A Unified Approach to Planning Support in Hierarchical Coalitions

Clauirton de Albuquerque Siebra

Doctor of Philosophy

Centre for Intelligent Systems and their Applications

School of Informatics

University of Edinburgh

2006

Abstract

Hierarchical coalitions are organisations whose components carry out different planning and plan execution activities at each level. One of the principal aims of knowledge-based tools for coalitions is to support such components so that they are able to work synergically together, each of them accounting for part of the planning and execution process.

The use of planning assistant agents is an appropriate option to provide this kind of support. Agents can extend the human abilities and be customised for different planning activities performed along hierarchical decision-making levels. However, the use of standard planning mechanisms is not sufficient to deal with the complexity of problems associated with coalition domains. In these domains, activities cannot consist merely of simultaneous and coordinated individual actions, but they must also be developed on a collaborative framework that ensures an effective mutual support among joint members.

This thesis analyses groups of requirements associated with the development of joint human-agent planning agents, showing that they can be implemented, in a unified way, via a constraint-based ontology and related functions. The constraints' properties have already been used by several planning approaches as an option to improve their efficiency and expressiveness. This work demonstrates that such properties can also be employed to implement collaborative concepts, which are kept transparent to the planning mechanisms. Furthermore, the use of constraints provides several facilities to the implementation of advanced mechanisms associated with the human interaction, as also demonstrated here.

The practical aspects of such an approach are illustrated via a prototype that uses a disaster relief domain as a test-bed. The role of this prototype is to show: (1) the impact of collaborative concepts in the planning process; (2) the opportunities for human-agent interaction, and; (3) the easy customisation of agents through the definition of activity handlers and specific constraint managers. Finally, an additional domain associated with space applications is also discussed so that we can testify the generality of this approach.

Acknowledgements

To be completed...

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Clauirton de Albuquerque Siebra*)

Table of Contents

List of Figures

List of Tables

Part I

Background

Chapter 1

Introduction

Coalitions are organisations whose members work together to solve mutual goals. This chapter presents the research field of knowledge-based tools for coalition operations, starting by characterising the kind of domain that coalition members normally operate. The focus is on the performance of distributed planning and execution activities, and the challenges that they raise to the development of coalition support systems. Based on this discussion we define the specific problems that this thesis is concerned with and the contributions of our approach in this direction. Finally an overview of the remaining chapters is presented.

1.1 Coalition Support Systems and their Challenges

Coalition, 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 a coalition is the existence of a global goal, which motivates the activities of all coalition members. However, normally such members are not directly involved in the resolution of this goal, but in sub-tasks associated with it. For example, in a search and rescue coalition that aims to evacuate an island (evacuate island is the global goal), there are several sub-tasks (e.g., allocate helicopters, provide information about road conditions, etc.) that must be performed to reach the global goal. The following real scenario can be used as a basis to illustrate some common aspects of coalition domains.

Japan is a region prone to earthquakes due to the number of tectonic plates

> that converge below the country's surface. The Kobe Earthquake of January 1995 is an example of how disasters have tragic effects in urban areas. More than 6.500 civilians were killed, 80.000 houses fully collapsed and the number of sufferers were more than 1 million. Several events can happen during a large earthquake: buildings collapse, civilians are buried under collapsed buildings, fires spread along the city, roads are blocked causing problems for the evacuation of civilians and the work of rescue teams.

The Kobe Earthquake has the typical features of a scenario that requires the deployment of a coalition. Such features are:

- The nature of the problem demands a set of different abilities (fire brigades, rescue teams, paramedical professionals, police officers, military and civil defence personnel, etc.) to accomplish different tasks (extinguish fire, find buried people, take care of injured civilians, evacuation control and so on);
- Participants need to collaborate because they have limits in abilities and knowledge. For example, ambulances require a clear route to rescue injured civilians. If the route is blocked, ambulances must ask for help from truck units so that they unblock the route. Participants also collaborate during the planning process, each of them accounting for the part of the plan which it has more knowledge about;
- The rescue organisation requires a structure of command and control that coordinates the activities of participants. Coordination is also essential to avoid conflict and improve the use of time and resources. A coordination process avoids, for example, that three ambulances try to rescue the same injured civilian;
- The anywhere/anytime feature of disasters (such as earthquakes) requires that coalitions be rapidly created and flexibly changed as circumstances alter;
- There is the problem of integrating heterogeneous systems belonging to multiple organisations (hospitals, fire centres, etc.) with distinct doctrines and operational rules. This problem is more complex if we consider international coalitions where actions are misunderstood or lost in translation across cultural interfaces;

- The example of urban disaster relief domain relates to the area of *Operations Other Than War* (OOTW) [Walker, 1999], which also involves peacekeeping, peace-enforcing and non-combat evacuation domains. Such scenarios are good motivations for our work.

Research in knowledge-based tools for coalition operations considers one or more of these features during the development of theories and applications. The *CoAX Project* [Allsopp et al., 2002], for example, demonstrates the agent-based paradigm as a suitable way to deal with the technical issues of integrating different technologies for coalition support.

Within the CoAX system, heterogeneous components are seen as a set of distributed agents that are able to share understanding and information among themselves through a computing infrastructure called the *CoABS Grid* [Schmorrow, 2002]. The CoAX project contemplates operations that need to rapidly integrate various non-integrated and "come-as-you-are" systems, allowing capabilities to be assembled at run-time. CoAX was demonstrated in the *Binni Coalition Scenario* [Rathmell, 1999], a hypothetical scenario based on the Sudanese Plain (Africa) where a number of typical exercises for coalition forces can be simulated. During the CoAX demonstration in Binni, several systems (Ariadne [Knoblock and Minton, 1998], I-X Process Panels [Tate et al.,2002], etc.) were integrated to support an international war avoidance force in this virtual region. Details about this demonstration can be seen in [CoAX, 2004].

In brief, the current topics covered by knowledge-based tools for coalition operations include[1]:

- Innovative theory and techniques for coalition formation and support to similar virtual organisations;

- Applications and requirements for knowledge-based coalition planning and operations management;

- Knowledge-based approaches to command and control, coalition logistics and OOTW;

[1]Based on the proceedings of the Second International Conference on Knowledge Systems for Coalition Operations.

- Multiagent systems and the concept of agency in coalitions;
- Tools and techniques for simulation and modelling of coalition operations;
- Security and maintenance of private information or knowledge;
- Autonomous vs. centrally managed coalition operations.

The International Conference on Knowledge Systems for Coalition Operations (Edinburgh/1999, Toulouse/2002, Florida/2004) is the principal forum of discussion for research in this area. The focus of the work are on Military OOTW domains, however the technology generated can be extended to any other kind of coalition domain.

1.2 Examples of Coalition Domain

A coalition can be deployed in any domain that contains the features discussed in Section 1.1. During this thesis the practical examples are mostly associated with disaster relief domains. However the examples can be easily mapped to others, such as military and space exploration domains. Figures 1.1 illustrates such domains.

Urban disaster relief domains (Figure 1.1a) require the union of agents that support activities related to search and rescue. The coalition instance in Figure 1.1a is based on the *RoboCup Rescue simulator* [Kitano and Tadokoro, 2001], which we have used during our evaluations (details in Chapter 9). Agents are represented by fire brigades, ambulances, police forces, fire stations, ambulance centres, police offices and a search and rescue command centre. The scenario represents a real part of Kobe City (Japan).

Figure 1.1b illustrates an example of a military domain where multinational forces integrate abilities during operations of peacekeepers, such as in the fictional Binni scenario. Agents are represented by the head country's Department of Defence, local operational agents and several military units. Multinational coalitions are a typical example of coalitions that require the development of shared representations reflecting different cultures, doctrines and languages. Groups such as the *Multinational Planning Augmentation Team* (MPAT) [MPAT, 2004a] work in this direction, trying to develop and become familiar with *Standard Operating Procedures* (SOP) [MPAT, 2004b], which represent well-founded and tried plans to be applied in specific situations.

Finally Figure 1.1c illustrates a futuristic scenario where robots and astronauts collaboratively work in interplanetary missions, such as a Mars Mission. Agents are represented by the Earth Mission Control Centre, Mars Habitats and units of astronauts and robots. A real example in this context is the *Aurora Project*, a long-term effort of the European Space Agency (ESA) that aims to send a team like that to Moon in 2024 and to Mars in 2033.

These domains also illustrate the nature of coalition structures. We can note that there are different levels of decision-making where components deal with different activities and knowledge. These levels compose a hierarchical structure, which has an important influence associated with the development of planning processes. Chapter 2 discusses in detail aspects associated with hierarchies.

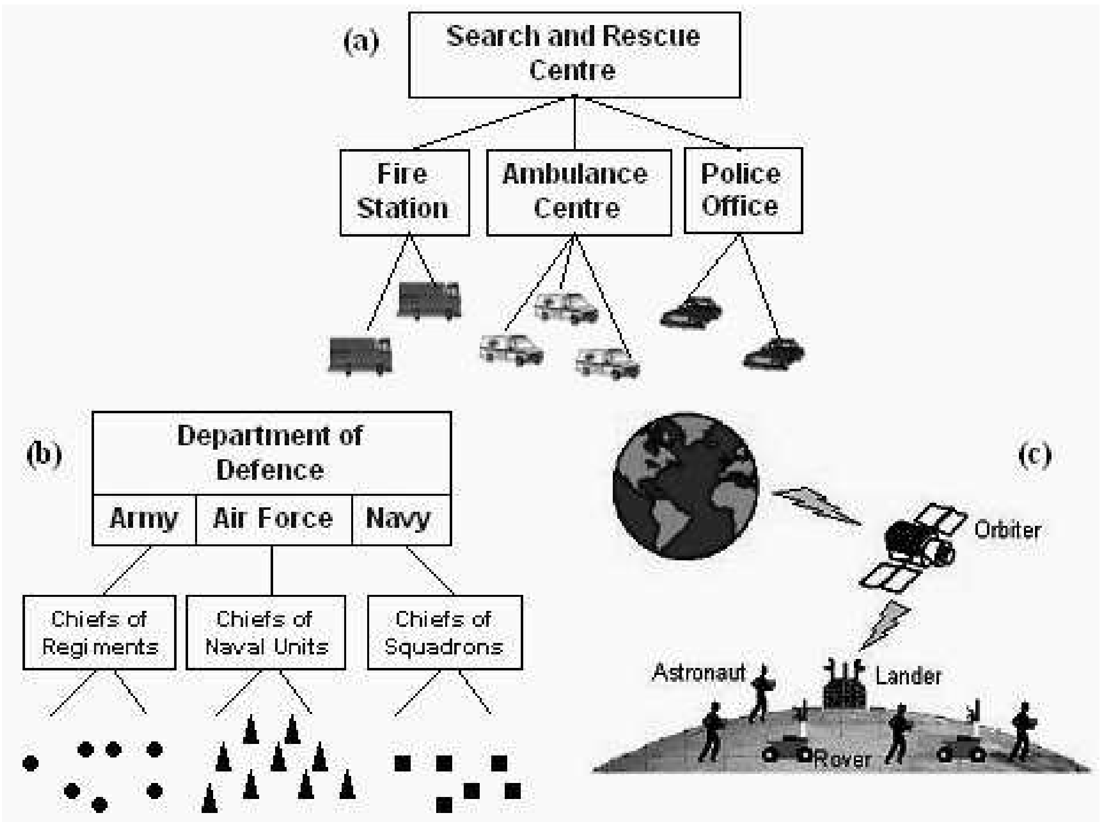


Figure 1.1: Examples of coalition domains: (a) disaster relief, (b) military and (c) space exploration domains.

1.3 Problem Definition

This thesis, in particular, is concerned with the specification of a framework to the development of hierarchical coalition support systems. Our main idea is to consider requirements associated with the implementation of multiagent planning mechanisms [Sycara, 1998] because they can bring several advantages to coalition operations such as prediction of failures, resource allocation, conflict identification and so on. However this framework must also be able to support other important requirements for coalition systems, related to collaborative theories and human-agent interaction. The problem is that the requirements that we want to consider are investigated by different research areas rather than in a unified way. Consequently the solutions are generally incompatibles or hard to integrate and there is a lack in the current literature about frameworks that discuss this issue.

This section discusses the relation between requirements, planning representation and mechanisms, highlighting the limitations of systems that do not consider all the requirements in a unified way.

1.3.1 Assistant Agents and their Customisation

We are delivering planning mechanisms to users via assistant agents. In this approach each participant has an agent that supports his/her tasks, providing for example, planning information and options to carry out activities. This approach is powerful because while users have the ability to take decisions based on their past-experience (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 such as O-Plan [Tate, 1997] and TRAINS [Ferguson et al., 1996] explore this fact, providing planning agents that interact in a mixed-initiative style with users during the development of a solution, such as a plan.

As introduced in the last section, the coalitions that we are working with are typically based on a hierarchical structure. Coalition members placed in levels of this hierarchy perform different and complementary planning activities. For example, the members located at the top level of the coalition presented in Figure 1.1a (search and rescue command centre) will account for directions and high level plans. Their planning perspective is very unlike the perspective of the hierarchical bottom members (fire brigades, police forces and ambulances), which are the members that are, in fact, executing the plan in the environment.

Considering this fact, agents that support coalition members at different levels of decision-making must be customised so that they are able to support the planning activities carried out at each level. From a practical perspective, this customisation means that agents must provide activity handlers (e.g. pathfinder, load balancing, etc.) to support a specific set of activities (Figure 1.2). Such handlers work on a planning representation, which expresses, for example, notions of environment, time, resources, priority, activity, state, etc.

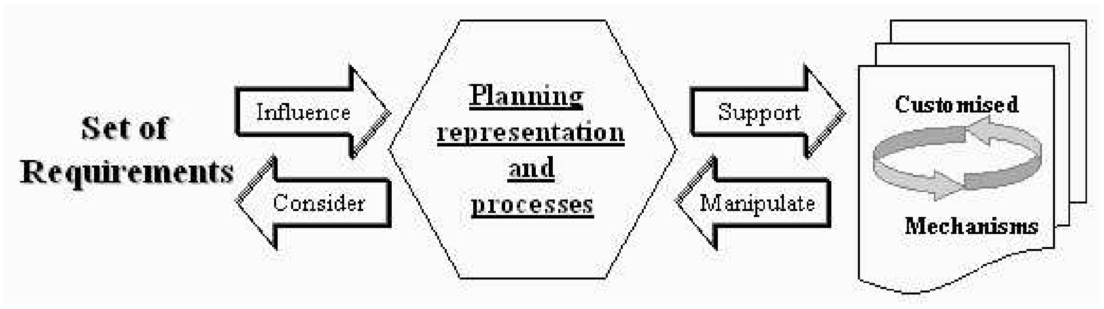


Figure 1.2: Relation between planning requirements, representations and mechanisms.

However, when agents are performing as part of a coalition, the complexity of the planning process increases due to the number of requirements that must be considered. Requirements associated with collaboration, distributed planning, coordination and user interaction are interrelated and the design of such one can influence others. Thus the implementation of individual solutions for each set of requirements is not a good practise. Part II of this thesis analyses in details such requirements and their role inside the planning representation.

1.3.2 Principal Limitations of the Current Research

Consider the *Multiagent Planning Architecture* (MPA) [Wilkins and Myers, 1998], for example. MPA could be an option to coalition support systems because it is a framework for integrating diverse technologies into a system capable of solving problems that cannot be solved by individual systems. Furthermore MPA is distinguished from other agent architectures [Moran et al., 1997] in its emphasis on large-scale planning applications, such as coalition operations.

MPA's agents are organised around the concept of planning cells. Each cell has a *manager agent* that accounts for composing the planning cell from a pool of available agents and distributing the tasks among the selected planner agents. The shared information, generated during the planning process, is kept in a plan server (central repository) and any agent of the cell can access it. Using this approach MPA integrates several separated stand-alone software systems (OPIS [Smith, 1994], Tachyon [Arthur and Stillman, 1992] and SIPE-2 [Wilkins et al., 1995]) that cooperate on large-scale problems.

MPA is an interesting approach to the integration of different planning solutions. However this integration presents limits for coalition support systems development, as any other approach that tries to incorporate existent tools and capabilities into a simple framework. The principal reason, as discussed in [Grosz, 1996], is that *collaboration between different problem-solving components must be designed into systems from the start. It cannot be patched on*. The problem here is how to incorporate collaborative requirements into a distributed planning process, so that the final joint plan is not just a sum of individual plans. Chapter 4 analyses this problem with more details, discussing some approaches to deal with it.

