As our representation is in First Order Logic, we need to first define quantifiers on the variables. As these variables do not specify quantifiers, the default universal quantifier is applied; in CIF/SWRL this is expressed as the `forall` construct. The format of a statement in CIF/SWRL is of the

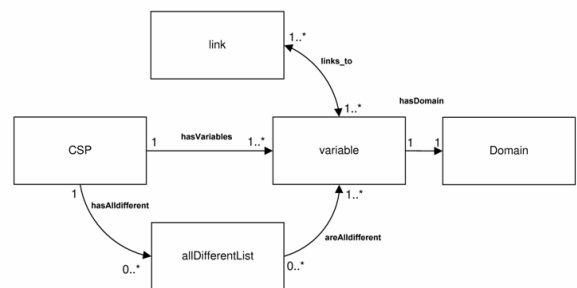


Figure 4: The Variable \mapsto Domain Ontology

form $\text{Antecedent} \mapsto \text{Consequent}$. For all constraints, we define the antecedent to be (trivially) true, and the several constraints are thus expressed in the consequent of the rule. We also require that the identity of the variables and their type information are also declared, so that this information can be linked to corresponding entities in the ontology (section 5.2). To express the actual constraints, we use SWRL's inbuilt mathematical and comparison operators.

5.2 Domain and Variable Information in OWL

Variable and domain information is represented using OWL. We have modelled this as a collection of $\text{Domain} \mapsto \text{Variable}$ relationships, as most of the information on the actual constraints is contained in the CIF/SWRL rules.

Figure 4 shows our ontology design. We have a CSP object, containing one or more variable objects that can be of type `integer` or `string`. Each variable contains a `hasDomain` relationship to a domain object, which contains domain information, showing the range of the variable as continuous, or as individual elements. In the CSP object, we have defined some extra information pertinent to the specified CSP problem (such as the `hasAllDifferent` property and a `solutionType` property (which define the

```
( $\forall l, p, q$ ) link( $l$ )  $\wedge$  links_to( $l, p$ )
 $\wedge$  has_name( $p, \text{'owner'}$ )  $\wedge$ 
satisfiedStringValue( $p, \text{'Brit'}$ )  $\wedge$ 
links_to( $l, q$ )  $\wedge$  has_name( $q, \text{'house'}$ )  $\mapsto$ 
satisfiedStringValue( $q, \text{'red'}$ )
```

Figure 6: A first Order Logic representation of the “The Brit lives in a red house” Einstein constraint

solution strategy for solving the CSP)⁴. Figure 5 shows the domains and variables for the example in section 2 defined using this ontology.

To let us to express more complex relationships between variables, each variable has a reference to one or more `link` entities. These entities allow us to define more complex relational constraints between the variables (basically allowing us to express constraints not just on the entities we are modelling, but on relationships between entities). As an example, a statement from the Einstein⁵ problem expressed in first order logic is given in figure 6. This shows how we use the `links_to` relationship to express a relationship between two variables. The `links_to(l, p)` and `links_to(l, q)` statements to show that a relationship `link(l)` exists between a person `p` and a house `q`. Using this link entity, we can then express constraints such as stating that a specific type of person lives in a specific type of house (in this case that the Brit lives in the red house).

6. SOLVING: CSP IMPLEMENTATION

A major design decision behind the implementation of this solver was to make the solving process freely available as a re-usable component. To this end, we have developed the CSP as a web service with an open queryable interface. We use AXIS⁶ as the platform, with the interface defined using a combination of OWL–lite to represent the variables and their domains and CIF/SWRL to represent the actual constraints⁷. We also implement a number of parsing algorithms, to support the following information flow:

KA Interface \mapsto CIF/SWRL + OWL \mapsto CHOCO Solver
data structures

For the principal classes of CSPs covered here, we have written a generalised finite domain constraint solver in CHOCO⁸. To formulate a problem in CHOCO we create a finite domain CSP instance. We then use mapping rules for each CIF/SWRL construct to transform the CIF/SWRL representation into CSP constraints.

⁴For a full definition of this ontology see <http://www.csd.abdn.ac.uk/~schalmer/akt>

⁵<http://www.davar.net/MATH/PROBLEMS/EINSTEIN.HTM>

⁶<http://ws.apache.org/axis/>

⁷For a detailed description of the design and capabilities of the semantic interface implementation, see Aberdeen University Computing Science Technical Report AUCS/TR0604

⁸<http://choco.sourceforge.net/>

In particular, we wish (in the future), to utilise the explanation-based constraint solving (e-CP) mechanism in this solver for modelling our “repair and remodel” concept (see Future Work section 7).

