Integrating Collaboration and Activity-Oriented Planning for Coalition Operations Support

Clauirton Siebra and Austin Tate

Centre for Intelligent Systems and their Applications
School of Informatics, The University of Edinburgh
Appleton Tower, Crichton Street, EH9 1LE, Edinburgh, UK
{c.siebra,a.tate}@ed.ac.uk

Abstract. The use of planning assistant agents is an appropriate option to provide support for members of a coalition. Planning agents can extend the human abilities and be customised to attend different kinds of activities. However, the implementation of a planning framework must also consider other important requirements for coalitions, such as the performance of collaborative activities and human-agent interaction (HAI). This paper discusses the integration of an activity-oriented planning with collaborative concepts using a constraint-based ontology for that. While the use of collaborative concepts provides a better performance to systems as a whole, a unified representation of planning and collaboration enables an easy customisation of activity handlers and the basis for a future incorporation of HAI mechanisms.

1 Introduction

Coalitions, from Latin *coalescere* (*co*-, together + *alescere*, to grow) is a type of organisation where joint members work together to solve mutual goals. The principal feature of any coalition is the existence of a unique global goal, which motivates the actions of all coalition members. However, normally such members are not directly involved in the resolution of this goal, but in sub-tasks associated to it. For example, a search and rescue coalition that aims to evacuate an island (evacuate island is the global goal) has several sub-tasks (e.g., refill helicopters, provide information about weather conditions, etc.) that must be performed to reach the global goal.

The basic idea of this paper is to support coalition members via assistant agents that provide planning facilities. In this scenario each member could have a personal agent that supports his/her activities and delivers, for example, planning information, coordination commands and options to carry out activities. This approach is powerful because while human users have the ability to take decisions based on their past-experiences (case-base reasoning), agents are able to generate and compare a significant number of options, showing both positive and negative points of such options. Projects as *O-Plan* [1] and *TRAINS* [2] explore this fact, providing planning agents that interact in a mixed-initiative style with users during the development of solutions, such as plans.

However the use of standard planning mechanisms is not sufficient to deal with the complexity of problems associated to coalition domains, such as disaster relief operations. In these domains, activities cannot consist merely of simultaneous and coordinated individual actions, but the coalition must be aware of and care about the status of the group effort as a whole [3]. The *Helicopters Attack Domain* [4] and the *Guararapes Battle* game [5] are two experiments that corroborate with this affirmation. The principal reason for the problems in such domains is that "collaboration between different problem-solving components must be designed into systems from the start. It cannot be patched on" [6]. Thus, the idea of our approach is to develop the collaborative framework together with the planning mechanism in a unified way, using a constraint-based ontology for that. The properties of constraint manipulation have already been used by several planning approaches as an option to improve their efficiency and expressiveness [7]. Constraints are especially a suitable option to complement the abilities of HTN planning as, for example, to represent possible resultant subproblems dependences of the decomposition process [8]. The use of constraints will also facilitate a future expansion of our system toward a better human support. Note that constraints are a declarative way of providing information so that users specify what relationship must hold without specifying a computational procedure to enforce that relationship. Consequently users only state the problem while the computer tries to solve it [9].

The remainder of this document is structured as follows: section 2 presents some initial aspects of our approach associated to hierarchies, planning representation and process. Section 3 summarises the collaborative formalism that our models are based on, explaining how we are modelling its principal concepts using the constraint-based ontology and associated functions. Finally section 4 concludes this work.

2 Hierarchical Coalitions and Conceptualisation

This section introduces three important concepts for our work: the coalition organisation, the planning representation and the activity-oriented planning process.

Hierarchies are a well-known and used structure for organising members of a team. Military organisations, for example, are the most common examples of hierarchical arrangements. One of the principal advantages of hierarchies is that they support the deployment of coordination mechanisms because such mechanisms can exploit their hierarchical organisational structure. This is because the organisation implicitly defines the agents responsibilities, capabilities, connectivity and control flow. In addition, hierarchies also have the following advantages:

- They are compatible with the *divide-and-conquer* idea so that the process of splitting a problem into smaller sub-problems is repeated in each level;
- Hierarchical levels may deal with different granularities of knowledge so that each of them does not need to specify all the details about the problem, and;
- It is possible to enclose problems in local subteams, instead of spreading them along the entire organisation.