Considering that one of the most important functions of knowledge-based tools for coalitions is to support human users, we also need to understand how they interact in the collaborative planning process. Note that we are not considering aspects of interface that could improve the human-agent interaction [Pegram et al., 1999]. Our focus is on the study of the additional requirements that the human presence brings to the development of collaborative planning agents. This important issue for coalition systems is not well explored by noteworthy collaboration theories, mainly because such theories only consider agent-agent interactions rather than human-agent interactions [Tambe, 2003]. Chapter 5 discusses in details this problem.

1.3.3 Specific Issues Addressed in this Thesis

The principal claim of this thesis is that we can integrate distinct groups of requirements (multiagent planning, collaboration and human interaction), associated with the development of hierarchical coalition support agents, via a unified framework provided by a constraint-based ontology and related functions. We argue and demonstrate that such framework brings several advantages for the agents' development, such as: well-known environment to represent and build plans; transparent way to incorporate collaborative concepts, which complement the planning abilities; opportunities for the development of more advanced human-agent mechanisms; and support to an easy customisation of activities handlers.

Based on this claim, we can formulate specific issues that lead to the development of this thesis. Such issues are:

- Formalisation of hierarchical coalitions (members, relations and rules among them) so that we can use the structural features of this kind of organisation for the development of command and control mechanisms (coordination);
- Investigation and categorisation of requirements that have influences on the development of models and processes used by agents operating in hierarchical structures;
- Specification of a unified representation of planning and collaboration that enables an easy customisation of activity handlers and an appropriate basis for the incorporation of requirements associated with human-agent interaction;
- Development of practical applications that demonstrate the real advantages of this approach and also stress its generality.

The development of collaborative activities into a hierarchical way, as carried out in this work, receives some criticisms [Grosz, 1996, Nwana et al., 1996] due mainly to the master-servant relationship between members. The question is if consensus about "what to do" and "who does what" must be reached via negotiation between a team of equals, or if they can be simply imposed to members of a team. Hierarchical organisations are closer to this latter option. In this way, it is important to show that the notion of collaboration is not lost inside a hierarchical organisation but, on the contrary, it can take advantages of several features of this structure.

The final outcome of this thesis is a unified approach that enables the configuration of coalition support systems for different kinds of domains. As collaboration has been designed as part of the planning process, we avoid the specification of additional domain-specific plans, which could be needed to ensure team collaboration. Note that the principal theories that have investigated the idea of collaboration only consider their approaches for multiagent systems. We advance this idea by looking upon the influences of human interaction during the collaborative process of agents. The new requirements associated with such human influences are also considered together with the planning and collaboration requirements, keeping the uniform representational feature of this approach.

1.4 Thesis Structure

The remainder of this thesis is organised as follows: Chapter 2 introduces the hierarchical architecture that we are using to organise the coalition components (humans and respective agents), showing how planning activities are performed along its levels. The discussion mainly considers coalitions composed of three levels of decision-making (strategic, operational and tactical), showing that components into different levels will typically have different perspectives of the coalition plan as well as different interests and objectives. Then we present a set of concepts associated with this architecture that will be used during the dissertation.

Part II investigates which are the requirements for planning development in coalition support systems. To do this, Chapter 3 explores the standard planning requirements, considering the *Hierarchical Task Network* (HTN) [Erol et al., 1994] the natural approach to perform planning activities in hierarchical structures, together with the use of constraint posting techniques, which complement the abilities of HTN planning. Chapter 4 shows that the simple use of planning does not ensure an effective collaboration among members of a coalition. Then we analyse four important theories associated with teamwork, which provide a fundamental set of requirements to guide the extension of the planning approach. Chapter 5 considers influences of human users on the collaborative planning process, going beyond the teamwork theories, which are only based on agent-agent (multiagent) interactions. In brief, the principal idea of these three chapters is to generate a robust set of requirements that lead the development of a hierarchical organisation of assistant agents for coalition members.

Part III details our unified representation that embodies all the requirements previously discussed via constraint-based models and operations on them. Chapter 6 discusses <I-N-C-A> [Tate, 2003], the constraint-based general-purpose ontology that we are using and extending to represent the models. Chapter 7 describes the specification of the models and their properties, which are based on the requirements discussed in Part II of this thesis. Chapter 8 presents the process of planning and execution, highlighting the development and use of activity handlers and constraint managers, and how they ensure the satisfaction of the coalition requirements.

Finally Part IV discusses the practical perspective of this thesis, together with its conclusions and work directions. Then, Chapter 9 describes results of this approach when applied to a disaster relief domain based on the RoboCup Rescue Simulator for the Kobe earthquake. Chapter 10 specifies the *Satellite Constellation* domain, whose principal objective is to demonstrate the generality of our approach when applied to a completely different kind of scenario. Chapter 11 concludes our dissertation by highlighting the contributions, final conclusions and possible extensions of the work described in this thesis.

Chapter 2

Abstract Architecture and Conceptualisation

This thesis considers the use of hierarchies as an option to organise members of a coalition. Hierarchies have a long history of being used as a coordination (command and control) structure, such as in military organisations. One of the principal advantages of hierarchies is their support for the *divide-and-conquer* concept. Furthermore, using hierarchies we are able to specify several levels of decision-making rationale where participants perform different and complementary activities, dealing with specific tasks and information.

This chapter starts by justifying the choice of hierarchy as an organisational structure for coalition coordination, showing its advantages and comparing it with other possible alternatives. Then we discuss the influences of this kind of organisation on the planning process. Based on this discussion we introduce a particular hierarchical instance, composed of three decision-making levels: strategic, operational and tactical. The idea here is to characterise each level in terms of the processes that it needs to support, and show how the planning works along this hierarchy. Finally we present a formal description of hierarchies and additional concepts, which are used during the discussions in the following chapters.

2.1 Hierarchies as Structure for Coordination

Coordination can be defined as a process in which agents engage in order to ensure their team acts in a coherent way, avoiding conflicts with one another and behaving as a unit. In multiagent systems coordination is treated as a sophisticated pattern of interaction, together with cooperation and negotiation [Jennings et al., 1998], and several approaches have been proposed such as the use of *blackboard systems* [Hayes-Roth, 1985] or the *contract net protocol* [Smith, 1988].

There are four principal reasons why the actions of multiple coalition members need to be coordinated [Jennings, 1990]:

- There are dependencies between their actions, either because local decisions made by one member have some impact on the decisions of other members (e.g., a fire brigade wants to use a road, but the police force is blocking such road) or because the possibility of harmful interactions among members (e.g., two firemen may attempt to use the same fire extinguisher);
- There is a need to meet global constraints. For example, astronauts' space-walk missions must be planned respecting the oxygen reserves of the station;
- Rather than members possessing a global view of the whole coalition, each of them only has local views, goals and knowledge which may conflict with others;
- Generally, no one individual member has sufficient information, competence or resources to solve an entire problem. A simple task of rescuing a buried civilian, for example, can involve several members to lift heavy objects.

Hierarchies are a well-known and widely used structure for coordinating members of a team. Military organisations are the most common examples of such a hierarchical arrangement. Since ancient times, mainly represented by the *Roman Legions*, until the current time, military units have been using hierarchies as the way to improve the efficiency of command and control mechanisms.

The use of hierarchies facilitates the deployment of coordination mechanisms because such mechanisms can exploit the hierarchical organisational structure. This is because such an 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. The process of splitting a problem into smaller subproblems is repeated at each level;
- Hierarchical levels may deal with different granularities of knowledge so that each level does not need to have all the details about the problem;
- It is possible to enclose problems to be dealt with by local subteams, instead of spreading it over the coalition.

On the other hand, there are also critics on the use of hierarchies to achieve coordination in teams [Nwana et al., 1996]. First, superior members may exert much control over the subordinates' actions and hence their deliberative process. This fact mitigates against some benefits of Distributed AI (DAI), such as concurrency and minimal bottlenecks. However this limitation can be bypassed/minimised if we consider that the role of upper members is only to say *WHAT* subordinates members must do, rather than *HOW* they must do.

Second, the approach presumes that at least one agent has a global view of the entire coalition, in terms of plans, conditions and capabilities. However this limitation can be relaxed if we consider that each member has a complete view of itself and its roles, but only a partial view of others. An instance of this approach is implemented by the DSIPE distributed planning framework [DesJardins and Wolverton, 1999]. In DSIPE, each agent has a complete model of its own subplan and a partial model of subplans being developed by other agents. Section 7.2.3 returns to this problem, presenting a method to deal with this limitation.

Finally, designers should ensure that the hierarchical division is of sufficient granularity to compensate for the overheads which result from goal distribution and communication. The problem is that the distribution of small tasks can be more expensive that performing them in a single location [Durfee et al., 1987]. This is not an exclusive limitation of hierarchies, but of any distributed problem solving organisation. Section

2.3 shows that, in terms of logical arrangement, an example of a hierarchical organisation may be seen as a three-level structure, where each level has a well defined role associated with the planning process. We argue that such a well-defined division can help the process of goal distribution.

Considering these features, we can say that the use of hierarchies is a suitable option to support coordination in coalitions. Other possible approaches to coalition coordination could be: the blackboard architecture, contract net protocol assuming a decentralised market structure, or negotiation. However each of these approaches present disadvantages for a coalition's coordination.

In the blackboard schema members post to and read from a general blackboard. In this way there is not a direct member-to-member communication, resulting in a dangerous bottleneck. Differently, each member of a hierarchy can directly communicate with its superior or subordinates without the need of an intermediary component.

In a contract net protocol a manager announces a task to a set of contractors. Each contractor bids for the task and the manager awards the task to the contractor with the best bid. This approach is mainly limited because it does not presume members with contradictory demands. So it is not possible to detect or resolve conflicts. In the most simplistic form, hierarchies can ultimately detect and solve conflicts when superior members merge the solutions of their subordinates.

Finally coordination also can be achieved via negotiation [Sycara, 1989]. Negotiation is a type of interaction between two or more components that aims to reach a mutually accepted agreement on some issue. Despite the fact that negotiation is present in a significant part of the coordination research, it is a complex and time-consuming kind of interaction. Therefore, in domains that are time-critical, such as disaster relief operations, it could be appropriate to avoid such interaction. Note that we are interested in systems that support coalitions during the operation itself, rather than in a pre-operation time where negotiation processes could be in fact necessary. Furthermore, the members of a coalition are not self-interested. Thus they are not looking for taking advantage of some situation.

Architectures that are not based on a hierarchical structure find support in some works of collaborative system. The following scenario [Grosz, 1996], for example, is used to justify that the use of a superior/subordinate relationship is not appropriate during processes of collaboration:

> In a health care scenario, a patient arrives at the hospital with problems affecting his heart and lungs. Three specialists are needed, each providing different expertise needed for curing the patient. The cardiologist and pulmonary specialist must agree on a plan of action for reducing the water in the patient's lungs, and the infectious disease and pulmonary specialists must plan together the kind and amount of antibiotic to prescribe.

This scenario represents a team of equals because no single doctor is the manager, telling the others who does what. However scenarios like that are not in accordance with the coalition features, where the several degrees of decision that must be taken do not require the involvement of an entire coalition. For example, the fire propagation experts (team that accounts for predicting the fire evolution) do not need to interact with fire brigades to take decisions about important positions to be protected during a disaster relief operation. In fact, one of the interesting advantages of hierarchies, as commented early, is this power of dividing logical levels of decision-making, avoiding the need for members to deal with knowledge that is not important for them. However, we must note that the kind of decision taken by teams of equals could be important inside of each level. Figure 2.1 illustrates this scenario.

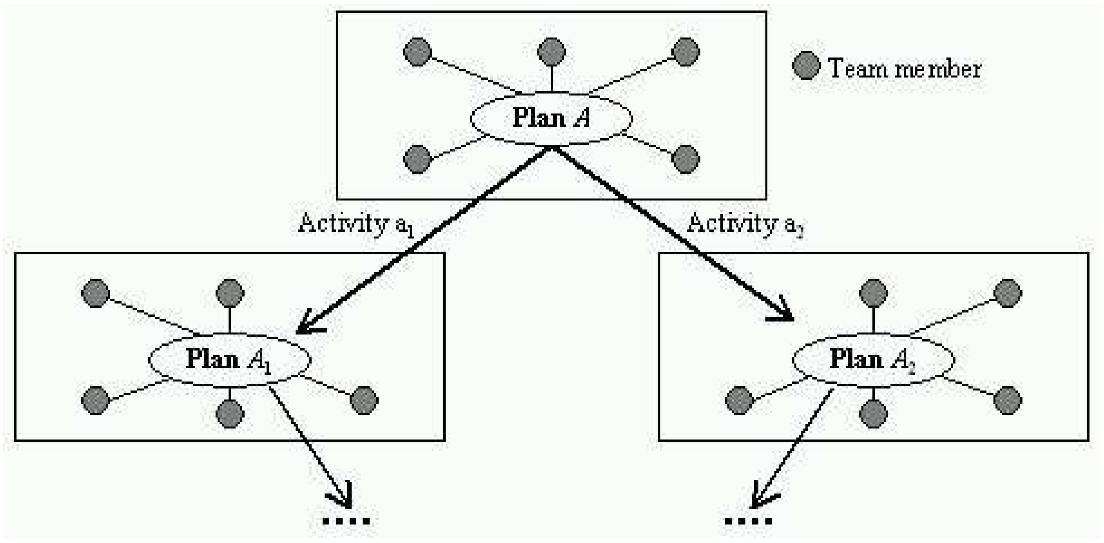


Figure 2.1: Teams of equals performing inside of decision-making levels.

According to the figure, teams of equals could be organised inside each level so that they agree on solutions before sending commands to down levels. Such an idea is also important when a member has two or more superiors. As each superior accounts for generating commands, two or more superiors will probably be a source of conflicts because they can have antagonistic goals. In this way, superiors must reach a mutual agreement before interacting with their subordinates.

2.2 Hierarchies and Planning

The planning process along hierarchies is comprised of individual plan steps. At the upper or more abstract levels, each plan step achieves more effects than a plan step at a lower level. The bottom of a hierarchy is composed of primitive plan steps, which are the activities that an agent can directly execute.

The simplest way of performing planning along hierarchies is to consider a conflict-free context. In this case, subplans can be independently created at lower levels and, after that, directly merged at upper levels. However this kind of context is not common in real domains.

To bypass this problem, planning in hierarchies follows a centralised or a distributed approach. In the centralised approach [Georgeff, 1983], superior agents receive all partial plans from their subordinates, analysing them in order to identify potential inconsistencies and conflicting interactions. Then, superior members attempt to modify these partial plans and combine them into a plan where conflicting interactions are eliminated.

Conversely, in the distributed approach the idea is to provide each member of the hierarchy with a model of the other members' plans, such as in the DSIPE system [DesJardins and Wolverton, 1999]. Such members must communicate in order to build and update their individual plans and their models of others' until all conflicts are removed. Sections 7.2.3 and 7.2.4 show that our work takes a middle-ground approach between these two options.

Independently of the approach used to perform planning processes along hierarchies, we must consider that coalition members, placed at different hierarchical levels, deal with different activities to create their individual (sub)plans. In this way, there is a need for customising the planning support according to such activities. A first step in this direction is to characterise the kind of activities carried out at these levels. An option is to analyse each level via its inputs, outputs, processes and knowledge required by such processes (Figure 2.2).

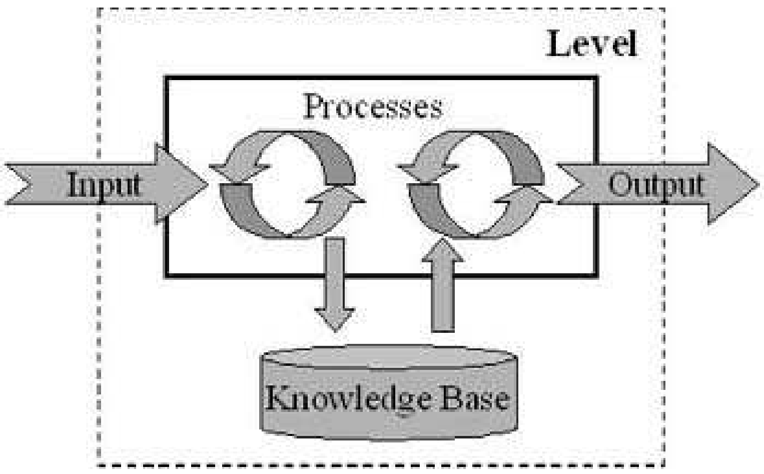


Figure 2.2: Characterising a level via input, output, processes and required knowledge.