For scheduling we do not use the generic CSP solver, as we have found the scheduling problems to be distinct from the others classes (because of the temporal relations involved). The scheduling algorithm is constructed using the Java Constraint Library (JCL)⁹, and had been used in the CONOISE¹⁰ project for managing agent commitments [2]. It is based on a cumulative scheduling algorithm, although it uses the soft constraints extension from the JCL to manage and assign priorities to the scheduled tasks, allowing one to find subsets of satisfiable tasks on over-constrained problems.

7. DISCUSSION AND FUTURE WORK

A number of projects have looked at the problem of CSP modelling. An approach taken by [5] is to provide a high-level language, ESSENCE, for specifying CSP problems. This language is then translated into a CSP by a system, CONJURE, that refines the specification. Renker [9] uses UML and OCL to provide a modelling framework for constraints. Alternative approaches include Fish et. al [1] who look at diagrammatic modelling of constraints using “spider diagrams”, a formalisation based on Venn diagrams.

We plan to evaluate the web-based KA interfaces for the cryptarithmic and mapping problems. We will compare the numbers of tasks formulated and the average time taken by two groups namely one doing the task using pen-and-paper and the other using the CSP interfaces. We will also administer a questionnaire about how the interfaces could be enhanced. A further aim of our future work is to implement a KA interface for each of the CSP types identified in section 4.2, and then to link them so as to reduce the number of questions that a user needs to answer. Earlier, in section 3, we said we would have to address the taxing problem of how a user decides which type of problems they want to solve and hence which interface to use. The interface design outlined in section 4.1, where the user is presented with a series of linked “templates”, finesses that issue to some extent. However, we will wish, both by observation of users and questionnaires, to assess how successful this design has been. Further, although we have described systematically the various CSP tasks, we expect, as we continue to use the interfaces with actual tasks, that these descriptions will become refined.

Another major issue we wish to explore is that of providing user feedback on problems to help with remodelling and repair. When a user specifies a set of constraints on a problem, there may be no solution returned for that set as the problem is over-constrained. In such circumstances, constraint relaxation [4] techniques aim to partially solve a given problem

⁹<http://liawwww.epfl.ch/JCL/>

¹⁰<http://www.conoise.org>

```

<?xml version="1.0"?>
<rdf:RDF
  xmlns:cspTemplate="http://www.csd.abdn.ac.uk/~schalmer/akt/cspTemplate.owl#"
  xmlns="http://www.csd.abdn.ac.uk/~schalmer/akt/exampleInstance.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.csd.abdn.ac.uk/~schalmer/akt/exampleInstance.rdf">

  <cspTemplate:csp rdf:ID="cspInstanceExample">
    <cspTemplate:hasVariable rdf:resource="#variable1"/>
    <cspTemplate:hasVariable rdf:resource="#variable2"/>
    <cspTemplate:hasVariable rdf:resource="#variable3"/>
    <cspTemplate:hasCSPType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">assignment</cspTemplate:hasCSPType>
    <cspTemplate:hasConstraints rdf:datatype="http://www.w3.org/2001/XMLSchema#string"><p style="margin-top: 0">
      www.csd.abdn.ac.uk/~schalmer/akt/constraintFile.owl<p></cspTemplate:hasConstraints>
    <cspTemplate:allDifferent rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</cspTemplate:allDifferent>
    <cspTemplate:solutionType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">minimize</cspTemplate:solutionType>
  </cspTemplate:csp>

  <cspTemplate:variable rdf:ID="variable1">
    <cspTemplate:rangeStart rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</cspTemplate:rangeStart>
    <cspTemplate:rangeStop rdf:datatype="http://www.w3.org/2001/XMLSchema#int">9</cspTemplate:rangeStop>
    <cspTemplate:hasType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">integer</cspTemplate:hasType>
    <cspTemplate:hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">A</cspTemplate:hasName>
  </cspTemplate:variable>

  <cspTemplate:variable rdf:ID="variable2">
    <cspTemplate:rangeStart rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</cspTemplate:rangeStart>
    <cspTemplate:rangeStop rdf:datatype="http://www.w3.org/2001/XMLSchema#int">9</cspTemplate:rangeStop>
    <cspTemplate:hasType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">integer</cspTemplate:hasType>
    <cspTemplate:hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">B</cspTemplate:hasName>
  </cspTemplate:variable>

  <cspTemplate:variable rdf:ID="variable3">
    <cspTemplate:rangeStart rdf:datatype="http://www.w3.org/2001/XMLSchema#int">0</cspTemplate:rangeStart>
    <cspTemplate:rangeStop rdf:datatype="http://www.w3.org/2001/XMLSchema#int">9</cspTemplate:rangeStop>
    <cspTemplate:hasType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">integer</cspTemplate:hasType>
    <cspTemplate:hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">C</cspTemplate:hasName>
  </cspTemplate:variable>

</rdf:RDF>

