Commitment Management Through Constraint Reification

Stuart Chalmers Alun Preece Timothy J. Norman Peter M.D. Gray
Department of Computing Science, University of Aberdeen,
Kings College, Aberdeen, AB24 3UE, Scotland, UK.
{schalmer,apreece,tnorman,pgray}@csd.abdn.ac.uk

Abstract

In a virtual organisation (VO), a group of cooperating agents may offer resources (or services) consisting of all, or part, of the sum of their individual contributions. Some of these resources may already be in use for certain time periods (existing commitments), and these existing commitments may vary in value to the VO. Using constraint reification and cumulative scheduling methods, we investigate ways in which agents may manage their resources and existing commitments when faced with a decision to take on new commitments. We describe a technique that allows an agent to intelligently construct satisfiable permutations consisting of existing and new commitments, and use preference and quality information to choose between these permutations. We show how constraint reification is used to model whether commitments are breakable/re-negotiable and how this influences the permutations available to the agents.

1. Introduction and Motivation

A virtual organisation (VO) is a grouping of semi-independent autonomous agents (representing different individuals or organisations) each of which has a range of service-providing capabilities and resources at their disposal. These agents co-exist and often compete with one another in a virtual marketplace environment. Each agent attempts to sell its services in a way that maximises their individual gain, by advertising the services to potential customers in terms of the cost and qualities of the services. There are occasions where one or more of the agents may realise there are potential benefits to be obtained from pooling resources, either with a partner offering complementary expertise (to provide a new type of service package), or with a direct competitor (by forming a coalition). When this opportunity for pooling is identified, the potential partners go through a process of trying to form a new VO to exploit the perceived niche.

Consider two examples:

- A high bandwidth mobile service provider may agree to collaborate with a streamed video content provider in the delivery of such content as a service to mobile devices. This corresponds to a new type of packaged service.
- A pair of relatively small airline companies with complementary routes agree to form a coalition and coordinate their services so that they may offer flights between a wider range of destinations, with a view to becoming more competitive in this market.

The problem of VO formation is addressed in detail in [10]. If the agents succeed in forming a VO, the collection of partners will then act as a single conceptual unit in the context of providing the new service (though they will likely retain their individual identity outside this context). In general, each participating agent will commit some part of its available service-providing capacity to the VO; it will retain the remaining portion either to offer other services outwith the VO (perhaps as a partner in some other VO), or to expand the collective services provided by the VO. Every entity participating in a VO therefore needs a reasonably sophisticated decision-making capability to manage its existing commitments, and decide when to take on new commitments. Moreover, the VO itself needs this capability as a whole.

When an agent (or coalition of agents) receives a request to agree to the provision of the resources under its management, it may or may not be able to fulfill this request alongside existing commitments. In essence, if it were to asceed to the new request, the problem of allocating its resources under commitments it has made will become over-constrained. The agent (or group) then has a choice to commit to the new request — and break one or more existing commitments — or to reject the request. This choice may depend on influences such as the value of existing commitments, who the commitment is for, and whether it is renegotiable.

AAMAS'04, July 19-23, 2004, New York, New York, USA.

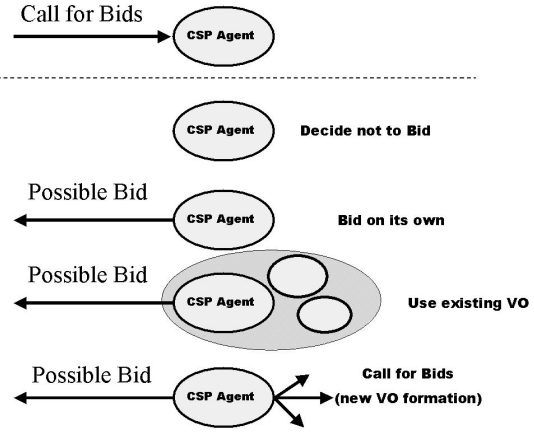

Figure 1. The agent decision making process.