The elements of the figure can be defined as below:

- Inputs: are tasks in the form of commands, goals or activities that a level receives and needs to deliberate on, so that some kind of feedback can be generated. Usually inputs are the triggers of processes;
- Outputs: are the results of processes, representing primitive activities or requests for the performance of complementary activities;
- Knowledge: are the set of facts that a level needs to support its processes. Generally levels have a pre-defined knowledge about their domain, however this knowledge evolves with the perception of new facts via interaction with both environment and other levels;
- Planning processes: are the reasoning mechanisms that act according to inputs and use the level's knowledge to produce some kind of output. In this way, processes are strongly related to the inputs (tasks) that the level must deal with.

We must notice that this analysis supports the design of organisations because it helps in identifying planning processes that are strongly related. Thus such processes can be grouped in the same level rather than being broken between two or more levels. As discussed before, the distribution of trivial or small tasks can be more expensive than performing them in a single location. In this way, the organisational design needs to ensure that subordinates are of sufficient granularity to compensate for the overheads which result from the distribution of activities.

2.3 Decision-Making Levels

Based on Figure 2.2, this section analyses a particular instance of a hierarchical organisation, which identifies three levels of decision-making (Figure 2.3): strategic, operational and tactical[1]. This hierarchical arrangement is a common practise in military models of command and control [Killion, 2000, Ferguson, 2004, U.S Marine, 1994] and consistent with knowledge engineering work[2]. Furthermore, this section also highlights the need of customising planning processes at each of these levels.

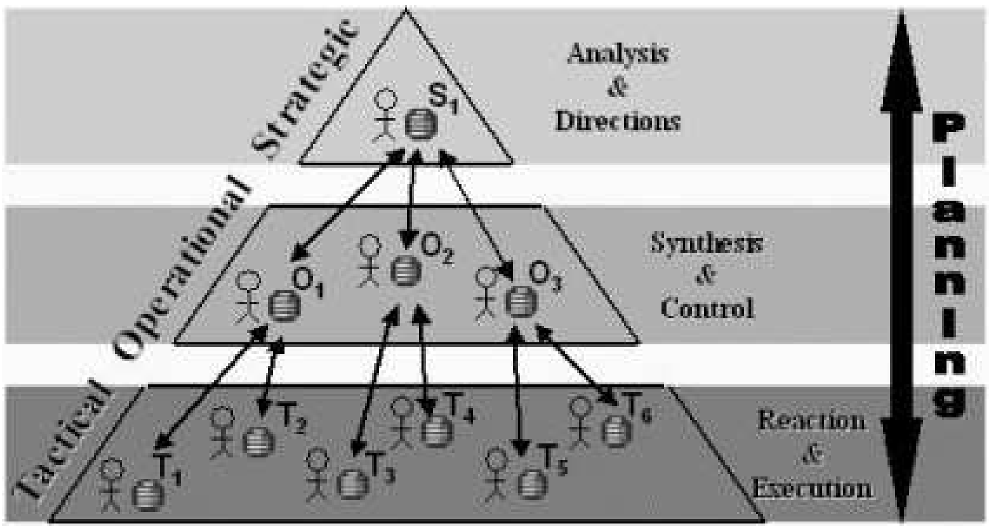


Figure 2.3: Abstract idea of a three level hierarchical organisation.

[1]Note that this order, which is used in military domains, is often changed to "strategic, tactical and operational" in business domains.

[2]KADS methodology [Schreiber et al., 1999], for example, separates the different "task types" (diagnosis, interpretation, monitoring, etc.) into three classes: analysis, synthesis and modification tasks.

2.3.1 Strategic Level

The strategic level accounts for developing plans in a high level of abstraction, or "what-to-do" plans. In other words, the level specifies what must be done, but it does not give details about how something must be done. In this way, the principal tasks are related to analysis, directions and comparison of courses of actions. Table 2.1 summarises the features of this level.

Feature	Description
Input	Generally a complex and abstract task
Output	Requests for the performance/filling of "what-to-do" plans
Time	Long-term goals
Influence	The entire coalition is affected by its decisions
Knowledge	Global, diversified and non-technical
Processes	Problem analysis, definition of directions and priorities

Table 2.1: Strategic level features.

Considering a disaster relief domain (Figure 1.1a), the strategic level could be represented by the *Search and Rescue Command Centre* (SRCC). Just after an earthquake, the SRCC receives the tasks of rescuing injured civilians and decreasing the damages in the city. Analysing the problem, SRCC decides to divide the city into regions and set priorities for each of them (some regions can be more critical than others because they have a higher probability of having buried civilians, historic value such as museums and monuments, or present risks of increasing the catastrophe such as deposits of fuel and explosives).

SRCC can also analyse global information, such as speed and direction of wind to predict the fire behaviour and generate tasks to avoid future causalities. Possible outcomes of its deliberative process are: avoid the fire spread to region x, look for buried civilians in buildings of region y, keep unblocked the road z (because it is an important path to access resources), and so on. Note that such outcomes say what must be done without references on how they must be done. Furthermore they are long term goals, which can affect the entire coalition.

2.3.2 Operational Level

The operational level accounts for refining the plans produced at the strategic level, mainly providing the logistical resources for them via processes of resource scheduling and load balancing. Thus, knowledge about the operation environment is more detailed and limited coalition groups will be affected by the decisions. Table 2.2 summarises the features of this level.

Feature	Description
Input	What-to-do plans and possible restrictions on their performance
Output	Requests for the performance of specific tasks
Time	Mid-term goals
Influence	One or more sub-coalitions are affected by their decisions
Knowledge	More specialised, mainly on the operation environment and resources
Processes	Synthesis of plans, resource allocation, load balancing, etc.

Table 2.2: Operational level features.

The operational level could be composed of local units such as fire stations and hospitals. When such components receive subgoals from the strategic level, they start by checking the necessary conditions and options to reach the subgoals, according to their available resources. In this way, operational components are taken decisions at a different level because they are thinking in how the activities can be carried out.

Each operational unit has the function of employing its subordinates to attain specific goals through the design, organisation, integration and conduct of sub-operations. For that, each unit has its own skills and abilities so that its knowledge is more specialised in the field in which it is operating. This level also pays a significant attention in the relation resource/time. This means an efficient and balanced use of resources. Thus, processes such as automatic task allocation and load balancing are very useful.

2.3.3 Tactical Level

The tactical level is where the execution of operations actually takes place. For this reason the degree of knowledge of tactical components is very specialised on the domain that they are operating, and their decisions are generally taken on sets of atomic activities. As the components are performing inside a dynamic and unpredicted environment, their reactive capabilities and speed of response are very important so that the use of pre-defined procedures could be useful alternative. The output of this level is a set of atomic activities that are commonly executed by the own components. Table 2.3 summarises the tactical level features.

Feature	Description
Input	Specific tasks and possible restrictions on their performance
Output	Primitive operations (atomic activities)
Time	Short-term goals
Influence	Decisions ideally should not have influences on other levels
Knowledge	Very specialised
Processes	Pathfinder, patrolling, reactive procedures, knowledge sharing, etc.

Table 2.3: Tactical level features.

The tactical level, in a disaster relief operation, could be arranged by fire brigades, paramedics and police forces for example. For the performance of their tasks, these components could need specific intelligent processes such as a pathfinder, which looks for best routes to specific destinations, or patrolling mechanisms to trace routes that efficiently cover search areas. The tactical level is also the principal source of new information to the coalition because its components are in fact moving through the environment. In this way they are more propitious to discover changes and new facts that must be shared among their partners.

2.4 A Formal Description for Hierarchical Coalitions

The definition of hierarchies as organisational structures for coalitions is the first step toward the definition of a joint human-agent planning framework. However, a more formal description of such structures is important to be used as a basis for future discussions, so that related concepts can be introduced on a same perspective. Figure 2.4 illustrates the idea of a general hierarchy.

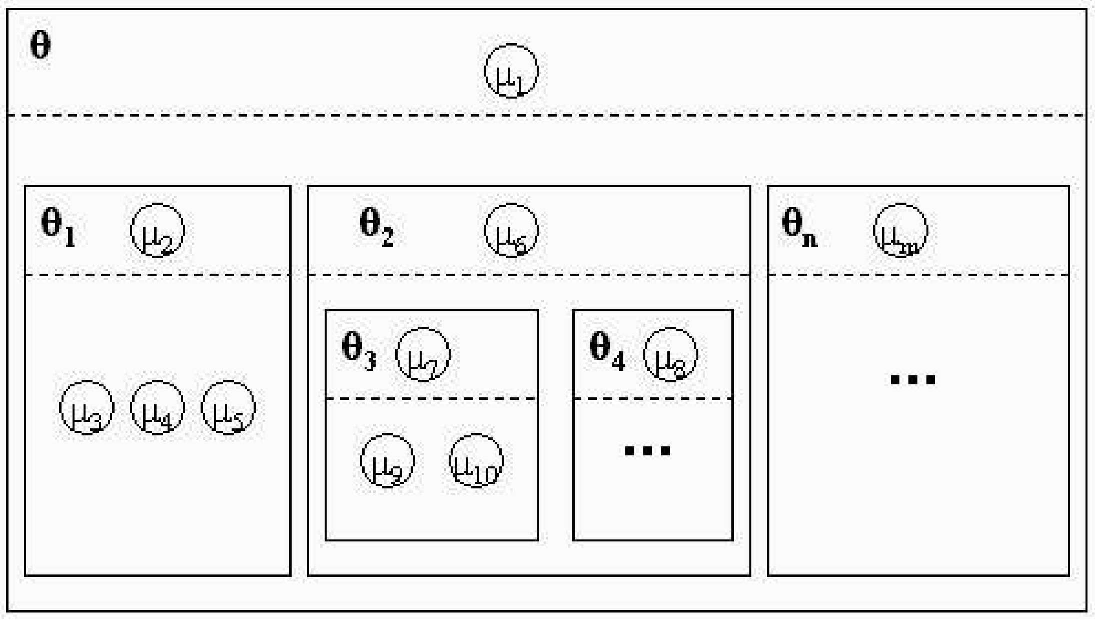


Figure 2.4: Example of a hierarchical coalition description.

Components (agents) that form a hierarchy are represented by μ_i, where i is an integer value from 1 to n (total number of components). We can define the following functions on hierarchical components:

- LEVEL(μ_i), returns the level of μ_i. The notion of levels is introduced through the idea that components at the same depth belong to the same hierarchical level;
- RELATION(μ_i,μ_j), returns the relation (e.g., superior or peer) of μ_i regarding μ_j. If such components do not have a relationship, the function returns null.

Using such functions we can deduce some initial properties. First, considering two different components μ_i and μ_j, if μ_i is peer of μ_j, then they are at the same level. However the contrary is not true because components in the same level can have a null relationship. Such a property can be expressed as:

$$\forall\ \mu_i, \mu j\ (i \neq j)\ \wedge\ RELATION(\mu_i,\mu_j) = \text{ peer}$$
$$\Rightarrow\ LEVEL(\mu_i) = LEVEL(\mu_j)$$

In the same way we can deduce properties for the cases where μ_i has a superior or a subordinate relation regarding μ_j. Note that in such cases μ_i and μ_j have to be in adjacent levels (we assume the highest level as level 1).

$$\forall\, \mu_i, \mu j\ (i \neq j)\ \wedge\ RELATION(\mu_i, \mu_j) = \text{ superior}$$
$$\Rightarrow\ LEVEL(\mu_j) - LEVEL(\mu_i) = 1$$

$$\forall\, \mu_i, \mu j\ (i \neq j)\ \wedge\ RELATION(\mu_i, \mu_j) = \text{ subordinate}$$
$$\Rightarrow\ LEVEL(\mu_i) - LEVEL(\mu_j) = 1$$

We are assuming that components can only set relations with components of their level or adjacent levels. Thus, the difference between their levels is 0 or 1:

$$\forall\, \mu_i, \mu j\ (i \neq j)\ \wedge\ RELATION(\mu_i, \mu_j)\ \neq \text{Null}$$
$$\Rightarrow\ |LEVEL(\mu_i) - LEVEL(\mu_j)|\ \leq\ 1$$

Relationships inside a coalition are always between two components. Each relationship also defines a communication channel between the components so that they can exchange useful messages for the performance of their plans. Messages can be represented by the tuple $< \mu_i, \mu_j, content >$, where μ_i and μ_j are the message sender and receiver respectively, and *content* could be instances of commands, goals, activities, feedback, facts and so on. The kind of relationship between μ_1 and μ_2 has influence on this communication, enabling or avoiding the sending of some types of message. For example, components that have a peer-peer relationship may not be able to exchange commands between them.

An option to describe a hierarchical coalition Θ is to consider Θ a composition of sub-coalitions. To this end, we can use the tuple $< \mu_i, S_{[1..m]} >$, where μ_i is a superior agent and $S_{[1..m]}$ is a set of subordinates that can be formed by components ($\mu_{[1..m]}$) or sub-coalitions ($\Theta_{[1..m]}$). In this last case, each Θ_i can recursively be decomposed in their components or sub-coalitions. For example, to represent the hierarchy of Figure 2.4 we have:

$$\begin{aligned}\Theta &= < \mu_1, [\Theta_1, \Theta_2, \Theta_n] > \\ &= < \mu_1, [< \mu_2, [\mu_3, \mu_4, \mu_5] >, < \mu_6, [\Theta_3, \Theta_4] >, < \mu_m, [...] >] > \\ &= < \mu_1, [< \mu_2, [\mu_3, \mu_4, \mu_5] >, < \mu_6, [< \mu_7, [\mu_9, \mu_{10}] >, < \mu_8, [...] >] >, < \mu_m, [...] >] >\end{aligned}$$

Another practical way to represent sub-coalitions is to use the concept of *interaction zones*. Each interaction zone Φ_i defines a group of agents that present a direct communication between them. For example, in Figure 2.4 we could define six interaction zones with their respective agents: $\Phi_1 = \{\mu_1, \mu_2, \mu_6, \mu_m, \}$, $\Phi_2 = \{\mu_2, \mu_3, \mu_4, \mu_5\}$, $\Phi_3 = \{\mu_6, \mu_7, \mu_8\}$, $\Phi_4 = \{\mu_7, \mu_9, \mu_{10}\}$, $\Phi_5 = \{\mu_8, ...\}$ and $\Phi_6 = \{\mu_m, ...\}$. Note that the sets of agents in each zone Φ_i are always represented by one superior and one or more subordinates. In this way, the tuple $< \mu_i, S_{[1..m]} >$ can be applied to represent such sets as sub-coalitions. Considering this idea, we have the following sub-coalitions for each interaction zone: $\Theta_{\Phi_1} =< \mu_1, [\mu_2, \mu_6, \mu_m] >$, $\Theta_{\Phi_2} =< \mu_2, [\mu_3, \mu_4, \mu_5] >$, $\Theta_{\Phi_3} =< \mu_6, [\mu_7, \mu_8] >$, $\Theta_{\Phi_4} =< \mu_7, [\mu_9, \mu_{10}] >$, $\Theta_{\Phi_5} =< \mu_8, [...] >$ and $\Theta_{\Phi_6} =< \mu_m, [...] >$. In brief, a general rule for a coalition $\Theta =< \mu_i, S_{[1..m]} >$ is:

$$\text{IF } S_{[1..m]} = \mu_{[1..m]} \Rightarrow \Theta_\Phi =< \mu_i, [\mu_1, ..., \mu_m] >$$
$$\text{IF } S_{[1..m]} = \Theta_{[1..m]} \Rightarrow \Theta_\Phi =< \mu_i, [Superior(\Theta_1), ..., Superior(\Theta_m)] >$$

Using such a definition we can consider that a coalition has a number of interrelated sub-coalitions that are themselves hierarchically structured. Each sub-coalition is a stable intermediate form and can most of the time act without help from the complex structure. At this point we can apply the following function to return plans from a (sub)coalition:

- PLAN(Θ_i,p), returns the (sub)plan of a (sub)coalition Θ_i to a proposition p. The same function can be applied to return the plan of a component μ_i.