We are considering hierarchical organisations arranged into three levels of decision-making: strategic, operational and tactical. However this is not a "must restriction" so that hierarchies can be expanded to *n* levels. A last important aspect of hierarchical coalition systems is that there is a need of customising their agents so that they are able to support specific activities carried out into each decision-making level.

Our planning representation is based on <I-N-C-A> (Issues - Nodes - Constraints - Annotations) [10], a general-purpose constraint-based ontology that can be used to represent plans in the form of a set of constraints on the space of all possible plans in the application domain. Each plan is considered to be made up of a set of Issues and Nodes. *Issues* may state the outstanding questions to be handled and can represent unsatisfied objectives, questions raised as a result of analysis, etc. *Nodes* represent activities in the planning process that may have sub-nodes (sub-activities) making up a hierarchical description of plans. Nodes are related by a set of detailed *Constraints* of diverse kinds such as temporal, resource, spatial and so on. Annotations add complementary human-centric and rationale information to the plan, and can be seen as notes on their components.

The system architecture considers planning as a two-cycle process, which aims to build a plan as a set of nodes (activities) according to the <I-N-C-A> representation. In this way, the first cycle tries to create candidate nodes to be included into the agent's agenda, respecting its current constraints. If this process is able to create one or more candidate nodes, one of them can be chosen and its associated constraints are propagated, restricting the addition of future new nodes. The agents' agenda contains the net of activities that agents intend to perform. If an agent receives a new activity, it must generate *actions* that include this new node in its agenda. Each action is a different way to perform this inclusion so that different actions generate different nodes configurations. We call this process of *activity-oriented planning* because agents provide context sensitive actions (e.g., delegations, Standard Operating Procedures, dynamic plan generation and specific solvers) to perform activities.

In brief, the role of agents is to provide actions to decompose nodes until there are only executable nodes. Each action is implemented by an activity handler, which uses one or more different constraint managers to validate its constraints. For example, the action of applying a SOP is a handler that decomposes an activity according to the SOP specification. For that, the handler uses specific constraint managers that check the pre-conditions in which the SOP can be applies, signalising in case of conflicts.

3 The Collaborative Framework

Planning agents are able to support the performance and coordination of activities, however they do not ensure collaboration between coalition members. For that, coalition systems must also consider notions of collaboration since their phase of conception. Considering such fact, this section summarises the collaborative formalism that we are exploring and how we are implementing its ideas following the <I-N-C-A> approach.

3.1 Requirements of the Teamwork Theory for Collaboration

The *Teamwork Theory* [11] provides a set of formal definitions that lead the design of collaborative systems. Several works have proposed frameworks based on such definitions. *SharedPlans* [12], for example, argues that each collaborative agent needs to have mutual beliefs about the goals and actions to be performed, and about the capabilities, intentions and commitments of others agents. *STEAM* [13] is an implemented model of teamwork, where agents deliberate upon communication necessities during the establishment of joint commitments and coordination of responsibilities. Although these and others works have different approaches to deal with different technical problems, an investigation on these works can distinguish the following principal requirements: (1) agents must commit on the performance of joint activities; (2) agents must report the status of their activities, informing on progress, completion or failure, and; (3) agents must mutually support the activities of other coalition members, sharing useful information or using their own resources for that.

3.2 Activities Commitment

We are implementing both ideas of activities commitment and status reporting through the "AgentCommitment" function. This section analyses the commitment of activities, while the next section (Section 3.3) discusses the status reporting process.