The various decision-making possibilities can be enumerated from the point-of-view of a single service provider agent. The purpose of such an agent is to be able to create a bid in reply to a call for services, and decide how much resource it can, and more importantly, how much resource it *wants* to provide as a bid for the procurement of that service. Furthermore, any agent may, when considering what to offer, opt to attempt to form a VO by recruiting partners if it identifies a shortfall in its existing resources available. Each agent must, therefore, be able to act as a contractor and supplier in any given situation.

Figure 1 shows this overall scenario, where the agent acts as the supplier and receives a call for bids. The following responses are possible: (i) the agent can decide *not* to bid for the service; (ii) it can bid using only its own resources; (iii) it can provide a bid from within an existing VO collaboration utilising the resources of the combined VO; or (iv) it can identify a need for extra resources not available within the existing VO. This last option represents the scenario where the agent becomes the contractor, and itself begins the process of recruiting partner agents in the environment.

The remainder of this paper focusses on this decision-making process, introducing a technique based on constraint reification and cumulative scheduling. Managing agent commitments entails distinguishing between the importance of commitments, being able to identify conflicting commitments, and being able to provide solutions to these conflicts. Using finite domain constraint solving highlights conflicting commitments and the use of constraint reification allows an agent (or group of agents) to attach information to such commitments on whether they are breakable. The agent(s) can then identify important constraints and begin to differentiate between commitments and how valuable they are to the agent(s). This technique allows an agent to reason explicitly over a space of satisfiable and unsatisfiable combinations of commitments, and hence to decide rationally when to take on new commitments, when to break or renegotiate existing commitments, and how to choose between alternative combinations of commitments.

2. Modelling Resources using Cumulative scheduling

An agent's resources can be defined by two values, the length of time that the agent can provide the resource for, and the amount of resource available to the agent. Furthermore,, to model the use of an agent's resources, we need to record when resources are scheduled to be utilised under an agreement and in what quantity — these are what we refer to as commitments.

To do this we model the resources of an agent and its commitments as a cumulative scheduling constraint satisfaction program (CSP) [2] using finite domains to represent the time and amount of the resources available and constraints over those domains to represent commitments to provide some or all of the resources for a particular time period.

Cumulative scheduling is defined as the maximum allowable limit of a finite resource that *can* be used by an agent or agents at any given time [1]. So there is a communal resouce of which various agents can obtain an amount (i.e. the resource is commited to a certain agent). To allow multiple contributions of the same resources *from* different agents, instead we have the agents contributing to the communal resource and we define the required value (new commitment) as a *minimum* allowable limit on this communal resource so that the set of agents *must* provide this service *at least or above* the required threshold limit over the required time to satisfy the new constraint. This means that as well as showing what commitments there are on the resources provided by the agents we can also model what resource are contributed by each agent towards the communal 'pool' that will be required to satisfy the new commitments.

To explain our cumulative scheduling based algorithm more formally, we first define the problem. Given a set of n agents, each of whom can provide a specific finite amount of a resource R, $\{R_1 \dots R_n\}$, a set of start times describing when the agent can begin providing each of the resources $\{S_1 \dots S_n\}$ and a set of durations over which the resource is available $\{D_1 \dots D_n\}$ we can say, for an agent i, that the function $\delta_i(t)$ is true if the current time t is within the agent's resouce start and end time.

$$\delta_n(t) = \begin{cases} 0 & if\ \ (t < S_n) \vee (t > (S_n + D_n)) \\ 1 & otherwise \end{cases}$$

AAMAS'04, July 19-23, 2004, New York, New York, USA.

Then, an amount r of resource R is available over a time period $0 \ldots v$ iff:

$$\forall t \in \{0 \ldots v\} \quad \sum_{a=1}^{n} R_a \delta_a(t) \geq r$$

In other words, the total sum of the resource provided by the set a of agents $\{1 \ldots n\}$ at any time between $0 \ldots t$ does not fall below the resource limit r specified. Using this representation means that we can also use constraints on the agent resource domains to represent existing commitments on those resources (and thus taking away resources from the communal 'pool'). In our scenario, this helps us to model the decision making process because the agents can look at their own resources and their own existing commitments, and see whether they can accomodate the new allocation of resources asked of them.