Plans are intricately linked to the idea of levels so that components on the same level share a common degree of plan abstraction. The following property can be defined to relate plans of an upper level component with the plans of their subordinates:

$$\forall < \mu, S_{[1..m]} > \ PLAN(\mu, p) \ = \ \bigcup_{i=1}^{m} PLAN(S_i, p_i)$$

This property is important to corroborate, for example, the idea of enclosing planning problems inside the sub-coalition where they were generated. In this way, if PLAN($< \mu, s_{[1..m]} >$,p) has a problem, μ must deal with such a problem together with its subordinates $S_{[1..m]}$. Only if this is not possible, μ will report the problem to its superior.

Part II

Planning Threads

Chapter 3

HTN and Constraint Posting Ideas for Multiagent Planning

Part II of this thesis investigates and lists the requirements for plan development in coalition support systems [Siebra, 2005]. The discussion is divided into three threads: implementation of a multiagent planning framework, extensions of this framework to support collaborative activities and involvement of humans during collaborative planning processes.

This chapter, in particular, discusses the requirements for multiagent planning, using the *Hierarchical Task Network* (HTN) and *constraint posting* approaches as theoretical solutions. Then we justify the use of such ideas, considering the hierarchical organisation described in the last chapter, showing how they can be applied to describe the temporal and resource planning models, which form the basis to face fundamental distributed planning problems such as conflict resolution and task allocation.

3.1 Multiagent Planning and Hierarchies

The idea behind multiagent planning (MAP) [Sycara, 1998] is to extend classical AI planning to domains where several agents can plan and act together. While planning for a single agent is a process of constructing a sequence of actions considering only goals, capabilities and restrictions of the environment; planning in a MAS domain also considers the restrictions that the other agent's activities place on an agent's course of actions.

In a MAP domain, each agent is, in fact, a planning agent rather than a simple executor of assigned pre-planned activities. This feature brings several advantages to the planning process as a whole: reduced dependence on centralised control and avoidance of a single point of failure; robust team performance in the face of unreliable, limited-range communication (agents have more autonomy); local planning resources used to solve local problems without reliance on higher levels; and high-level planning and communication only necessary when local planning fails.

However, to produce a coherent plan, agents must be able to recognise subgoal interactions so that they can avoid or resolve them. This problem is known as *conflict resolution* and there are several approaches to face it. Examples are:

- Inclusion of a static agent to recognise and resolve subplans interactions. In this approach [Rao and Georgeff, 1991], agents send their plans to this static agent, and it examines such plans looking for critical regions where, for example, contention for resources could cause them to fail. Then the static agent inserts synchronisation messages so that one agent would wait until the resource is released by another agent;

- Use of the *partial global planning* framework [Decker and Lesser, 1995], where interactions among agents take the form of communicating plans and goals at an appropriate level of abstraction. Such communications enable a receiving agent to form expectations about the future behaviour of a sending agent, thus improving agent predictability and network coherence.

Together with the development of new methods and frameworks, as exemplified above, there is a trend in MAP research to use the know-how of classical AI planning. In this way, several works have been investigating ways to extend or integrate the planning technology for single agents. For example, the *Multiagent Planning Language* (MAPL) [Brenner, 2003] extends the *Planning Domain Definition Language* (PDDL) [McDermott et al., 1998, Fox and Long, 2001] so that it supports specific features of planning in multiagent domain such as concurrent acting and synchronising on actions of unknown duration. The *Multiagent Planning Architecture* (MPA)

[Wilkins and Myers, 1998] is another example in this context. However, rather than extending previous technology, MPA provides a framework for integrating diverse single planning agents via well-defined and uniform interface specifications.

An important research direction for our work is associated with the extension of *Hierarchical Task Network* (HTN) planning [Erol et al., 1994] for multiagent environments. HTN planning is an abstraction-based plan representation that allows an agent to successively refine planning decisions. As the name itself suggests, HTN planning is the natural choice for planning performance in hierarchical organisations because such approach distinguishes actions and goals of different degrees of abstraction and importance for each level.

The work of Corkill [Corkill, 1979] is an example of effort to extend a HTN planner, in this case the NOAH (Nets of Action Hierarchies) planner [Sacerdoti, 1977], for multiagent environments. The basic planning procedure and plan representation used by Corkill are the same as those of NOAH: planning proceeds through a hierarchy of plan levels, where at any plan level a partial plan is a partial order of goals and primitive actions. Each distributed agent solves its goals in the same way as NOAH, at each level expanding each unplanned goal by finding an applicable operator that solves it. In addition, Corkill also distributes the NOAH world model and mechanisms to deal with interdependencies. For that end, each agent is provided with a consistent initial world model that has enough detail to perform local plan development. As this world model is changed during local planning, the changes are communicated to other agents. When an agent receives world-model changes from other agents, it revises its own world model and determines whether any of its locally planned actions invalidate the received world-model changes. If they do, an attempt is made to sequence its planned actions with the planned actions of the other agents so that the actions no longer interfere with each other. A set of inter-planner protocols have been developed that accomplish this ordering and detect situations where a suitable ordering cannot be established.

Two others examples of efforts in this direction are DSIPE (Distributed SIPE) [DesJardins and Wolverton, 1999] and A-SHOP/IMPACT [Dix et al., 2002]. DSIPE, a distributed version of SIPE [Wilkins, 1988] (System for Interactive Planning and Execution), is most similar to the distributed version of NOAH, however extending its ideas by focusing on communication improvements among agents and the creation of a common partial view of subplans. In a different way, the A-SHOP/IMPACT project aims to integrate HTN planning agents (A-SHOPs) via a multiagent environment, called IMPACT [Eiter et al., 1999], which provides facilities for the A-SHOP agents' interaction and coordination.

In our project we are using HTN decomposition as an abstract way of performing planning in hierarchical coalitions. However we are joining the HTN ideas with an underlying constraint-based representation of plans. In this way, the planning framework can employ powerful problem solvers based on search and constraint reasoning methods, and still retain human intelligibility of the overall process and plans that are created. The next sections show the advantages of using a constraint-based representation and the options to employ it.

3.2 Constraint and Planning

A constraint is simply a local relation among variables, each taking a value in a given domain [Bartak, 1999]. A constraint often takes the form of an equation or inequality, but in the most abstract sense it is simply a logical relation among several variables expressing a set of admissible value combinations. In other words, constraints restrict the possible values that variables can take.

Constraints have several interesting properties, some of them complementary to the abilities of HTN planning:

- They efficiently describe problems that incorporate resources and time. For example, the problem of preventing overlaps of a number of tasks with specific durations that share a resource. Note that in classical HTN, issues associated with the resources and priorities are ignored, and time is represented implicitly by means of instant transitions in a transition graph;

- While global constraints allow an easy representation of inherent disjunction in the non-overlap problem (e.g., activity A may precede activity B or activity B may precede activity A), HTN provides a natural way to stipulate global constraints on plans [Erol et al., 1994];

- Constraints are declarative so that they specify what relationship must hold without specifying a computational procedure to enforce that relationship. Thus users only state the problem while the computer solves it [Freuder, 1978]. Taking advantages of this property, users can declare set of constraints for each HTN task that control its decomposition (if it is non-primitive) or execution (if it is primitive);

- In addition, constraints may specify partial information (they need not uniquely specify the values of its variables), they are additive (the order of imposition does not matter), and they are heterogeneous (relations can be defined between variables with different domains).

Constraint Satisfaction Problem (CSP) is the process of identifying a solution to a problem which satisfies all specified constraints. The CSP structure of variables and constraints can be represented as a graph with variables and constraints as nodes. A variable node is linked by an edge to a constraint node if the corresponding constraint relation includes the variable.

Planners that employ constraints and CSP in some stage of the planning process can be grouped into three categories [Nareyek et al., 2005]: maximal graphs, complete CSP representation and constraint posting.

The technique of planning with maximal graphs requires that all constraints of the domain are posted once. In this way, the CSP is constructed such that it includes all possible plans up to a certain number of actions. As for planning problems the number of actions is not known in advance, a stepwise extension of the CSP structure must be performed if no solution can be found.

The advantage of constructing a structure with all possibilities is that decisions can easily be propagated through the whole CSP, and future decision options that become infeasible in consequence can be eliminated. However as constraints in hierarchical organisations must be stepwise/added depending on the current plan refinement state at each level rather than posted once, this technique is not suitable for them. Examples of planning systems that use maximal graphs are CPlan [Beek and Chen, 1999] and GP-CSP [Do and Kambhampati, 2000].

The complete CSP representation attempts to reformulate entire planning problems into the CSP paradigm. Differently from the approach of maximal graphs, this approach extends the idea of constraint programming so that CSP graphs can be changed dynamically instead of being defined a priori. This feature is very important for real domains, where the environment is continually changing. The notion of dynamic CSP (DCSP) [Verfaillie and Schiex, 1994] is one of the approaches to represent such situation. DCSP is a sequence of CSPs, where each one differs from the previous one by the addition or removal of some constraints. The *Extensible Uniform Remote Operations Planner Architecture* (EUROPE) [Frank et al., 2000] is an example of planner that uses the DCSP approach. Another slight different approach is the *Structural Constraint Satisfaction Problem* (SCSP) used by the EXCALIBUR system [Nareyek, 2000]. These two approaches differ because DCSP tries to revise a variable assignment with given changes to the constraint graph and does not include graph changes as part of the search. In a different way, SCSP defines solutions by specifying correctness tests, which are restrictions on admissible constraint graphs.

Despite the advantages of complete representation of plans as CSPs, this option is not suitable for our approach because a full constraint model is not able to explicitly describe important planning components such as activities and issues. In this way, as better explained in chapter 6, we are using constraint-based models, where constraints are part of the plan description together with other important components.

Considering hierarchical arrangements, the constraint posting [Stefik, 1981] approach is a good option to employ constraints during the planning process. This approach is based on additions and retractions of constraints during the planning process, where constraint satisfaction is used as an add-on function to check the satisfaction of restrictions such as availability of resources or temporal intervals.

One particularly relevant feature of constraint posting is its abilities to plan hierarchically by introducing new constraints and variables. In fact, using this approach, the CSP graph can be built stepwise so that only specific subproblems are covered within the constraint solving process at each level. Thus, upper level agents only need to deal with constraints that are necessary to define a plan according to the degree of abstraction required by their levels. The work of refining a plan, with additional constraints, is left to lower level agents. The use and additional abilities of constraint posting can be illustrated via its application in past planners, as discussed in the next section.

3.3 Constraint Posting Planners

Several planners have used the constraint posting approach during their planning process. Via the analysis of such planners, it is possible to illustrate the ideas behind this technique. For this end, this section discusses the use of constraint posting in five different planners: MOLGEN, MACBeth, Descartes, parcPlan and O-Plan.

MOLGEN [Stefik, 1981] is a hierarchical planner that uses the constraint posting approach to represent interactions between subproblems. MOLGEN proceeds hierarchically by formulating constraints of increasing detail as planning progresses. In this way, each hierarchical level that introduces new constraints does not work with all of the details at once. However, as the resultant subproblems of the decomposition process generally include dependencies, MOLGEN uses constraints to represent the interactions between them.

The key idea of MOLGEN is based on the *constraint propagation* [Bartak, 2001] technique. In constraint propagation, information deduced from a local group of constraints is recorded as a change in the given CSP. Such changes are used to make further deductions and therefore further changes. This causes the effects of originally locally constrained information to spread gradually throughout the entire CSP. Thus, constraint propagation works like a communication channel for constraints of subproblems. When such constraints are shared, they bring together the requirements from subproblems, which are used to anticipate interference between them and eliminate the interfering solutions.

MACBeth [Goldman et al., 2000] presents similar ideas to MOLGEN, also being a combination of hierarchical task network with constraint reasoning techniques. As each task is expanded by MACBeth, constraints from the corresponding method are added to a constraint management engine. In the case that more than one method applies, the planner develops all of the alternatives. While the plan is built, constraints propagate both up, from sub-tasks to tasks, and down, from tasks to their expansions. This constraint propagation process enables the planner to identify flaws in the plan, as happens in MOLGEN.

However, MACBeth extends some ideas of constraint propagation used by MOLGEN. For example, consider that a task *Extinguish*(Fire$_1$) contains the constraint *ableToExtinguish*(?fireExtinguisher,Fire$_1$), which represents a subdomain D of the fire extinguishers that are able to extinguish Fire$_1$. Rather than searching for values that satisfy this constraint, MOLGEN posts this constraint and delays selecting particular objects until the selection is further constrained by other steps during the planning process. If the resultant domain is $D = \{F_1, F_2\}$ and F_2 is allocated to another task, MOLGEN is not automatically able to allocate F_1 to the task of extinguish Fire$_1$. In a different way, MACBeth will ensure that one of these two elements of D is assigned to the "extinguish" task, keeping track of elements of D. If F_2 is later assigned to a different and unrelated task, MACBeth will automatically assign F_1 to the "extinguish" task, with no additional introspection or backtracking.

Differently from both planners above, parcPlan [Lever and Richards, 1994] is a temporal planner, rather than a HTN planner, that uses the constraint posting technique to mainly achieve temporal reasoning. For that, time-points are represented by finite-domain variables, allowing parcPLAN to utilise finite-domain constraint propagation techniques.

In parcPlan, constraints are provided in two different types to represent a problem domain: action constraints and integrity constraints. Action constraints define relations that must hold between time points of actions. Integrity constraints are classified as temporal and non-temporal. Temporal integrity constraints place bounds on the durations of properties and actions, or prohibit certain temporal configurations of properties and/or actions. Non-temporal integrity constraints express other aspects of the problem domain, such as quantitative constraints on resources. Despite the fact that parcPlan uses a discrete rather than a continuous model of time (what means, for example, that there is a maximum time period that a plan can occupy), this planner demonstrates the easy use and expressiveness of constraints in dealing with temporal reasoning.

The Descartes [Joslin and Pollack, 1995] planner advances the idea of constraint posting by also postponing some of the decisions of actions choice. In Descartes, every planning decision is represented by a variable, with constraints on each variable representing criteria that must be satisfied by the corresponding decision. These constraints are managed by a general-purpose constraint engine, so that even postponed decisions play a role in reasoning about plans.

The constraint posting approach of Descartes is similar to the MOLGEN's approach. With MOLGEN, once a constraint is posted it serves to rule out impossible values for variables, and guides the selection of operators to instantiate as interacting steps are added to the plan. Descartes extending this approach to apply to all decisions, not just variable binding. Also, differently from MOLGEN, Descartes is not a HTN planner, being better applied to puzzle-like problems.

O-Plan [Tate et al., 1994] is a HTN planner whose principal feature is to provide a modular way of manipulating constraints. For that end, O-Plan uses a number of specialised *constraint managers* to work on a plan, all sharing a constraint representation that allows them to interact. In this way, resource utilisation is handled by one constraint manager, temporal reasoning by another, and so on. An interesting feature of this approach is that as better modules become available for specific kinds of reasoning, they can be incorporated into the system.

Differently from Descartes and MOLGEN, which use constraint propagation in an active way during future decisions, O-Plan maintains an agenda representing decisions yet to be made and the postponement of constraints does not affect the handling of such agenda items. In terms of temporal representation, O-Plan is similar to parcPlan, also using constraints to define time points for activities. Each O-Plan activity has distinct start and finish instants in the form of time windows. Then the constraint manager, associated with temporal reasoning, revises information contained in time windows so that the overall consistency of temporal constraints is checked.

From this discussion, we can resume the principal uses of constraint posting by planners in the following points:

- Constraint posting is relevant to least-commitment[1] planners as they typically operate by eliminating inconsistent choices without committing to particular values. Thus, this technique has been used to implement least-commitment on the selection of objects used in actions, aspects of resource and temporal reasoning;
- This approach is useful to represent interactions between subproblems, avoiding conflicts and guiding the selection of new steps;
- The planners extensively make use of constraints to represent models of time, resource and ordering. This is because its easy use and power of expressing relations between domain variables.

Note that the use of constraint posting is not restricted to HTN planners. We are using this technique together with HTN approach such as O-Plan, MOLGEN and MACBeth have been using. However Descartes and parcPlan are non-HTN planners and also make use of the constraint posting facilities.

3.4 Requirements for Planning in Coalitions

During the last section, we have introduced constraint posting as an approach for planning performance. However, there are different ways that we can use constraints to describe a plan model, depending on the requirements of a particular set of problems that we want to handle. In this context, this section discusses the likely requirements for planning, considering its application in coalition scenarios. The focus is on two principal models (temporal and resource models) that, together, account for supporting several processes associated with planning such as conflict resolution, task allocation and load balancing.