According to <I-N-C-A>, each plan p is composed by a set of plan nodes n_i. If a superior agent, that has p as goal, sends such nodes n_i to its subordinate agents, then a commitment between the superior and each subordinate must be done. For that end, the function "AgentCommitment", implemented by each agent of the hierarchy, receives as input parameter the node $n_i \in p$ and the *sender* identifier.

```
01. function AgentCommitment(sender,n_i)
02.    subplan ← GenerateNodes(n_i)
03.    if(∃(subplan))
04.       if(HasNodesToBeDelegated(subplan)) then
05.         Delegate(subplan,subord) ∧ WaitCommits()
06.         if ∃s (s ∈ subord)∧(¬Commit(s)) then
07.           go to step 04
08.       Report(sender,n_i,COMMITED)
09.       while(¬Complete(subplan))
10.         if(JustReady(subplan)∨Changed(subplan))then
11.           Report(sender,n_i,EXECUTING)
12.         if(Violated(subplan)∨Receive(FAILURE)) then
13.           go to step 04
14.       end while
15.       Report(sender,n_i,COMPLETION)
16.    else
17.       Report(sender,n_i,FAILURE)
18.       ∀s (s ∈ subord) ∧ HasCommitment(s,subplan_s),
19.         Report(s,subplan_s,FAILURE)
20. end
```

At this point we can discuss some implications and features of this function. First, n_i has a set of constraints associated to it so that the "GenerateNodes" function (step 02) considers such set to return an option (nodes list or *subplan*) to perform n_i. If a subplan is possible (step 03) and it does not depend of anyone more (step 04) then the agent can commit on n_i (step 08). However, if *subplan* depends on the commitments of subordinates, then the agent must delegate the necessary nodes to its subordinates and wait their commitments (step 05). This implies that commitments are done between a superior agent and their subordinates and, starting from the bottom, a "upper-commitment" only can be done if all the "down-commitments" are already stabilised. Note that a delegation on the same *subplan* cancels previous commitments.

Second, an interesting implication is that if some subordinate agent is not able to commit (step 06), the agent returns (step 07) to generate other subplan option rather than sending a failure report to its superior. This approach implements the idea of enclosing problems inside the sub-coalition where they were generated.

Finally if the agent is not able to generate a *subplan* for n_i, it reports a failure to its superior (step 17). In addition, it must also alert their subordinates that n_i has failed and consequently its subnodes can be abandoned (step 18-19).

We can note that the principal advantage of this approach is that commitments are naturally manipulated during the constraint processing as any other constraint. In other words, the failure in a commitment is interpreted as a problem of constraint satisfaction, which can trigger a common process of recovery, such as a replanning.

3.3 Status Reporting

After reporting a commitment (step 08), an agent a_i must monitor and report execution status until the completion/failure of n_i. The principal questions here are when to send a report and which information should be reported. There are two obvious cases in which a_i performing n_i must report: first when a_i completes n_i, and second when happens an execution failure and a_i is not able to deal with the failure by itself. In the first case a_i only needs to send a completion report, while in the second case a failure report must be sent optionally with the failure reasons (constraints unsatisfied).

Reports associated to progress are a more complex case. In order agents do not have to communicate each step of their execution. Previous works have already identified communication as a significant overhead of risk in hostile environments [13]. Furthermore, in case of progress reports, the information associated with them should be useful to the monitoring process.

Considering these facts, we start from the principle that relevant information, once sent, becomes common knowledge and hence unnecessary to updates. For example, if a_i informs that it has just started the execution of n_i, a_i does not need to send other messages informing that it still executing n_i. From this point we must think on which could be the information generated during activities execution and that is likely to help superior agents during the process of monitoring. For that end, consider the following scenario: when a_i commits on the performance of n_i, it informs which *subplan* it is going to use. The constraints of such *subplan* can have the following classes of values: *concrete*, which expresses known values or estimated values if a_i has a good idea

about the process; and *variables* to be used during unpredicted situations, indicating that a_i does not have idea about some specific information.