```

Figure 5: An Example Domain Representation For The Task Described in Section 2

by maximizing the number of constraints applied. Alternatively, we may find a problem under-defined (i.e. there are a vast number of possible valid solutions) where we may wish to elicit more information from the user to constrain the problem further.

Our CSP implementation is based on a simple generalised finite domain solver, so we are not able to solve the range of problems that a language such as ESSENCE can support. While it is not our intention to do this, we do wish to classify the subset of problems which can be solved as well as explore the representation of CSPs using existing semantic web technology. As discussed earlier in the paper, there were several motivations for using a semantic-web compliant representation for the CSP. The principal issues still to be investigated, is whether it is possible to exploit the fact that the intermediary representation of the CSPs is OWL and CIF/SWRL, by enhancing the initial user-provided information about a task with relevant information available from the semantic web [12].

In this paper, we have described a set of user interfaces to aid in the formulation of tasks as CSPs. We have described our representation of the CSPs using semantic web technology, namely OWL for variable and domain representation and CIF/SWRL for representing constraints. We have also described the flow of information, from the KA interfaces, to a semantic web representation and finally to the native constraint solving language.

Acknowledgments

This work is supported by the Advanced Knowledge Technologies (AKT)¹¹ Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council (EPSRC). The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton, and the Open University. The authors are also grateful to Lin Lin and Xuezhou Yuan, who provided the implementation of the web service gateway used the CIF/SWRL transformation tools. The constraint fusion ideas and portions of the solving mechanism were developed in the context of the KRAFT¹² and CONOISE¹³ projects, funded by the EPSRC and British Telecom.¹⁴

8. REFERENCES

- [1] Fish A. and Flower J. Investigating reasoning with constraint diagrams. In *VLFM04, Visual Languages and Formal Methods*, volume 127, pages 53–69. Elsevier, Rome, April 2005.
- [2] S. Chalmers, A. Preece, T. J. Norman, and P.M.D. Gray. Commitment management through constraint reification. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 430–437, Columbia University, New York, July 2004.

¹¹<http://www.aktors.org>

¹²<http://www.csd.abdn.ac.uk/~apreece/Research/KRAFT.html>

¹³<http://www.conoise.org>

¹⁴grant number GR/N15764/01

- [3] L Eshelman. Mole: a knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus (Ed.), editor, *Automating Knowledge Acquisition for Expert Systems*, pages 37–80. Kluwer Academic, Norwood, Mass, 1988.
- [4] 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.
- [5] A.M. Frisch, M. Grum, C. Jefferson, B. Martinez-Hernandez, and I. Miguel. The essence of essence: A constraint language for specifying combinatorial problems. In *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 73–88, 2005.
- [6] G. Kahn, S. Nowlan, and J. McDermott. More: an intelligent knowledge acquisition tool. In *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, pages 581–584. 1985.
- [7] Sandra Marcus and John McDermott. SALT: a knowledge acquisition language for propose-and-revise systems. *Artif. Intell.*, 39(1):1–37, 1989.
- [8] Craig McKenzie, Alun Preece, and Peter Gray. Extending SWRL to Express Fully-Quantified Constraints. In Grigoris Antoniou and Harold Boley, editors, *Rules and Rule Markup Languages for the Semantic Web (RuleML 2004)*, LNCS 3323, pages 139–154. Springer, Hiroshima, Japan, 2004.
- [9] Gerrit Renker. A modeling framework for constraints. In *8th International Conference on Constraint Programming*, pages 8–13, September 2002.
- [10] Barbara Smith. A tutorial on constraint programming. Technical Report 95.14, School of Computing Research Report, University of Leeds, April 1995.
- [11] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.
- [12] Yi Zhang, Wamberto Vasconcelos, and Derek Sleeman. Ontosearch: An ontology search engine. In *The Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge, 2004.