3. Motivating example

To illustrate the use of cumulative scheduling for resource modelling and management, we now present a detailed example. This example is a simplification of the type of problem that occur in the CONOISE application domains alluded to in the introduction (see also [10]).

Consider two 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 (shown in the first schedule in figure 2):[1]

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

If a new request, `N` is received by the agents to provide 15x from 0→10, then the agent has 4 main choices[2]:

- Reject `N` and satisfy existing commitments `c1`, `c2` & `c3` (Schedule 1 in Fig.2)
- Accept `N` and break `c1` & `c2` (Schedule 2)
- Accept `N` and break `c3` (Schedule 3)
- Accept `N` and break `c1` & `c3` (Schedule 4)

As the number of agents and commitments increases the number of possible combinations of solutions that satisfy all the commitments (and solutions that break commitments) grows exponentially. Also the number of trivial solutions (i.e. solutions that vary in extremely small detail) increases (e.g. schedule 3 could take `7x` from `a1` and `8x` from `a2` rather than `5x` and `10x`which would not affect the commitments broken). The main emphasis behind the CSP/cumulative scheduling 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 extend the cumulative scheduling CSP 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.

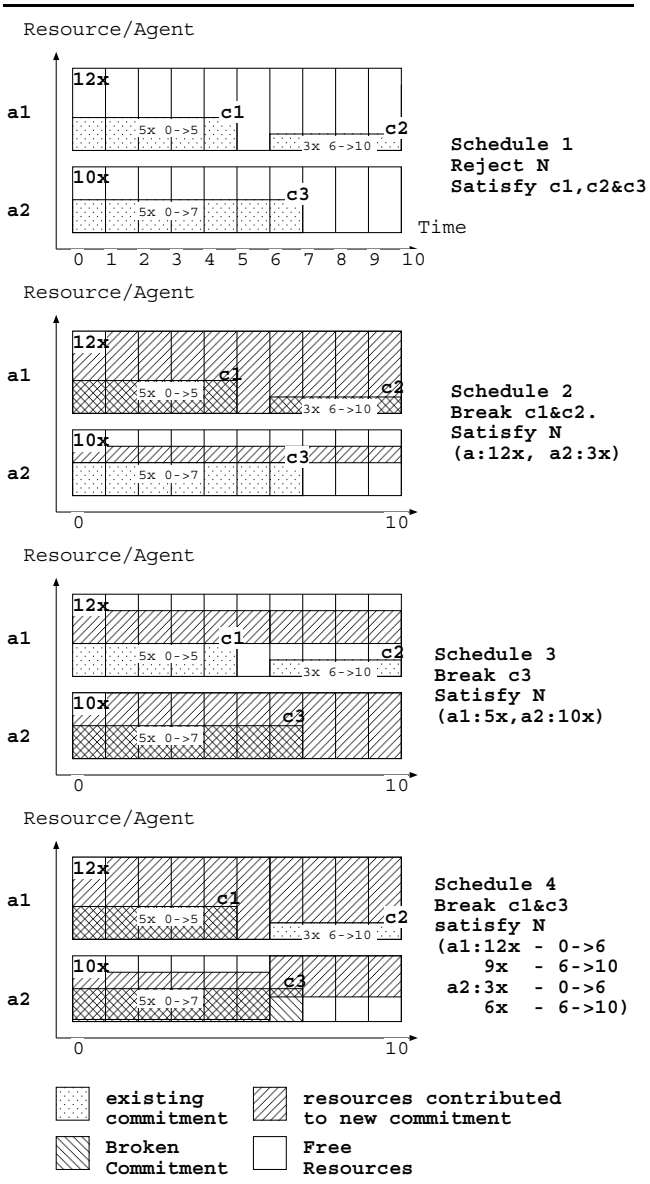

Figure 2. Agent `a1` & `a2`'s Options for providing new commitment N

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

[2] Although there are many value permutations, in terms of commitments broken these are the main choices.

AAMAS'04, July 19-23, 2004, New York, New York, USA.