3.4.1 Temporal Model

The definition of temporal models for planning agents can follow two main approaches [Narayek, 2001]: sequence-oriented and explicit timeline. The temporal references

[1]A strategy exemplifies least-commitment if, when faced with a number of alternative choices, it carries forward all possibilities rather than making a (backtrackable) commitment.

for these approaches are, respectively, time points (instants) at which a state variable changes its value, and time periods (intervals) during which a proposition holds. While an instant is represented as a variable raging over the set $\Re$ of real numbers, an interval is a pair (x,y) of reals, where x $\leq$ y.

Figure 3.1 illustrates a disaster scenario where we can apply the two approaches to represent the temporal notion of a coalition operation. The coalition $<\mu_0, [\mu_1, \mu_2, \mu_3]>$ has the goal of extinguishing two fires F_1 and F_2. μ_0 represents the command centre that must coordinate the activities of two fire brigades (μ_1 and μ_2) and one police force (μ_3). The fires are localised in different positions and the roads (R_1 and R_2) to access such fires are blocked. The figure also describes a possible total order plan for this scenario, which respects the temporal constraints showed in the figure as a graph.

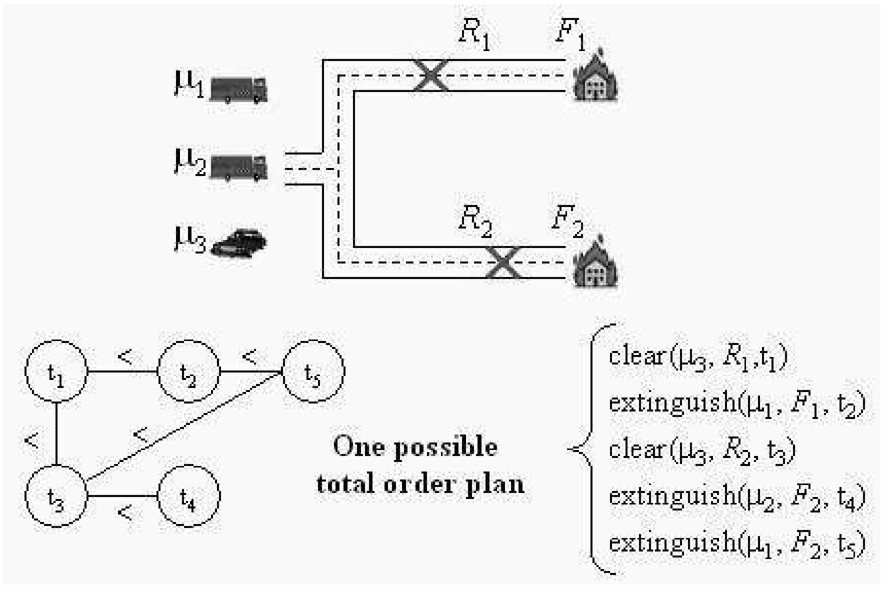

R_1
F_1
μ_1
μ_2
μ_3
R_2
F_2
t_1
t_2
t_3
t_4
t_5
<
One possible total order plan
clear(μ_3, R_1, t_1)
extinguish(μ_1, F_1, t_2)
clear(μ_3, R_2, t_3)
extinguish(μ_2, F_2, t_4)
extinguish(μ_1, F_2, t_5)

Figure 3.1: Example of disaster operation scenario.

The general expression *action*(μ_i,*obj*,t) means that the agent μ_i must perform the action *action* on the object *obj* at the instant *t*. In a similar way, such expression could be redefined as *action*(μ_i,*obj*,$[t_i,t_f]$), which means that μ_i must perform *action* starting at t_i and finishing at t_f. Using such notations, temporal relations between these temporal references can be expressed as binary constraints between instants (t) in a sequence-oriented approach, or between intervals ($[t_i,t_f]$) in an explicit timeline approach. A graphical representation for instants (Figure 3.2a) and intervals (Figure

3.2b) is given below. Note that this is only a possible option in time for such instants and intervals. Several other combinations could be possible to solve such problem.

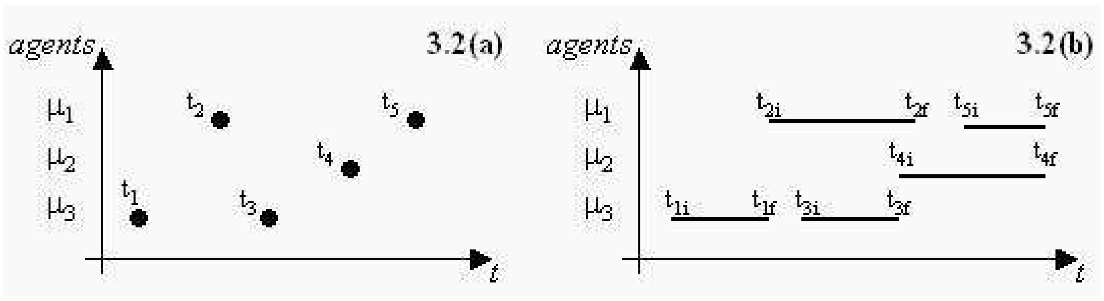


Figure 3.2: Graphical representation to instants and intervals.

The STRIPS framework [Fikes and Nilsson, 71] is the most familiar example of sequence oriented use. STRIPS considers actions as instantaneous state changers at a specific instant t, so that each action creates a different world state. If we consider that actions are labelled by temporal references, we can use constraints only to express precedence (e.g., $t_3 < t_4$) of time points between such actions. Based on this brief description, we can list some limitations of this approach for our disaster scenario:

- As actions are instantaneous, there is no provision for asserting what is true while an action is in execution. For example, during the action of extinguishing a fire, the water quantity of μ_2 is continually decreasing. At some time point it will finish so that the action becomes impossible;
- It is not possible to define more complex relations between actions. For example, to express a situation where μ_1 helps μ_2 during part of its task;
- Since state descriptions do not include information about action occurrences, such approach cannot represent the situation where one action occurs while some other event or action is occurring (simultaneous actions). For example, a situation where μ_1 is extinguishing F_1 while μ_3 is clearing R_2;
- Finally, it is also not possible to represent interactions of actions with external events beyond the agent's direct control. For example, if μ_2 wants to synchronise its extinguish action with a period of rain, which could facilitate its task.

Differently, a conceptual way to represent the explicit timeline idea via constraints, as used in [Allen, 1991], is to represent time intervals as variables and relations between two variables as constraints. In this way, many different temporal constraints can be defined:

- *In(*$[a_i, a_f], [b_i, b_f]$*)* - interval *a* is contained in interval *b*;
- *Disjoint(*$[a_i, a_f], [b_i, b_f]$*)* - *a* and *b* do not overlap in any way;
- *Starts(*$[a_i, a_f], [b_i, b_f]$*)* - interval *a* is an initial subsegment of *b*;
- *Finishes(*$[a_i, a_f], [b_i, b_f]$*)* - *a* is a final subsegment of *b*;
- *SameEnd(*$[a_i, a_f], [b_i, b_f]$*)* - intervals *a* and *b* end at same time;
- *Overlaps(*$[a_i, a_f], [b_i, b_f]$*)* - *a* starts before but overlaps *b*.

Approaches in this direction are more suitable to deal with scenarios as exemplified in Figure 3.1. For example, each interval has initial and final instants so that other actions and world state changes can be represented during such interval. In addition, as shown above, more complex relations can be defined between the intervals.

Note that in both examples, where constraints are used to define precedences on instants or to define more complex relations between intervals, such constraints present a *qualitative* feature. Qualitative models of time do not given metrical information about duration or about absolute time positions. Thus, they are useful to synchronise actions of unknown duration (e.g., the time for extinguishing a fire), or to support levels of abstraction in plans, avoiding that agents take decisions about things that they do not care about at the moment (least commitment principle). Using qualitative time, for example, the command centre μ_0 can define that μ_1 will extinguish F_1 after μ_3 has informed it that R_1 is clear. Note that μ_1 does not need to define specific times for that.

However, a quantitative model of time (e.g., *clearRoad(*μ_3*,*R_1*,[10,40]* meaning that such activity spends 30 units of time) is also important to improve the quality of decisions. In processes of resource allocation, for example, this approach could increase the amount of concurrent activities, avoiding that agents wait long times to receive tasks. For instance, if μ_0 knows that μ_3 will probably spend 30 units of time to clear

R_2, μ_0 could allocate the task of extinguishing F_1 to μ_2 during 30 units of time. Note however, that agents do not have control on the duration of several activities (extinguish fire, find buried people, etc.) in disaster domains. Thus, values for activities' duration will generally be estimations that could be based, for example, on past experiences.

Based on this discussion, we can define the first planning requirement for coalition support systems:

> ***Requirement 1****: the temporal planning model must be based on an explicit timeline approach, which must enable the representation of both quantitative and qualitative temporal references as well as relations between them.*

The implementation of such requirement brings a considerable level of expressiveness to manipulation of time. From a single expression like [a,b], considering a and b temporal references, the unique restrictions are that a must be non-negative and b must be greater than or equal to a. In this last case, the interval becomes an instant. Usually the temporal references a and b are time points, however it is also possible to define them as intervals (e.g., [[a_i,a_f],[b_i,b_f]]) so that we can express situations like "the activity starts between a_i and a_f and finishes between b_i and b_f. Temporal references can be absolute values, or variables to be completed in further steps. This could be important to represent lack in temporal knowledge of activities. Constraints can define relations between intervals or temporal references of these intervals. Such constraints are likely to be numerical relations, however logical constraints (e.g., meet, overlap, etc.) can also be applied. A more detailed survey on temporal constraints can be seen in [Schwalb and Vila, 1998].

3.4.2 Resource Model

Resources are objects whose cost or available quantity induces some restriction on the operations that use them. Examples of resources in our disaster relief scenario are fire brigades, ambulances, police forces and so on. Note that each resource also has its own resources. For instance, fire brigades must manage resources such as water and fuel during the performance of a given task.

In the context of planning with resources, a solution plan is defined as a plan that achieves the goals while allocating resources, which often have a limited availability, to operations in such a way that all resource constraints are satisfied. In order, constraints associated with resources are complementary to those associated with time, creating, together, the framework that accounts for leading the process of deciding *when* and *how* to do an activity [Ghallab et al., 2004].

In general, resources can be classified into two types: *reusable* and *consumable*. A resource is reusable if it is released unchanged after use. In a different way, consumable resources are consumed by activities during their use and, in general, they can be produced by other activities (e.g., refilling the amount of water in the fire engine's tank).

In our disaster scenario (Figure 3.1), the resources are the two fire brigades μ_1 and μ_2, and the police force μ_3. During the planning process of μ_0, such resources can be used as reusable resources so that when μ_1 has finished to extinguish F_1, for example, it can be automatically sent to another mission. Differently, during its own planning process, μ_1 must consider that it is a consumable resource so that it needs to allocate time for refilling both water and fuel tanks between or during its tasks. Figure 3.3 illustrates this idea.

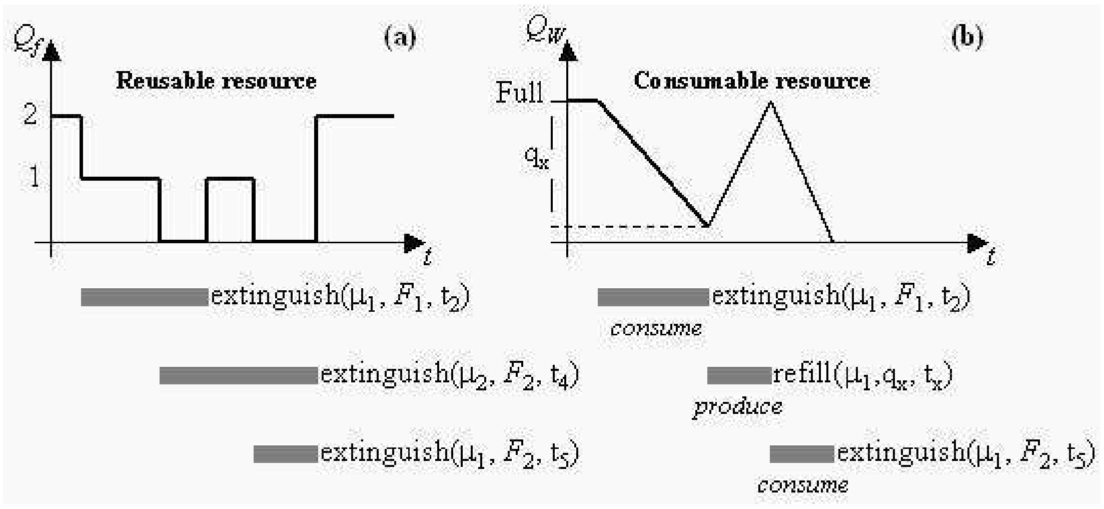


Figure 3.3: Example of reusable (a) and consumable (b) resources.

Figure 3.3a illustrates the task allocation of fire brigades in the μ_0's plan. In this case, resources are represented by fire brigades (μ_1 and μ_2) and Q_f represents the quantity of fire brigades. Note that when a fire brigade finishes an activity, it is automatically available to new tasks. Figure 3.3b focuses on a consumable resource (water) of μ_1. Here Q_w means the quantity of water available. In this case μ_1 must distinguish times for consuming and producing such resource, considering both times in its plan to extinguish fires.

The specification of each activity could bring, together with the usual preconditions and effects, the requirements that specify which resources and what quantities of these resources are needed for its performance. Considering a coalition (multiagent) scenario, the resource requirements of an activity will also guide the process of finding subordinate agents that are able to perform such activity. Normally, superior agents may have the choice between several alternative resources (agents), and a resource also may be shared between several activities. For example, if μ_0's plan has an activity to extinguish a fire in a building of 9 levels, it must allocate this task to a fire brigade that has a suitable ladder for that activity. If both μ_1 and μ_2 have the necessary ladder, μ_0 can use other variables to choose one of them.

Considering this scenario, superior agents need to be able to access an appropriate description of its subordinate agents. In a more complex use of these descriptions, superior agents could also compose new capabilities through the interoperation and union of the capabilities of other agents. From this discussion, we can elaborate a second requirement for planning in coalition support systems:

> ***Requirement 2****: the resource planning model must support the tasks of localising services/agents that provide specified capabilities, and also provide information that enables reasoning on such capabilities.*

Based on this requirement, the representation of agents could be provided at two levels. The first level providing a high-level description of agents, informing what they are able to do. The second could bring a more in depth description of the agents' capabilities, such as their functional behaviour and attributes (response time, accuracy, information required for its activation, etc.).

3.5 Summary

This section has introduced the study of planning in multiagent scenarios, considering a hierarchical organisation for their agents. We have justified the use of the HTN framework as the natural choice to implement planning in hierarchies, together with the constraint posting technique to augment it. The principal point was to show that the HTN and constraint posting approaches have complementary features, which can improve the performance of planning.

Based on these ideas, we have defined two important requirements associated with planning design in coalition support systems. First, the temporal model must be based on an explicit timeline approach, which must enable the representation of both quantitative and qualitative temporal references as well as relations between them. Second, the resource model must support the tasks of localising agents that provide specified capabilities, also providing information that enables reasoning on such capabilities. Together such models are able to support several planning mechanisms, such as the process of finding out how to perform a given set of activities using a limited number of resources in a limited amount of time.

Chapter 4

Teamwork Theories for Collaborative Planning

Teamwork research defines several important concepts regarding the design of agents so that they are in fact able to play as members of a collaborative team. The principal idea is that the team's joint activities do not consist merely of coordinated individual actions, but each member needs, for example, to make commitments on reporting status of their ongoing activities and support the activities of other members.

This chapter starts by discussing, in Section 4.1, why the traditional multiagent planning framework is not sufficient to ensure collaboration between agents. Then, Section 4.2 discusses four principal teamwork proposals: *Joint Intentions*, *Shared-Plan*, *Joint Responsibilities* and *Planned Team Activities*. The discussion is directed by the influences that a hierarchical organisation can have on the teamwork concepts. Finally, Section 4.3 traces a comparative analysis of such approaches, highlighting the principal teamwork requirements, while Section 4.4 summarises the important ideas of this chapter.

4.1 Why is planning not enough?

The complexity of problems associated with coalitions, such as disaster relief operations or military missions, requires that the planning and execution activities of coalition members be performed in a collaborative way. For that, such 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 [Levesque et al., 1990].

Planning approaches, such as described in the last chapter, support the operations of distributing tasks and synchronising their performances. Thus, there is a coordination of activities so that conflicts and redundant tasks can be avoided. However, these operations still do not ensure collaborative behaviour. The two cases below illustrate the problems that can appear when teams only use traditional planning approaches.