Using constraints with concrete values, the superior agent of a_i can perform more confident reasoning on the ongoing activities. For example, if it knows that the activity of a_i will take 30 minutes, it can allocate a_i to a new activity after 30 minutes. Thus, if during the execution of *subplan* one of its concrete values is changed or its variables are instantiated, a progress report must be sent.

Returning to the function, while the planning is not complete (step 09), the changes in *subplan* (step 10) are monitored and sent to superiors as a ongoing execution report (step 11). Constraint violations and failure reports are also monitored (step 12) so that the agent firstly tries to repair the problem by itself (step 13) before sending a failure report. Note that this discussion expands the plan (or subplan) status from the three early discussed (possible, no-ready, impossible) to five (complete and executing additionally). The <I-N-C-A> definition for activities contains a status attribute that can be filled with one of these options.

At last, the principal point of this function is that the reasoning associated to commitments and reports are based on results of constraint processing. This fact is illustrated by the functions "Complete" (step 09), "JustReady" (step 10), "Changed" (step 10) or Violated (step 12), which are all applied on constraints. Thus, we still having the same basis for working, which is also used by the planning mechanisms.

3.4 Mutual Support

The principal idea behind mutual support is to enable that one agent has knowledge about the needs of other agents. For example, an agent a_1 knows that a specific road is clean so that it uses this constraint in its plan. However, as the world is dynamic, the road can be blocked. If any other agent finds out that such road is no more clean, it must inform this fact to a_1. Thus this informer agent is supporting the performance of a_1. The easier option to implement this feature is to force agents to broadcast any new fact to all coalition. Consequently all agents will have their world state updated and problems like that can be avoided. However, this is not an appropriate approach in terms of communication and agents will also receive several useless information.

Consider now that Θ_x is a coalition and that each agent $a_i \in \Theta_x$ has a plan p_i with a set of conditional constraints C, which a_i desires that hold so that p_i still valid. In this case each $c_i \in C$ is a constraint that a_i believes that is true and hopes that still being true. Then a_i broadcasts C for Θ_x so that other agents of its subgroup know what it needs. Such agents must deal with C according to the function below:

```
01. function MutualSupport(a_i,C,myBel)
02.    while(∃c_i c_i∈C)
03.       if(∃c_i c_k c_i∈C ∧ c_k∈myBel ∧ Contrast(c_i,c_k)) then
04.          newActivity ← CreateActivity(Goal(c_i))
05.          if(¬∃newActivity) then Inform(a_i,c_k)
06.          Retire(c_i,C)
07.       if(∃c_i c_i∈C ∧ ¬Valid(c_i)) then Retire(c_i,C)
08. end
```

According to this function, which is applied by agents that receive C from a_i, agents must compare their believes *myBel*, which are also a set of constraints, with C. If they find some "contrast" (step 03), they must try to create a new activity whose goal is to turn c_i true (step 04). If this is not possible, they must inform a_i that c_i is no more holding and its new value is c_k (step 05).

The Contrast function (step 03) says that two constraints, which are supposed to be equal, are not equal. However we are also considering as contrast the situation where two constraints have the potential to be equal. For example, ((color Car) = ?x) and ((color Car) = blue). In this case the two constraints are in contrast because they have the potential to be equal if the variable ?x assume the value "blue". This second type of contrast is very useful in the following class of situations. Suppose that one of the activities of a_i is "rescue injured civilians". For that end, a_i firstly needs to find such civilians so that they have the following conditional constraints: ((position ?a) = ?b), ((role ?a) = civilian) and (status ?a) = injured). This set of constraints implies that the variable ?b is the location of an injured civilian ?a. Then if other agents that have or discover a set of constraints that contrast with the set sent by a_i, they must inform a_i (note that in this case no make sense to create a new activity).