We implement this as a reification extension to the current cumulative scheduling CSP that uses a combination of reification and constraint value labeling to provide the required commitment management and prioritisation.

4. What is Constraint Reification?

Constraint Logic Programming attempts to find a satisfiable subset of values from an initial set given a number of limitations (constraints) imposed. Constraint Reification associates a value with each constraint, which can be evaluated to true or false depending on the satisfiability of the constraint. These reified values can also be reasoned about, thus we have a set of meta-level boolean values showing the constraints that have been succesfully applied (and the ones that have failed to be applied). The reification process is bidirectional, so we can also specify that a constraint must be satisfied (i.e. must have a true value for its reification), even if this means that a suboptimal set of the constraints may be chosen to be satisfied for the problem, or indeed the problem itself is not satisfied.

We thus promote the constraints from being value limiters on knowledge to pieces of knowledge themselves. The application of the constraints then become just as important as satisfying the problem itself.

In the following example we have a variable D with a finite domain of possible values:

$$D \in \{1,2,3,4,5,6,7,8,9\}$$

and two constraints on those domains:

$$D \leq 5,\ D > 5$$

If we reify these constraints thus:[3]

$$D \leq 5 \iff A,\ D > 5 \iff B$$

we assign a true or false value to each constraint (A and B respectively).

$$A \in \{0,1\},\ B \in \{0,1\}$$

So when we try and find a solution to this problem, we end up with one of the following:

$$D \in \{1,2,3,4,5\}$$
$$A=1,\ B=0$$

$$D \in \{6,7,8,9\}$$
$$A=0,\ B=1$$

Both sets of values for D include breaking at least one of the constraints. We can specify that a constraint *cannot* be broken by specifying a '1' as the reified value. So stating B=1 means that any solution provided for D *must* now hold true to the constraint X>5 (and we thus lose the first solution above). In this way we can begin to differentiate between the importance of constraints in problems by making them breakable or non-breakable.

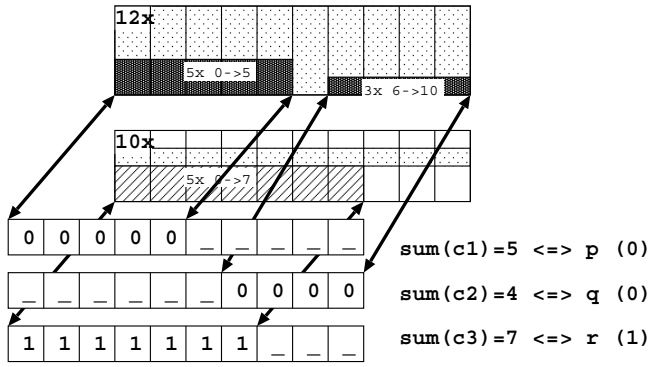

Figure 3. A Reification Example

We utilise this reification process by attaching a reified value to each of the existing commitments and the new commitment. We can then create solutions where certain commitments are 'allowed' to fail (and produce a '0' reified value). We can also specify non-breakable constraints by specifying their reified value as '1'.

5. Management Strategies

5.1. Reifying Commitments

Figure 3 shows an example of the reification strategy working on schedule 2 from Figure 2. We first have a reified value on each of the time slices for c1,c2&c3. This corresponds to whether the existing commitments have been satisfied at that particular time when the new constraint N (15x:0→10) is applied[4]. To have a singular value to sum up these slices and therefore show whether the whole constraint has been satisfied or not, we also have a reified value on these binary values (shown as p,q&r in the example) specifying that their sum must be equal to the length of time given for the duration of the commitment (i.e. the constraint has held over the given time).

We similarly construct a set of reified values for checking the validity of commitment N. Given a set of resources R, $\{R_1 \ldots R_x\}$ and a set of applied existing commitments C, $\{C_1 \ldots C_y\}$, we can define a reified value V for each time slice of a new commitment to provide value N over time $0 \ldots v$ as:

3 We use the notation <=> to indicate that a constraint has been reified.

4 The underscores represent the times slices where the commitment is not valid.

AAMAS'04, July 19-23, 2004, New York, New York, USA.