4.1.1 Case I: The Helicopters Attack Domain

The helicopters attack domain, which is completely described in [Tambe, 1997a], involves pilot agents for a coalition of synthetic helicopters. The plan consists in attacking enemy vehicles and coming back to the home-base. For that end, the helicopters must fly to a holding point, where one or two *scout helicopters* will fly forward and first scout the battle position. Based on communication from the scouts, other coalition members fly forward to the battle position. Here, each pilot repeatedly masks/hides their helicopters and unmasks to shoot missiles at enemy targets. Once the attack has been completed, they return to their home-base.

For this experiment, each pilot has an operator hierarchy, which is very similar to reactive plans. Each operator consists of precondition rules for selection, rules for application and rules for termination. At any moment, only one path through this hierarchy is active. The principal failures associated with this experiment are enumerated below [Tambe, 1997a]:

1. Upon abnormally terminating engagement with the enemy, the coalition commander returned to home-base alone, abandoning members of its own coalition at the battle position;

2. Upon reaching the holding area, the coalition waited, while a single scout started flying forward. However, the scout unexpectedly crashed into a hillside, so that the rest of the coalition just waited indefinitely for the scout's scouting message;

3. Only a scout made it to the holding area, while all other helicopters crashed or got shot down. However the scout scouted the battle position anyhow and waited indefinitely for its non-existent company to move forward;

4. While evading an enemy vehicle encountered enroute, one helicopter pilot agent unexpectedly destroyed the vehicle via gunfire. However it did not inform others, so that an unnecessarily circuitous bypass route was planned.

To address these and other failures, a possible approach was to add domain-specific coordination plans, enabling, for example, that after scouting the battle position, a scout executes a plan to inform that the battle position is scouted to those waiting in the holding position. However, several difficulties were associated with this approach. First, there is not a way to anticipate failures, so that coordination plans have to be added on a case-by-case basis. Second, as the system continues to scale up to increasingly complex scenarios, such failures continually recur, so that a large number of special case coordination plans are potentially necessary. Finally, it is difficult to reuse such plans in other domains.

4.1.2 Case II: The Guararapes Game

Guararapes Battle [Siebra and Ramalho, 1999, Silva et al., 2001] is a strategic computer game, partially implemented during my MSc project, where four different ethnic groups dispute the domain of a region. For that end, the groups can simply fight between themselves, or try more complex negotiation processes to form alliances, define periods of peace, or exchange resources. Each group has a coordinator agent, which can be a player or another autonomous agent. The reasoning of each agent was implemented as a production system so that a set of configurable rules account for guiding its behaviour.

During some experiments, *Guararapes* also presented several problems, similar to the helicopters attack domain. Some of them are listed below:

1. During a negotiation, two members of different groups agreed in setting up a temporary alliance. However this alliance is not positive for one of the groups;

2. During a fight between two groups A and B, some members of A started to run away from the battle field, leaving the others to fight alone. Even seeing this behaviour, the members of A that are still fighting are not able to infer what to do because they do not know what this behaviour means;

3. When a member discovers a hidden enemy position, it does not share this information with the other members;

All these behaviours are not in accordance with the collaborative idea that coalitions should present. The solution was to add specific rules to deal with each case. For example, a rule to force agents to inform others that they are leaving the battle field. In this way, it is not difficult to see that this project suffered from the same limitations discussed in the previous case (section 4.1.1).

4.2 Principal Teamwork Proposals

The cases discussed in the last section stress the notion that *collaboration between different problem-solving components must be designed into systems from the start. It cannot be patched on.* [Grosz, 1996]. In this way, coalition planning processes need to be designed on a collaborative framework that ensures commitments of individual components in carrying out their activities, considering the global coalition objective.

Teamwork research [Cohen and Levesque, 1991] encloses a set of ideas that support the implementation of this collaborative framework. In fact, teamwork has become the most widely accepted metaphor for describing the nature of multiagent collaboration [Sierhuis et al., 2003]. The works discussed in this section are the principal proposals to implement the teamwork concepts and, although they present different approaches to deal with specific technical problems, they agree on the fundamental teamwork concepts, as discussed later.

4.2.1 Joint Intentions

The *Joint Intention Theory* [Levesque et al., 1990] specifies how a group of agents can cooperatively act together by sharing certain mental states about joint activities.

According to this theory, a team will properly perform joint activities if its agents have established commitments on the performance of such activities, while they also mutually believe that the team will do that.

Based on this idea, the focus of Joint Intentions theory is on how to manage the potential divergences about the mental states of members, allowing them to arrive at private beliefs about the status of shared activities. Such divergences can appear because agents are situated in real and dynamic environments where external events are probable causes of changes in goals or failures in actions. Thus current commitments and beliefs can be changed as time passes, causing problems for the performance of activities. For example, a police force concludes that it is not able to clear a road, abandoning such activity. Consequently the plans of other agents that make use of this road will become invalid, but such agents do not know this information (they believe that the road will be cleared) and will spend time and resources executing their plan until discovering by themselves this new fact.

To deal with problems like this, the Joint Intentions theory proposes a set of principles to lead the behaviour of agents that intend to perform joint activities. Such principles can be described on three levels. First, *Weak Goals* (WG) that specify the conditions under which an agent holds a goal, and the actions it must take if the goal is satisfied, irrelevant or impossible. Second, *Joint Persistent Goals* (JPG) that specify a joint commitment of a team in achieving a goal. Finally, *Joint Intentions* (JI) that are defined in terms of WG and JPG. Using the predicates[1] BEL(μ,p), MBEL(Θ,p) and INT(μ,p), and assuming that e is an extra condition to represent the reasons that μ (member of a coalition Θ) may have for keeping the goal, WG can be defined as [2]:

$$\begin{aligned}\mathrm{WG}(\mu,\Theta,p,e) \equiv\ & [\mathrm{BEL}(\mu,\neg p) \wedge \mathrm{INT}(\mu,\Diamond p)] \\ & \vee [\mathrm{BEL}(\mu,p) \wedge \mathrm{INT}(\mu,\Diamond \mathrm{MBEL}(\Theta,p))] \\ & \vee [\mathrm{BEL}(\mu,\Box\neg p) \wedge \mathrm{INT}(\mu,\Diamond \mathrm{MBEL}(\Theta,\Box\neg p))] \\ & \vee [\mathrm{BEL}(\mu,\neg e) \wedge \mathrm{INT}(\mu,\Diamond \mathrm{MBEL}(\Theta,\neg p))]\end{aligned}$$

[1]BEL(μ,p) and INT(μ,p) are concepts based on the BDI (*Belief-Desire-Intention*) Theory [Rao and Georgeff, 1995] and they mean, respectively, that agent μ believes proposition p to be true and μ intends to perform p. MBEL(Θ,p) means that all agents in Θ mutually believe that p holds.

[2]$\Diamond p$ is a modal operator that requires p to be true at some point in the future (eventually), while $\Box p$ requires that p to be true from now on (always).

According to this definition, if μ has a WG to carry out p while e is not irrelevant (e is not false), one of the following conditions will hold:

- μ believes that p is not true and intends that p be true at some future time;
- Having μ privately discover p to be true, μ intends that all the coalition Θ mutually believes that p is true;
- Having μ privately discover p to be unachievable, μ intends that all Θ mutually believes that p is unachievable.
- Having μ privately discover p to be irrelevant because e has became false, μ intends that all Θ mutually believes that p is (temporally) unachievable.

In practical terms, if μ has a WG to carry out p, μ has a commitment to perform p (or try at least), or to communicate its completion, failure or irrelevance to Θ. e specifies the conditions under which agents may drop their intentions while still believing that p is untrue and satisfiable. With this definition in place, we can define Joint Persistent Goals as follows:

$$\begin{aligned}\mathrm{JPG}(\Theta,p,e) \equiv\ & \mathrm{MBEL}(\Theta,\neg p)\\ & \wedge\ \mathrm{MBEL}(\Theta,\mathrm{MINT}(\Theta,p))\\ & \wedge\ \mathrm{UNTIL}[\mathrm{MBEL}(\Theta,p) \vee \mathrm{MBEL}(\Theta,\Box\neg p) \vee \mathrm{MBEL}(\Theta,\neg e),\\ & \quad \forall\mu \in \Theta\ \mathrm{WG}(\mu,\Theta,p,e)]\end{aligned}$$

According to definition, a JPG(Θ,p,e) holds if all the following conditions are satisfied:

- Θ mutually believes that p is currently false;
- Θ mutually believes that Θ holds p as a goal (Θ mutually intend to do p);
- Each member μ of Θ holds p as a WG until Θ mutually believes that p is true, unachievable or irrelevant (predicate UNTIL says that its second argument will be true as long as the first holds.).

A JPG ensures that coalition members cannot decommit until p is mutually believed to be achieved, impossible or irrelevant. This commitment also ensures that members stay updated about the status of team activities, and thus do not unnecessarily face risks or waste their time. Finally, using JPG we can define Joint Intentions (JI) as[3]:

$$\mathrm{JI}(\Theta,a,e) \equiv \mathrm{JPG}(\Theta,\mathrm{DONE}(\Theta,\mathrm{UNTIL}[\mathrm{DONE}(\Theta,a),\mathrm{MBEL}(\Theta,\mathrm{DOING}(\Theta,a))]?;a),e)$$

The notation $x?;y$ means: "perform activity y, x being true initially. So the equation above states that Θ has a joint intention to do activity a if their agents have a joint persistent goal to achieve a, and mutually believe that they are doing a until the time that they have done a. The JPG (and thus the JI) may be abandoned at any time if e becomes false. By using such ideas in our example, if the police force is not able to carry out the activity of clearing the road, it cannot simply abandon the activity because it has a commitment in a JPG. According to such commitment, it will need to communicate its failure, keeping the status of the coalition updated.

If agents are arranged in a hierarchical organisation, as discussed in Section 2.4, the coalition will not only stabilise one JPG (commitment). In this case, the number of JPGs corresponds to the number of interaction zones Φ_i involved in the performance of p. For example, considering the coalition Θ in Figure 4.1.

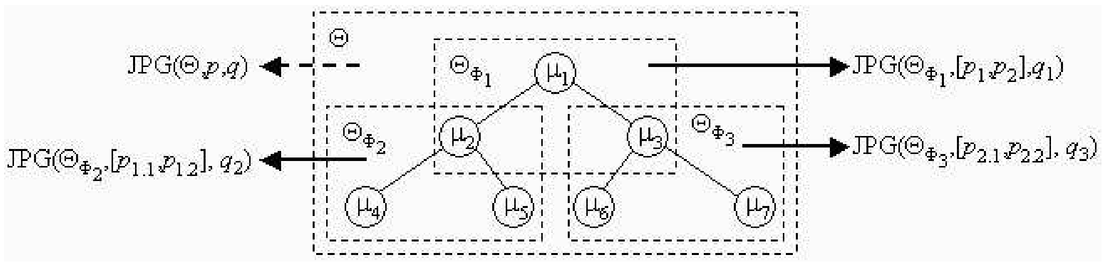

$\mathrm{JPG}(\Theta,p,q)$
Θ
Θ_{Φ_1}
μ_1
$\mathrm{JPG}(\Theta_{\Phi_1},[p_1,p_2],q_1)$
$\mathrm{JPG}(\Theta_{\Phi_2},[p_{1.1},p_{1.2}], q_2)$
Θ_{Φ_2}
μ_2
μ_3
Θ_{Φ_3}
$\mathrm{JPG}(\Theta_{\Phi_3},[p_{2.1},p_{2.2}], q_3)$
μ_4
μ_5
μ_6
μ_7

Figure 4.1: JPG commitments in a hierarchical organisation.

The coalition Θ can be decomposed into three interaction zones: Φ_1, Φ_2 and Φ_3. Sub-coalitions (Θ_{Φ_1}, Θ_{Φ_2} and Θ_{Φ_3}) in each zone makes a commitment on parts of p and the commitment JPG(Θ,p,e) indirectly emerges from the sub-coalitions' commitments. If Θ_{Φ_2}, for example, makes a commitment on the performance of p_1 ($p_1 = p_{1.1} \cup p_{1.2}$),

[3]DOING(μ/Θ,a) indicates future execution (starting immediately) of activity a for an agent μ or a coalition Θ, while DONE(μ/Θ,a) indicates immediate past execution of a for μ or Θ.

which is part of p, the following sentences will hold in Θ_{Φ_2}: WG(μ_2,Θ_{Φ_2},[$p_{1.1}$,$p_{1.2}$],e), WG(μ_4,μ_2,$p_{1.1}$,e) and WG(μ_5,μ_2,$p_{1.2}$,e). Furthermore μ_2 will also make a WG with μ_1, specified as: WG(μ_2,μ_1,p_1,e). It is important to note that such approach follows the principles of enclosing the problems into sub-coalitions and decreasing the communication among agents, because problems are only reported to superior agents.

Joint Intentions theory was mainly used in a practical way via the *STEAM Project* [Tambe, 1997b]. STEAM (Shell for TEAMwork) enables explicit representation of team goals and plans, and teams' joint commitments. The communication is driven by such commitments, so that team agents may interact to attain mutual belief while building and disbanding joint intentions. STEAM was applied in three different domains. First to provide collaboration among helicopters during an attack mission (see Section 4.1.1). Second in a transport domain, also using synthetic helicopters as agents. Finally in the RoboCup synthetic soccer domain, where agents play the role of football players. A summary of the results for these three domains can be seen in [Tambe, 1997a].

4.2.2 SharedPlans

In a similar fashion way to Joint intentions, *SharedPlans Theory* [Grosz et al., 1999] argues that, for efficient collaboration, each agent of a team needs to have mutual beliefs about the goals and actions to be performed and the capabilities, intentions and commitments of the other participants. In addition, SharedPlans also considers mechanisms that enable agents to interleave planning and acting, avoiding the adoption of conflicting intentions and keeping the focus on the information that they actually need for their activities.

According to SharedPlans, if an agent μ (where $\mu \in \Theta$) intends to do a basic activity a, this requires that μ believes that it is able to execute a and it is committed to doing so:

$$\text{BASIC}(a) \wedge \text{INT}(\mu,a) \Rightarrow \text{BEL}(\mu,(a \in \text{CAPABILITY}(\mu))) \wedge \text{WG}(\mu,\Theta,a,e)$$

Note that we are using WG to represent the individual commitment of μ in realising a. If a is complex, then μ intending to do a requires either that μ has a *Full Individual*

Plan (FIP)[4] for carrying out a or a *Partial Individual Plan* (PIP) for carrying out a together with a plan FIP^{Elab} to elaborate the partial plan into a full plan:

$$\text{COMPLEX}(a) \wedge \text{INT}(\mu,a) \Rightarrow \text{FIP} \vee (\text{PIP} \wedge \text{FIP}^{Elab})$$

Thus, using a recursive definition to create the hierarchy of activities, μ has a FIP for performing a if: μ has a complete plan for performing a, μ intends to do each activity in that plan and μ has a subordinate FIP to do each complex activity in that plan. The same idea is used to define this process for teams of agents. A team Θ has a SharedPlan (SP) to do some multiagent activity a either by having a *Full SharedPlan* (FSP) to carry out a or by having a *Partial SharedPlan* (PSP) to carry out a together with a plan FSP^{Elab} to elaborate the partial plan into a full plan:

$$\text{SP} \Rightarrow \text{FSP} \vee (\text{PSP} \wedge \text{FSP}^{Elab})$$

A SP is reducible to the individual plans, beliefs and intentions of the various team agents, and a single-agent activity a_i in the decomposition hierarchy is called resolved if: an agent μ_i has been selected to do a_i, μ_i intends to do a_i, and the other members of the group have a set of supportive mutual beliefs and intentions-that (INT.TH)[5] a_i succeed. Similarly a multiagent activity a_j, which is assumed to be ultimately decomposable into a basic or complex single-agent activity, is called resolved if: a subteam Θ_j has been selected to do a_j, Θ_j has a SP to do a_j, and the other agents of the group have a set of supportive mutual beliefs and intentions-that Θ_j succeed.

While a FSP is characterised by a complete activity decomposition hierarchy where each activity has been fully resolved, a PSP is characterised by a possible incomplete activity decomposition hierarchy where some or all activities may be unresolved. In this way, the use of PSPs enables that agents interleave planning and execution because the team and subteams are not obligated to have a complete idea of their activities. This aspect could be important because agents do not, in some cases, begin a collaboration with all of the conditions (beliefs, commitments, etc.) in place. In such cases, agents normally start with only partial knowledge about the environment and other

[4]The complete definitions of FIP, PIP, FSP and PSP can be found in [Grosz et al., 1999].