The Valid function (step 07) accounts for eliminating the constraints that no more represent conditions to sender agents. This is important to avoid that agents still sending useless information and also to decrease the number of messages in the coalition. A practical way to do that is to consider that all $c_i \in C$ has a timestamp that indicates the interval that such constraint is valid. Using a timestamp as $(t_{initial}, t_{final})$ and considering that $t_{initial}$ and t_{final} are ground values, the Valid function only needs to compare if the condition (t_{final} < current-time) is true to eliminate the respective constraint. However, the use of timestamps fails if sender agents does not know the interval that their conditional constraints must hold. Note that the use of timestamps tries to avoid that agents need to broadcast messages saying that they no more need that a group of constraints hold. Rather, timestamps enable that agents reason by themselves on the elimination of such constraints.

One of the principal advantages of the MutualSupport function is that it improves the information sharing because the sending of information is guided by the constraint-based knowledge that each agent has on the activities of its partners. In addition, the function also decreases the number of messages changed among agents.

4. Conclusion and Directions

This work shows the integration of an activity-oriented planning with notions of collaboration via a unified constraint-based framework. This framework enables an easy development of activity handlers, which can be customised according to the tasks of each decision-making level. Ongoing experiments of this proposal are using the RoboCup Rescue simulator as principal domain of evaluation. The first idea is to demonstrate the importance of coordination and collaboration during coalition operations. However, the principal purpose is to show that the development of planning mechanisms can be maintained independent of the collaborative framework.

Acknowledgement

Clauirton Siebra' scholarship is sponsored by CAPES Foundation under process number BEX2092/00-0. This material is based on research within the I-X project sponsored by the Defense Advanced Research Projects Agency (DARPA) and US Air Force Research Laboratory under agreement number F30602-03-2-0014 and other sources.

The University of Edinburgh and research sponsors are authorised to reproduce and distribute reprints and on-line copies for their purposes not withstanding any copyright annotation here on. The views and conclusions contained here in are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of other parties.

References

1. Tate, A.: Mixed-Initiative Interaction in O-Plan. Proceedings of the AAAI Spring Symposium on Computational Models for Mixed-Initiative Interaction, Stanford, California, USA (1997)
2. Fergunson, G., Allen, J., Miller, B.: TRAINS-95: Towards a Mixed-Initiative Planning Assistant. Proceedings of the Third Conference in AI Planning Systems, AAAI Press, Menlo Park, California, USA (1996) 70-77
3. Levesque, J. Cohen, P., Nunes, J.: On Acting Together. Proceedings of the Eighth National conference on Artificial Intelligence, Los Altos, California, USA (1990) 94-99
4. Tambem, M.: Towards Flexible Teamwork. Journal of Artificial Intelligence Research, Vol. 7 (1997) 83-124
5. Siebra, C., Ramalho, G.: An Architecture to Support Synthetic Actors in Virtual Environments, Second Brazilian Workshop on Artificial Intelligence, Rio de Janeiro, Brazil (1999)
6. Grosz, B.: Collaborative Systems, AI Magazine, Vol. 17-2 (1996) 67-85
7. Nareyek, A., Fourer, R., Freuder,, E., Giunchiglia, E., Goldman, R., Kautz, H., Rintanen, J., Tate, A.: Constraints and AI Planning, Notes from the AAAI Workshop on Constraints and AI Planning, Austin, Texas, USA (2000)
8. Stefik, M.: Planning with Constraints (MOLGEN: Part 1). Artificial Intelligence, Vol. 16-2 (1981) 111-140
9. Freuder, E.: Synthesizing Constraint Expressions. Communications ACM, Vol 21-11 (1978) 958-966
10. Tate, A.: <I-N-C-A>: an Ontology for Mixed-Initiative Synthesis Tasks. Proceedings of the IJCAI Workshop on Mixed-Initiative Intelligent Systems, Acapulco, Mexico (2003)
11. Cohen, P., Levesque, H.: Teamwork, Special Issue on Cognitive Science and Artificial Intelligence, Vol. 25 (1991) 487-512
12. Grosz, B., Hunsberger, L., Kraus, S.: Planning and Acting Together, AI Magazine, Vol. 20-4 (1999) 23-34
13. Tambe, M.: Towards Flexible Teamwork, Journal of Artificial Intelligence Research, Vol. 7 (1997) 83-124