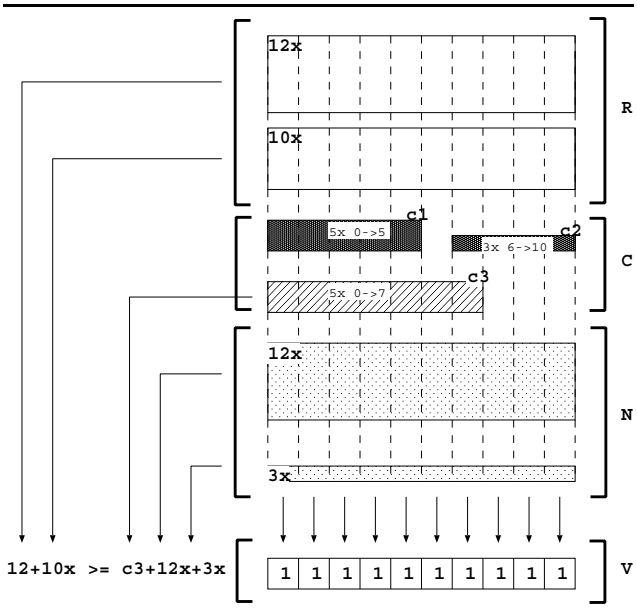

Figure 4. Reification of new commitment

$$\forall t \in \{0 \ldots v\} \quad \left(\sum_{i=1}^{x} R_i\right) \geq \left(\left(\sum_{j=1}^{y} C_j\right) + N\right) <=> V$$

Using these reified values, we can check whether the new constraint has been satisfied over its required duration. This can be seen graphically in Figure 4 (Note that in this case `c1` and `c2` are not applied as part of the existing set of commitments `C`).

5.2. Value Labeling and Limiting Strategies

Even for the simple example in Figure 2 we can see that there are many possible solutions and permutations of solutions. We can have numerous combinations of values from `a1` and `a2` contributing to the `15x` needed for commitment `N`, plus we add extra complexity by allowing constraints `c1,c2&c3` to be broken.

The most important solutions though, are ones that show the breaking of commitments, and what combination or subset of commitments need to be broken in order to satisfy new commitments. To be able to do this, as well as prioritise commitments, we look at the way in which the values for the reified constraints, as well as the values for the resources themselves are found.

To prioritise and label values we follow three general rules:

- Designate non-breakable commitments
- Order resources/commitments is terms of priority
- Choose value labeling strategy

To start with we can designate certain commitments as non-breakable by giving their reified value as '1', rather than leaving them uninstantiated. For instance, in the example in Figure 3 we could specify the reified value `p` to be 1, therefore limiting the solutions found to those which only satisfy its associated constraint `c1`.

The ordering of the resources/commitments, plus the value labeling strategy are interlinked, as certain labeling strategies re-order the variables too. The labeling strategies used are standard strategies provided by most CSP systems, such as first fail, bisect domain, max, min [8]. We do however, add additional prioritising strategies to these to weight constraint labeling in certain ways specific to our cumulative scheduling CSP. We have 3 main strategies, all of which can be specified at run-time (Section 7 discusses in more detail methods for utilising these strategies):

- Favour new commitment: Will find solutions that break any existing commitments to satisfy new commitments.
- Favour existing commitments: Will find solutions that break the least existing commitments for the new commitment.
- Favour specific agent: Used in conjunction with the previous strategies but will try to keep as many of the favoured agent's commitments as possible while carrying out the existing or new commitments strategies.

6. Implementation

The cumulative scheduling constraint program is written using the SICStus Prolog finite domain constraint library [9], which allows us to construct finite domains as sets of integers, and define constraints over one or more of these domains. This library also includes the ability to reify those constraints. Our reification mechanism itself is defined as a simple predicate of the form:

```
reify(+R,+C,+V,+N,+S,-RES)
```

Where `R` is a list of the resources available, `C` is a list of constraints on those resources, `V` is the time period over which value `N` is required and `S` is the strategy to be used by the agent for assigning values and reifying the constraints to produce a list `RES` of results (such as those shown in Figure 5). The strategies available for `S` are described in section 5.2.