[5]INT.TH(μ,p) and INT.TH(Θ,p) are similar to the notions of DES(μ,p) and MDES(Θ,p) respectively, however motivating possible supportive actions.

participants, and use specific mechanisms (e.g., communication) to gather information during their operations.

An important aspect of this theory is that during the process of decomposition, team members are only required to know that a plan exists to enable one or more teammates to perform activities, but not the details of such plan. Only the agents selected to do a given subactivity need to be directly involved in the corresponding plan for it.

The proposal of SharedPlans to avoid conflicts is also based on the recursive process of decomposition. Considering that agents have established a mutual belief that there is a plan to solve a problem, the recursive process generates at each step a new set of activities on which agents also need to make agreements. Furthermore via intentional attitudes (INT.TH), an agent is able to say to others which propositions need to hold so that its activities can be performed. Thus, such attitudes of an agent directly constraint the intentions that other agents adopt, affecting their plan-based reasoning.

We can note that the decomposition process can provide by itself a way to create a hierarchical organisation. For example, considering the case where the coalition Θ has a FSP to do a multiagent activity a, we have the following implications:

1. Θ mutually believes that its members intend-that Θ performs a;
2. Θ mutually believes that it has a full plan to perform a;
3. Each single-agent activity a_i or multiagent activity a_j of the plan has respectively some agent μ_i in Θ that intends to perform a_i, or some sub-coalition Θ_j in Θ that has a FSP to perform a_j;
4. Θ mutually believes either that μ_i intends to perform a_i, or Θ_j has a FSP to perform a_j; and that both are able to perform their respective activities;
5. Θ mutually believes that each member intends-that μ_i or Θ_j be able to perform a_i or a_j respectively.

According to such implications, if $\Theta = <\mu_1, [<\mu_2, [\Theta_x, \mu_5]>, <\mu_3, [\mu_6, \mu_7]>]>$[6] intends to perform a, a hierarchy could be defined by the following sentences:

[6]From this coalition we can infer $\Theta_{\Phi_1} = <\mu_1, [\mu_2, \mu_3]>, \Theta_{\Phi_2} = <\mu_2, [\mu_s, \mu_5]>$ and $\Theta_{\Phi_3} = <\mu_3, [\mu_6, \mu_7]>$, where μ_s is the superior member of Θ_x.

- MBEL(Θ_{Φ_1},INT.TH(Θ_{Φ_1},a)) $\wedge$ MBEL(Θ_{Φ_1},PLAN(Θ_{Φ_1},a)), from implications 1 and 2;

- $\exists$(p_1=PLAN(Θ_{Φ_2},a_1)) $\wedge$ $\exists$(p_2=PLAN(Θ_{Φ_3},a_2)) from implication 3, where a_1 and a_2 are multiagent activities of p. For p_1 we have (note that p_2 can be decomposed in a similar way):

  - MBEL(Θ_{Φ_2},INT.TH(Θ_{Φ_2},a_1)) $\wedge$ MBEL(Θ_{Φ_2},PLAN(Θ_{Φ_2},a_1)), from implications 1 and 2;
  - $\exists$PLAN(Θ_x,a_3) $\wedge$ INT(μ_5,a_4) from implication 3, where a_3 is a multiagent and a_4 is a single-agent activity of p_1;
  - MBEL(Θ_{Φ_2},PLAN(Θ_x,a_3)) $\wedge$ MBEL(Θ_{Φ_2},INT(μ_5,a_4)) from implication 4;
  - MBEL(Θ_{Φ_2},INT.TH(Θ_{Φ_2},$a_3 \in$ CAPABILITY(Θ_x)) $\wedge$ MBEL(Θ_{Φ_2},INT.TH(Θ_{Φ_2},$a_4 \in$ CAPABILITY(μ_5)) from implication 5.

- MBEL(Θ_{Φ_1},PLAN(Θ_{Φ_2},a_1)) $\wedge$ MBEL(Θ_{Φ_1},PLAN(Θ_{Φ_3},a_2)) from implication 4;

- MBEL(Θ_{Φ_1},INT.TH(Θ_{Φ_1},$a_1 \in$ CAPABILITY(Θ_{Φ_2})) $\wedge$ MBEL(Θ_{Φ_1},INT.TH(Θ_{Φ_1}, $a_2 \in$ CAPABILITY(Θ_{Φ_3})) from implication 5.

A visual result of these sentences can be seen in Figure 4.2. Following this process of decomposition, we can successively create several levels of abstractions until all activities become single-agent activities.

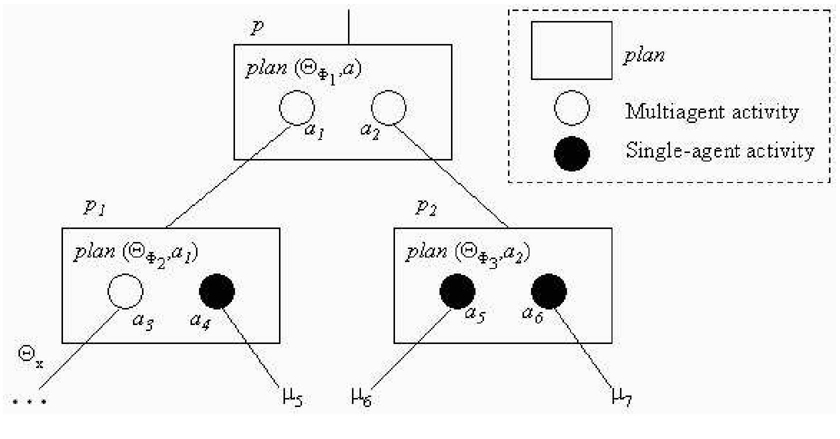


Figure 4.2: Result of a hierarchy, described via SharedPlans.

Implementations of SharedPlans include a collaborative interface agent for an air travel application called COLLAGEN [Rich and Sidner, 1997] and a collaborative multiagent system for electronic commerce [Hadad, 1997]. COLLAGEN uses SharedPlans to represent decisions, which have been made as a result of actions and utterances during a dialogue, as plan trees. Nodes (plans) in a specific tree represent mutually agreed decisions upon intentions (e.g., to perform a task), and the tree structure represents the subgoal relationships among these intentions. In the electronic commerce scenario, SharedPlans may be formed between agents belonging to the same enterprise that aim to work together, maximising their enterprise's benefits; or also among agents that are self-motivated and interested in collaboration because they may improve their individual benefits.

4.2.3 Joint Responsibilities

The *Joint Responsibilities* theory [Jennings, 1992] extends the Joint Intentions ideas to include the notion of plan states. According to this theory, an important reason for explicitly distinguishing between goals and plan states becomes evident by examining what happens after the two types of commitment failure. In the former case, the team's activity with respect to the particular goal is over. However if the group becomes uncommitted to the common solution (a plan) there may still be useful processing to be carried out. For example, if the plan is deemed invalid, the agents may try a different sequence of actions which produce the same result. Thus dropping commitment to the common solution plays a different functional role than dropping a goal.

The concept of common solution is a way of dealing with interdependence of activities. If subproblems are solvable in isolation, it may be impossible to synthesise their results because the solutions are incompatible or because they violate global constraints. In order the Joint Responsibilities theory argues that a common goal is not sufficient to guarantee that collaborative problem solving will ensue. Team members also need to agree upon a common solution for achieving their goals. However this stipulation does not imply that the common solution must be developed before joint actions can commence nor that it cannot evolve over time. Rather it reflects the fact that team members must believe that eventually they will be able to agree upon, and work under, a common solution with respect to their shared objective.

The Joint Responsibilities formalism starts by defining the notion of relationships between activities of common solutions, which are represented by plans. Considering B the set of basic activities and M the set of multiagent activities, if $[b_1,b_2] \in B$, $[m_1,m_2] \in M$ and $\Re_{x,y}(x,y)$ represents the relationship between activities x and y; we could have relationships such as $\Re_{m_1,m_2}(m_1,m_2)$, $\Re_{m_1,b_2}(m_1,b_2)$ or $\Re_{m_1,b_1,b_2}(m_1,b_1,b_2)$. Otherwise If two activities are independent we have: $\neg\Re_{x,y}(x,y)$.

Activities can be combined into finite sequences Σ to specify more complex interactions. Sequences are composed of at least one activity and may contain a mix of basic/multiagent and related/independent activities. Basic activities are assumed to be solved by activity sequences of length one (e.g. p_1 is solved by $\Sigma = \{b_1\}$). The representation Σ_σ means that the sequence Σ is executed in order to fulfil the goal σ.

The theory distinguishes activities to be carried out from agents who will execute them. This enables the planning mechanisms to be independent of task and resource allocation considerations. Once the activity sequence has been defined, the agents who will in fact perform it need to be decided upon. Considering Θ a coalition, the following instantiations need to be done:

- Basic activity instantiation: $\ll \mu, b \gg$, where agent $\mu \in \Theta$ is involved in the performance of the basic activity $b \in B$;
- Multiagent activity instantiation: $\ll \Theta_j, m \gg$, where $\Theta_j \subseteq \Theta$ is involved in the performance of the multiagent activity $m \in M$;
- Activity sequence instantiation: a sequence of basic and multiagent activity instantiations, which specifies the respective agents who will perform them. If Σ_σ is an activity sequence, its instantiation is denoted by Σ'_σ.

At this point we can define the predicate RELATION-OK to indicate that the relationship between two activities σ_i, $\sigma_j \in \Sigma'_\sigma$ is satisfied or there is not relation between them. Such definition is given as:

$$\begin{aligned}\text{RELATION-OK}(\ll \Theta_i,\sigma_i \gg, \ll \Theta_j,\sigma_j \gg, \Sigma'_\sigma) \equiv\ & \Re_{\sigma_i,\sigma_j} \\ & \vee (\neg\exists\, \Re_{\sigma_i,\sigma_j} \in \Sigma'_\sigma)\end{aligned}$$

Using such predicate and $c?;a$ to mean activity a with c holding initially, we can define the conditions to an agent μ and a sub-coalition Θ_i to perform a basic activity b and a multiagent activity m_i respectively:

$$\begin{aligned}
\text{PERFORM}(\ll \mu, b \gg, \Sigma'_\sigma) \equiv\ & (\forall \ll \{\mu_w..\mu_x\}, \sigma_i \gg\ \in \Sigma'_\sigma) \\
& \quad \text{RELATION-OK}(\ll \mu, b \gg, \ll \{\mu_w..\mu_x\}, \sigma_i \gg, \Sigma'_\sigma)?; \\
& \quad \text{DOING}(\ll \mu, b \gg) \\
\text{PERFORM}(\ll \Theta_i, m_i \gg, \Sigma'_m) \equiv\ & (\forall \ll \{\mu_w..\mu_x\}, m_j \gg\ \in \Sigma'_m) \\
& \quad \text{RELATION-OK}(\ll \Theta_i, m_i \gg, \ll \{\mu_w..\mu_x\}, m_j \gg, \Sigma'_m)?; \\
& \quad (\exists \Sigma'_{m_i}\ PERFORM(\ll \Theta_i, m_i \gg, \Sigma'_{m_i}))
\end{aligned}$$

The first definition simply means that all relationships involving b in Σ'_σ must be satisfied. The second is a recursive definition where Σ'_{m_i} is a solution developed by Θ_i for solving m_i. However, before dealing with a joint activity, members of a coalition must agree upon a common solution. This fact is specified as:

$$\text{NEED-COMMON-SOLUTION}(\ll \Theta_i, \sigma \gg)\ \equiv\ \Diamond \exists \Sigma'_\sigma\ PERFORM(\ll \Theta_i, \sigma \gg, \Sigma'_\sigma)$$

At this point a particular emphasis is given to define conditions under which it is rational to drop commitments to the agreed solution and the actions which must be taken in such circumstances. The failure conditions are:

- The motivation for performing one of the activities is not present (LACKING-MOTIVE),
- The agreed sequence does not achieve the desired outcome (INVALID),
- One of the specified activities cannot be performed (UNATTAINABLE);
- One of the agreed activities was not performed (VIOLATED).

These conditions represent situations in which a member μ in Θ can detect, for itself (local problem), that the common solution is no longer sustainable. Thus, it needs to reassess its commitments to the agreed solution Σ'_σ. Local problems are represented as:

$$\begin{aligned}
&\text{LOCAL-PROBLEM}(\mu, \ll \Theta, \sigma \gg, \Sigma'_\sigma) \equiv \\
&\qquad \text{BEL}(\mu, \text{LACKING-MOTIVE}(\ll \Theta, \sigma \gg, \Sigma'_\sigma) \vee \\
&\qquad\quad \text{INVALID}(\ll \Theta, \sigma \gg, \Sigma'_\sigma) \vee \\
&\qquad\quad \text{UNATTAINABLE}(\ll \Theta, \sigma \gg, \Sigma'_\sigma) \vee \\
&\qquad\quad \text{VIOLATED}(\ll \Theta, \sigma \gg, \Sigma'_\sigma)
\end{aligned}$$

Differently to a LOCAL-PROBLEM, a NON-LOCAL-PROBLEM appears if an agent realises that one of its fellow team agents has dropped commitments to the solution. In this case the agent needs to reassess its position to take this new information into account. NON-LOCAL-PROBLEM is represented as:

$$\begin{aligned}
&\text{NON-LOCAL-PROBLEM}(\mu, \ll \Theta, \sigma \gg, \Sigma'_\sigma) \equiv \\
&\quad \mu \neq \mu_i \; BEL(\mu, (\exists \mu_i \in \Theta \;\; \text{LOCAL-PROBLEM}(\mu_i, \ll \Theta, \sigma \gg, \Sigma'_\sigma)))
\end{aligned}$$

Thus, the situation under which an agent μ drops commitments to an agreed common solution Σ'_σ for a multiagent activity instantiation $\ll \Theta, \sigma \gg$ is given by the disjunction of both problems: DROP-SOL-COMMIT $\equiv$ LOCAL-PROBLEM $\vee$ NON-LOCAL-PROBLEM.

To ensure that the problem is disseminated within Θ, μ must inform all other members about the fact that it is no longer committed and also the reason why. In this way, the theory uses the concept of *Individual Solution Commitment* (ISC) to represent a high level description of how each agent should behave in its own planning performance and toward others with regard to the agreed solution[7]:

$$\begin{aligned}
&\text{ISC}(\mu, \ll \Theta, \sigma \gg, \Sigma'_\sigma) \equiv \\
&\quad WHILE \neg \text{DROP-SOL-COMMIT}(\mu, \ll \Theta, \sigma \gg, \Sigma'_\sigma) \; DO \\
&\qquad (\forall \ll \{\mu, \mu_w..\mu_x\}, \sigma_i \gg \in \Sigma'_\sigma \text{), where } \{\mu, \mu_w..\mu_x\} \subseteq \Theta \\
&\qquad\quad \text{BEL}(\mu, \Diamond PERFORM(\ll \{\mu, \mu_w..\mu_x\}, \sigma_i \gg, \Sigma'_{\sigma_i})) \wedge \\
&\qquad\quad \Diamond PERFORM(\ll \{\mu, \mu_w..\mu_x\}, \sigma_i \gg, \Sigma'_{\sigma_i})) \\
&\quad \text{WHEN} \\
&\qquad \text{GOAL}(\mu, MBEL(\{\Theta, \text{DROP-SOL-COMMIT}(\mu, \ll \Theta, \sigma \gg, \Sigma'_\sigma)))
\end{aligned}$$

[7]In the ISC definition we have WHILE p DO q WHEN r: while p is true, q will remain true. When p becomes false, q will be false and r will become true.