We have embedded this solving mechanism in an agent implemented using the JADE agent platform[5]. The agent is

5 http://sharon.cselt.it/projects/jade/

AAMAS'04, July 19-23, 2004, New York, New York, USA.

```
****************************************
REQUIRED RESOURCES
N: 15 x : 0 -> 10
****************************************
CURRENT RESOURCES
12 x-a1 : 0 -> 10 -Using 5 x-c1: 0 -> 5,
                   Using 3 x-c2: 6 -> 10
10 x-a2 : 0 -> 10 -Using 5 x-c3: 0 -> 7
****************************************
SETTING UP TIME SLICES & TOTALS
Setting up 0 - 10 for x-a1 - total 12
Setting up 0 - 10 for x-a2 - total 10
CHECKING EXISTING COMMITMENTS
****************************************
SETTING UP NEW REQUIREMENT
****************************************
FINDING SOLUTIONS
****************************************
Solution(1)-
contributions to new commitment
 [0,0,0,0,0,0,0,0,0,0] : (x-a1)
 [0,0,0,0,0,0,0,0,0,0] : (x-a2)

REIFIED CONSTRAINTS
 [0,0,0,0,0,0,0,0,0,0]  - [N][0] 15x:0->10
 [1,1,1,1,1,_,_,_,_,_]  - [c1][1]
 [_,_,_,_,_,_,1,1,1,1]  - [c2][1]
 [1,1,1,1,1,1,1,_,_,_]  - [c3][1]
****************************************
Solution(2)-
contributions to new commitment
 [12,12,12,12,12,12,12,12,12,12] : (x-a1)
 [3,3,3,3,3,3,3,3,3,3] : (x-a2)

REIFIED CONSTRAINTS
 [1,1,1,1,1,1,1,1,1,1]  - [N][1] 15x:0->10
 [0,0,0,0,0,_,_,_,_,_]  - [c1][0]
 [_,_,_,_,_,_,0,0,0,0]  - [c2][0]
 [1,1,1,1,1,1,1,_,_,_]  - [c3][1]
****************************************
Solution(3)-
contributions to new commitment
 [5,5,5,5,5,5,5,5,5,5] : (x-a1)
 [10,10,10,10,10,10,10,10,10,10] : (x-a2)

REIFIED CONSTRAINTS
 [1,1,1,1,1,1,1,1,1,1]  - [N][1] 15x:0->10
 [1,1,1,1,1,_,_,_,_,_]  - [c1][1]
 [_,_,_,_,_,_,1,1,1,1]  - [c2][1]
 [0,0,0,0,0,0,0,_,_,_]  - [c3][0]
****************************************
Solution(4)-
contributions to new commitment
 [12,12,12,12,12,12,9,9,9,9] : (x-a1)
 [3,3,3,3,3,3,6,6,6,6] : (x-a2)

REIFIED CONSTRAINTS
 [1,1,1,1,1,1,1,1,1,1]  - [N][1] 15x:0->10
 [0,0,0,0,0,_,_,_,_,_]  - [c1][0]
 [_,_,_,_,_,_,1,1,1,1]  - [c2][1]
 [1,1,1,1,1,1,0,_,_,_]  - [c3][0]
****************************************
```

Figure 5. Example program output

queryable as a service using FIPA ACL with the message content specified in RDF [5].

The communication between Prolog and the JADE platform is through Jasper[6], a close coupling Java/Prolog interface that allows us to use Java as the parent application and load in the SICStus prolog runtime kernel into the JVM and instantiate and use it as a SICStus object.

7. Discussion

The purpose of this research into the management of an agent's commitments is to offer a general method for answering the question: "Can I meet a new request for resource/service provision, and, if not, what are my options?" In the example used throughout this paper, introduced in section 3, we have considered a simple case in which there are three prior commitments and a request to adopt a further commitment, such that the agent is left with the decision to reject the new request or drop at least one of its prior commitments.

In developing a good answer to the question posed we must first employ a language that is sufficiently expressive to capture real situations of resource use over time. Then, given this language for describing commitments, we require reasoning machinery that may be used to determine whether an arbitrary set of commitments may be met by the resources available — used to determine whether a new request may be accommodated without breaking existing commitments — and, if not, what minimal sets of existing commitments may be dropped to accommodate a new request. As has been argued within this paper, cumulative scheduling and constraint reification techniques offer a general and well-founded solution to this problem.

This solution — the generation of a set of options for resolving an over-constrained problem — is, however, partial. From this set of options, the agent must make a decision (possibly in collaboration with other members of the coalition) on which option is best (for the coalition as a whole). It is in answering this broader question that we see how the research reported in this paper complements decision theoretic mechanisms for selecting between such options such as leveled commitment contracts [11]. The introduction of penalties for unilateral decommitment from contractual obligations is a commonly employed mechanism for binding agents to their agreements and specifying (typically financial) reparation to the other party [7]; the other type of penalty commonly considered in the literature, being a loss of reputation [6].

In making a decision on which deal in a set of possible options available, D, the agent must take into account the following information: the revenue expected from tak-

6 http://www.sics.se/sicstus/docs/latest/html/sicstus.html/Jasper.html

	revenue per unit delivered	total revenue	decommitment penalty
c1	0.2	5	5
c2	1	12	5
c3	0.2	7	5
n	0.1	15	

Table 1. Example estimated revenues and penalties.

ing on the new commitment(s), the revenue lost from commitments to be released under a deal and any decommitment penalties that will be imposed. Let us assume that each deal, d, that is considered an option comprises of the new commitments that will be made under that deal, d_n, the commitments that remain unchanged under the deal (commitments to be kept), d_k, and those that are released, d_r. Also assume that, for each commitment, c, we are able to compute the expected revenue obtained, $rev(c)$, and assume that each commitment has a fixed penalty, $pen(c)$. We may then compute the best, or optimal deal, d^{opt}, from the set of options, D, in the following manner:

$$d^{opt} = \arg\max_{d \in D} \left(\sum_{c \in (d_n \cup d_k)} rev(c) - \sum_{c \in d_r} pen(c) \right)$$

Returning to the example introduced in section 3, suppose that the revenue expected for each commitment, c1, c2 and c3, and the new request, n, and the decommitment penalties for c1, c2 and c3 are shown in table 1. Note that all the decommitment penalties are the same, but the revenue per unit resource for each commitment varies (the client of c2, for example, is paying a premium). Considering these three options suggested by the scheduler and using the criterion given above for a deal, the options are evaluated as 24 for schedule 1, 12 for schedule 2 and 27 for schedule 3. Therefore, in the circumstances, the best option open to the agent is to accept the request and decommit c3.

Selecting and following d^{opt} is not necessarily the only possibile use of the information provided by the constraint solver. The information provided identifies a set of possible and minimal fixes to the agent's commitments that enable it to take on a new request. In the above, we have assumed that the agent has a static set of resources available to meet its commitments, but these resources are made available by members of a coalition, the use of which is being coordinated by this agent. The resources contributed by members of the coalition do not necessarily represent the resources that may, in fact, be available. One coalition member may, for example, have agreed in the past to make 50% of its resources available to the coalition, keeping the remainder for one-off deals outside the context of the coalition. It may be possible for the coalition agreement to be renegotiated to accommodate the new request without reneging on existing commitments. The options found can then be used guide this negotiation. Furthermore, the coalition manager may use background information about the expected availability of certain resources to further guide its search for a solution. For example, it may be aware that resources of type x are in great demand, but that resources of type y may be more freely available from members of the coalition. In this case, the coalition manager may consider options that break commitments to the provision of y resources before those that break commitments to x resources.

The possibilities considered so far for coalition manager when it encounters an over-constrained commitment management problem following the receipt of a request are to simply reject the request, break one or more existing commitments or to seek to renegotiate coalition agreements to transform the over-constrained problem into one that has a solution. These are not the only possibilities, however. The coalition manager may also seek to renegotiate the existing agreements it has with clients, again guided by the computation of which commitments are most highly valued (determined by sorting the set of options D using a criterion such as that discussed above). The manager may also seek to expand the coalition or out-source some of its commitments if this is sufficiently profitable (or if a loss is acceptable if it serves to uphold the reputation of the coalition).

8. Conclusion

In this paper, we have shown how the use of a cumulative scheduler employing constraint reification can provide a general and well-founded means for an autonomous agent to manage a set of resources. In particular, the technique allows the agent to identify options for fixing an over-constrained resource-use problem. These options may be used to guide the reformation of coalitions, or the re-negotiation of contracts between agents. Thus, over time, the set of commitments — expressed in the form of constraints — held by each agent will evolve in a cycle of scheduling, reification, (re-)negotiation, and coalition (re)formation. This employment of cumulative scheduling with reification is novel in the multi-agent systems context.

Currently, this decision-making component is being integrated and tested within the overall CONOISE architecture and testbed environment described in 6. This work is a development of previous work in KRAFT [3, 4], in propagating and solving mobile constraints imposed by various suppliers, within a configuration design context. KRAFT

AAMAS'04, July 19-23, 2004, New York, New York, USA.

lacked a mechanism for systematically relaxing constraints, which has now been rectified by our cumulative scheduler employing constraint reification.

Immediate future challenges include employing quality-of-service information in choosing between alternative service provisions, and linking the results of the deliberation process to the CONOISE negotiation algorithms. For further information, see: www.conoise.org

Acknowledgements

The CONOISE project is jointly funded by the DTI/EPSRC E-Science Core Programme and BT Exact – British Telecom's research, technology and IT operations business.

The project is a collaboration between BT Exact and Aberdeen, Cardiff and Southampton Universities.

References

[1] P. Baptiste, C. Le Pape, and W. Nuijten. Constraint-based scheduling–applying constraint programming to scheduling problems. In *International Series in Operations Research and Management Science*, volume 39. Kluwer Academic Publishers, 2001.

[2] Y. Caseau and F. Laburthe. Cumulative scheduling with task intervals. In *Logic Programing Proceedings of the 1996 Joint International Conference and Syposium on Logic Programming*, pages 363–377, 1996.

[3] P. M. D. Gray, S. M. Embury, K. Hui, and G. J. L. Kemp. The evolving role of constraints in the functional data model. *Journal of Intelligent Information Systems*, 12:113–137, 1999.

[4] P. M. D. Gray, K. Hui, and A. D. Preece. Finding and moving constraints in cyberspace. In *Intelligent Agents in Cyberspace*, pages 121–127. AAAI Press, 1999. Papers from the 1999 AAAI Pring Symposium Technical Report SS-99-03.

[5] K. Hui, S. Chalmers, P. Gray, and A. Preece. Experience in using rdf in agent-mediated knowledge architectures. In *Agent-Mediated Knowledge Management: Papers from the 2003 AAAI Spring Symposium*, volume SS-03-01, pages 82–89. AAAI Press, March 2003.

[6] M. Klein, J.-A. Rodriguez-Aguilar, and C. Dellarocas. Using domain-independent exception handling services to enable robust open multi-agent systems: The case of agent death. *International Journal of Autonomous Agents and Multi-Agent Systems*, 7:179–189, 2003.

[7] M. J. Kollingbaum and T. J. Norman. Supervised interaction: Creating a web of trust for contracting agents in electronic environments. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 272–279, 2002.

[8] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 1992.

[9] C. M., O. G., and C. B. An open-ended finite domain constraint solver. *Proc. Programming Languages: Implementations, Logics, and Programs*, 1997.

[10] T. J. Norman, A. Preece, S. Chalmers, N. R. Jennings, 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. *Research and Development in Intelligent SystemsXX: Proceedings of AI2003, the Twentythird SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 353–366, 2003.

[11] T. Sandholm and V. Lesser. Leveled-commitment contracting: A backtracking instrument for mulitagent systems. *AI Magazine*, 23(3):89–100, 2002.

AAMAS'04, July 19-23, 2004, New York, New York, USA.