According to this definition, for each activity that μ is involved in, it should believe that it is going to perform that activity and also that it will in fact perform the activity at the appropriate time. If there is some problem to perform the activity, μ will have a new goal of disseminating this information within Θ. Then, combining the given definitions, we can specify the two conditions concerned with performing activities in a multiagent group: agreeing to a common solution and defining how individuals should behave once such a solution has been chosen:

$$\begin{aligned} &\text{SOL-COMMITMENT}(\ll \Theta, \sigma \gg) \equiv \\ &\qquad \text{MBEL}(\Theta, \text{NEED-COMMON-SOLUTION}(\ll \Theta, \sigma \gg)) \wedge \\ &\qquad \text{MBEL}(\Theta, (\forall \mu_i \in \Theta\ ISC(\mu_i, \ll \Theta, \sigma \gg, \Sigma'_\sigma))) \end{aligned}$$

Finally we can define the mental state of joint responsibility which a coalition Θ must adopt if its agents are to jointly perform σ. Note that the definition uses the concept of JPG, which was defined in section 4.2.1:

$$\begin{aligned} &\text{JOINT-RESPONSIBILITY}(\ll \Theta, \sigma \gg) \equiv \\ &\qquad \text{MBEL}(\Theta, JPG(\ll \Theta, \sigma \gg)) \wedge \\ &\qquad \text{MBEL}(\Theta, \text{SOL-COMMITMENT}(\ll \Theta, \sigma \gg)) \end{aligned}$$

The use of Joint Responsibility in a hierarchical organisation implies that each sub-coalition $\Theta_i \subseteq \Theta$ establishes its own joint responsibility. This case is similar to the Joint Intentions theory (Section 4.2.1) where each sub-coalition needs to establish a JPG.

An example of the use of Joint Responsibility is for a fault detection and diagnosis system for electricity transportation management. This scenario deals with the process of taking electrical energy from where it is produced to where it is consumed. Agents are used to jointly detect, and if possible, repair problems during this process. In this way, agents make use of joint responsibilities to develop common solutions to problems and inform about possible faults during the performance of the agreed solution. Details of several experiments in this domain can be found in [Jennings, 1995].

4.2.4 Planned Team Activities

The *Planned Team Activities* approach [Kinny et al., 1992] proposes a framework to specify joint plan activities where the participant agents have a repertoire of plans supplied in advance, rather than being generated by them. This approach is mainly based on the idea that agents embedded in dynamic environments can rapidly respond to events by adopting a pre-defined plan.

The planned team activities language contains individual agent constants and team variables. Ground teams are teams that contains no variables. The language also distinguishes between ordered and unordered teams, which can be used to compose new teams. Thus if $\Theta_1, ..., \Theta_n$ are teams, then $\{\Theta_1, ..., \Theta_n\}$ is an unordered team and $(\Theta_1, ..., \Theta_n)$ is an ordered team.

The performance of a plan is represented as a graph ρ (a labelled, directed, acyclic and/or graph). Nodes in the graph are denoted by n_i,..., n_j and the k'th directed edge from node n_i to node n_j is denoted by e^k_{ij}. The labels on the plan graph employ operators (*,!,?,ˆ) to specify a plan expression ω, which denotes the performance of a basic activity $*(\Theta,a)$, achievement of a proposition f $!(\Theta,f)$, testing of f $?(\Theta,f)$, or waiting for f $\hat{}(\Theta,f)$. We can return the team that occurs in each plan expression using the function TEAMOF(ω).

A plan is defined as a tuple$(p, f_{purpose}, f_{precondition}, \rho)$, where p is a unique plan name, $f_{purpose}$ is a goal proposition, $f_{precondition}$ is the circumstance under which the plan can be executed, and ρ is a plan graph. Given a library of plans P and a goal state g, the function PLANS-FOR(P,g) returns the subset of P whose $f_{purpose}$ is equivalent to g.

Planned Team Activities also defines some important functions to guide the processes of team formation and plan selection at runtime. For example, there is a function for role assignment that creates a mapping from roles in the plan to members of a team. This is achieved by matching the formal team in the purpose of the plan to the acting team that invokes the plan. Then, a substitution S is a matching from an ordered team Θ_{ord} to an ordered ground team Θ_{ground} if:

- Θ_{ground} represents a single agent (an agent constant) and $S(\Theta_{ord}) = \Theta_{ground}$; or

- $\Theta_{ord} = (\Theta_{ord_1}, ..., \Theta_{ord_n})$ and $\Theta_{ground} = (\Theta_{ground_1}, ..., \Theta_{ground_m})$ are teams such that n$\leq$m and $\forall$i = 1...n, $S(\Theta_{ord_i}) = \Theta_{ground_i}$

Then, given a ground team Θ_{ground} and a plan p whose formal team is Θ_p, a role assignment $S_p^{\Theta_{ground}}$ over Θ_{ground} for p is a matching from Θ_p to Θ_{ground}. $\alpha S_p^{\Theta_{ground}}$ refers to the member that performs a role α. The process S of defining the team Θ_{ground} that will perform a specific plan p explicitly encodes the need of communication because members of a team must first commit or not to each delegated role α.

A Θ_{ground} has the capabilities to execute a plan p if $p \in$ PLANS(Θ_{ground}) and there exists a role assignment $S_p^{\Theta_{ground}}$ such that $\forall w \in SUBGOALS(p)$, $TEAMOF(w)S_p^{\Theta_{ground}}$ has the capabilities to perform the plan expression w. In this way, a ground team Θ_{ground} has the capabilities to perform a plan expression w if:

- w = *(Θ_{ground},a) and $a \in$ CAPABILITY(Θ_{ground}); or
- w = ?(Θ_{ground},f) or w = ^(Θ_{ground},f); or
- w = !(Θ_{ground},f) and $\exists p \in$ PLANS-FOR(PLANS(Θ_{ground}),f) such that Θ_{ground} has the capabilities to perform p.

Finally, using such definitions and considering that TRANSFORM(α,p) is a function that returns a role-plan for the role α in p, the definition of joint intention for this approach can be written as:

$$\begin{aligned}\text{JINTEND}(\Theta,p) \equiv\ & \textstyle\bigwedge_{\mu \in ROLES(p)} JINTEND(\mu S_p^{\Theta},p) \wedge \\ & MBEL(\mu S_p^{\Theta}, JINTEND(\Theta,p)) \wedge \\ & \text{MBEL}(\mu S_p^{\Theta}, \textstyle\bigwedge_{\alpha \in ROLES(p)} DONE(!(\alpha S_p^{\Theta}, TRANSFORM(\alpha,p))) \supset \\ & \quad \text{DONE}(!(\Theta,p)))\end{aligned}$$

According to this definition, a ground team Θ has a joint intention toward a plan p given a role assignment S_p^{Θ} if every agent has the joint intention toward the plan, every agent believes that the joint intention is being held by the team, and every agent believes that all agents performing their respective role-plans results in the team performing the joint plan.

Note that this approach stresses the idea of finding suitable capabilities to achieve a joint goal, guiding, in this way, the team selection process [Tidhar et al., 1996]. However, as related in [Kinny et al., 1992], the approach makes no provision for organisational structure within a team or between teams, such as the presence of a manager, or other hierarchical relationships. The representation could be extended to permit this, possibly resulting in modifications to the amount and type of communication between agents.

The ideas of the Planned Team Activities theory were used in an air combat simulation system, which enables the definition and configuration of both individual aircraft agents and teams of aircrafts. Details of this application can be seen in [Rao et al., 1993].

4.3 Comparative Discussion and Requirements

A comparative analysis of the teamwork approaches can be divided in five principal topics: commitments, communication/monitoring, mutual support, particular problem target and option for hierarchical description. From this discussion it is also possible to extract the fundamental requirements that must be considered by collaborative systems.

Although the works discussed in the last section lead their approaches to deal with different technical problems, they agree that agents involved in collaborative tasks must implement a notion of commitment. For that, a first requirement associated with collaboration could be expressed as:

> ***Requirement 3:*** *the collaborative model must consider the establishment of commitments to joint activities, enabling consensus on plans or their constituent parts.*

Considering this requirement, the Joint Intention Theory (henceforth, JIT) employs a strong notion of commitments, stipulating that agents are committed to a joint goal until they believe that such goal is satisfied or come to the conclusion that it is impossible to reach it. SharedPlans relaxes this notion of commitment via an intentional attitude (a mutual belief that something holds), which allows that agents drop intentions, but still having a potential intention to do any activity that they believe will contribute toward the previous commitment. The Joint Responsibility (henceforth, JR) framework refines the JIT, adding notions of plan and joint plan commitments. This includes a specification of the conditions under which joint plan commitments can be dropped. The Planned Team Activities (henceforth, PTA) is also based on mental states such as joint intentions and mutual believes. However such mental states also involve explicit references to the roles that each agent is performing, so that agents must mutually believe that performing their respective role-plans results in the team performing the joint plan.

Associated with communication in teamwork environments, we can note from the study of these proposals that the support to the development of such a process can be summarised by the following requirement:

> ***Requirement 4:*** *the collaborative model must provide ways to disseminate information associated with progress, completion and failure of activities.*

For that end, JIT embodies a rigid communication approach, which defines that if a member in a collaboration comes to believe privately that the joint goal is satisfied or is impossible to achieve, it incurs a commitment to make this fact mutually known. Differently, in SharedPlans agents are not required to communicate when they drop intentions. Instead, communication is only one option in such situations because, in fact, they possibly still having an intentional attitude forward such intentions. Thus, the decision of communicating requires additional reasoning. The communication approach of JR is similar to JIT once that JR is an extension of JIT. However note that it does an important extension in adding information about the conditions under which the commitment was dropped. PTA, in a similar way as JIT, builds into their definition of joint intention a requirement that agents communicate. In particular, the method they use for transforming a general plan structure to a plan in which teams are assigned to specific activities adds communication actions so that a message reporting the failure or success of sub-activities is broadcast to the members of the team after the sub-activity is (or is not) executed.

The notion of *team support* allows that a team member supports the performance of other members. In fact this is a very important feature of teamwork, however this is not very well explored by the approaches discussed here. One such requirement for team support development could be expressed as:

> ***Requirement 5:*** *the collaborative model must underline the idea of mutual support, providing ways to the specification of useful information sharing mechanisms and creation of supportive activities.*

JIT does not discuss in detail the ways in which mutual support is generated. The principal reason is because JIT deals with plans at a very high level of abstraction and does not address partiality in a significant way. For example, it does not represent in detail partial plans for individual or joint activities. Thus it is not able to express how one such plan can support the performance of another. In contrast, SharedPlans discusses efforts toward this requirement. Its formalism will lead, for example, an agent to share some particular information that it believes will enable the team to do an activity, which it desires to be performed by the team. In addition, SharedPlans also represents notions of partiality in plans, including, for example, more detailed specification of plans for activities. JR also presents a better representation for plans, distinguishing, for example, basic from multiagent activities. However, like JIT, JR does not consider the idea of mutual support. PTA provides detailed specifications of plans via libraries of pre-defined plans. However, it does not consider either situations in which agents have partial plans, or the construction of a new plan structure using elements from individual members' libraries. Furthermore, PTA does not discuss ways to represent mutual support or helpful behaviour.

We can note from the discussion that each framework tailors its approach to deal with a specific target problem. JI is more concerned with the problem of when to communicate, in particular the need for agents to inform one another whenever they drop a joint commitment. SharedPlans focus on how agents could form commitments to the activities of other participants of the team (mutual support). JR addresses in more details the conditions under which joint activities may falter and how team members should behave toward each other in such circumstances. PTA is more concerned with how agents can organise a suitable team to achieve a given problem.

Finally the frameworks present different levels of difficulty to allow for the description of collaboration in hierarchies. JIT does not discuss a direct way to express teamwork ideas in such organisations, however it is possibly an easy adaptation if we consider each subgroup as a particular team. In SharedPlans this adaptation is more obvious because the approach already considers a hierarchical process of decomposition. JR brings a similar level of difficulty than JIT once that it is just an extension of JIT. Differently, PTA needs a more detailed analysis and possible extensions so that it supports notions of organisational structures. The resume of this discussion can be seen in Table 4.1.

4.4 Summary

This chapter has discussed the need for developing planning processes together with principles of collaboration. In this way, the teamwork research was introduced as the most widely accepted metaphor for describing the nature of multiagent collaboration. Four different teamwork approaches (Joint Intentions, SharedPlans, Joint Responsibilities and Team Work Activities) were summarised so that we were able to raise important requirements for the implementation of coalition support systems.

Three ideas were stressed during this discussion: commitments, communication and mutual support. The concept of commitments is the core part of the teamwork research and it is used as a way for agents to define joint intentions to perform a collective activity while they share a certain mental state. Communication is important to update such mental state, enabling activity monitoring and correct use of information. Finally, mutual support that accounts for providing a more obvious sense of collaborative behaviour for each agent, even though it is not so well explored by the existing teamwork approaches.

Theory	Commitment	Communication/ Monitoring	Mutual Support	Particular Problem	Hierarchical Description
Joint Intentions	Strong notion based on joint mental states	Rigid approach explicitly represented in the theory	Not supported	Communication analysis	Easy adaptation to support such descriptions
SharedPlan	Based on intentions and desires on intentions	Not explicitly represented. Requires additional reasoning	Allows useful information sharing	Initial effort for mutual support	Possible via its hierarchical process of decomposition
Joint Responsibilities	Expands the JIT ideas with the notion of commitment to plans	Similar to JIT but dealing with additional information about commitment failures	Not supported	Study about failures in joint activities	Same as JIT
Planned Team Activities	Joint mental states with reference for roles	Build on the joint intentions definition via the transformation function	Not supported	Team formation	Not supported

Table 4.1: Comparison of Teamwork Theories

Chapter 5

Toward a Human-Agent Teamwork Model

The use of teamwork ideas supports the performance of collaborative planning activities by agents of a coalition, however they do not consider situations where agents interact with human users. This chapter starts by analysing, in Section 5.1, which problems can appear when humans are involved in planning activities using "teamwork agents". Then, Section 5.2 introduces the concept of *adjustable autonomy* and its two principal approaches (agent-based and user-based), explaining how it can be used to deal with such problems. Section 5.3 discusses a solution based on the well-known framework of *Mixed-Initiative Interaction*, however incorporating several notions of adjustable autonomy. As in the last two chapters, the principal aim of this discussion is to extract the requirements, in this case associated with human-agent interaction, that are important to the development of coalition support systems.

5.1 Limitations of Teamwork for Human Interaction

While early research on teamwork was mainly focused on agent-agent interaction, there is a growing interest [Bradshaw et al., 2002] in various dimensions of human-agent interaction. However, this new way of thinking about teamwork applications must firstly face additional problems that interaction with human users can bring up.

To illustrate such problems, consider the following scenario. During a disaster relief operation, the command centre sends the activities that must be performed by each paired agent/human. A truck *t* receives the activity *a* of "allowing access to the region on fire" so that the fire brigades can access such a region. When *t*'s agent receives this activity, it presents several plan options for *t* performing *a*. However, supposing that *t* is an uncertain human being, he/she spends more time than expected thinking about the options, so that the fire brigades as they arrive find the roads still blocked. Thus the whole plan is delayed.

Based on this example, we can note that agent inaction while waiting for a human response can lead to potential miscoordination with other coalition members. Thus, a new requirement to the development of human-agent teamwork systems could be raised as:

> ***Requirement 6:*** *the human-agent model must enable the definition of adjustable methods that complement the decision making process of human users.*

If the agent was developed considering this requirement, it could, in certain time critical situations, decide by itself one possible option and present it to *t*. For that, the agent will need some level of autonomy to take decisions that normally must be taken by humans. In some cases this could be dangerous because the agent can put the human user in situations of risk.

Now, consider that *t* decides to clear a narrow road that gives access to the region on fire because the work will be easier and quicker. However this road, in the complete plan, is working as a debris deposit so that others trunks are using it to drop off their waste. According to this example, we can note that local decisions taken by a coalition member can seem appropriate for her/him, but may be unacceptable to the team. In this way, an additional requirement can be expressed as:

> ***Requirement 7:*** *the human-agent model must provide ways to restrict user options in accordance with the global coalition decisions.*

Note that the teamwork theories are not specified to deal with such requirements. In the first case (Requirement 6), commitments could consider a deadline to force some reasoning process in the agent so that it returns at least a "not possible" answer. However, in cases where humans are the final decision-makers, agents are not normally permitted to play such role. In the second case (Requirement 7), commitments on plans and activities do not have effects on individual decisions of agents. In order, an agent already has set a commitment when it accepts an activity. However the theories are not at a level of granularity to force commitments on details about how to perform each activity. Note that this could be very hard in terms of communication and time.

5.2 The Adjustable Autonomy Approach

The requirements discussed in the last section can be seen as a problem of finding a suitable level of autonomy to the agents. Depending on this level of autonomy, agents can only carry out user commands or, at the other extreme, replace human reasoning, making all necessary decisions. Figure 5.1 illustrates this idea, showing a discrete classification for agent systems that present some kind of interaction with human users.

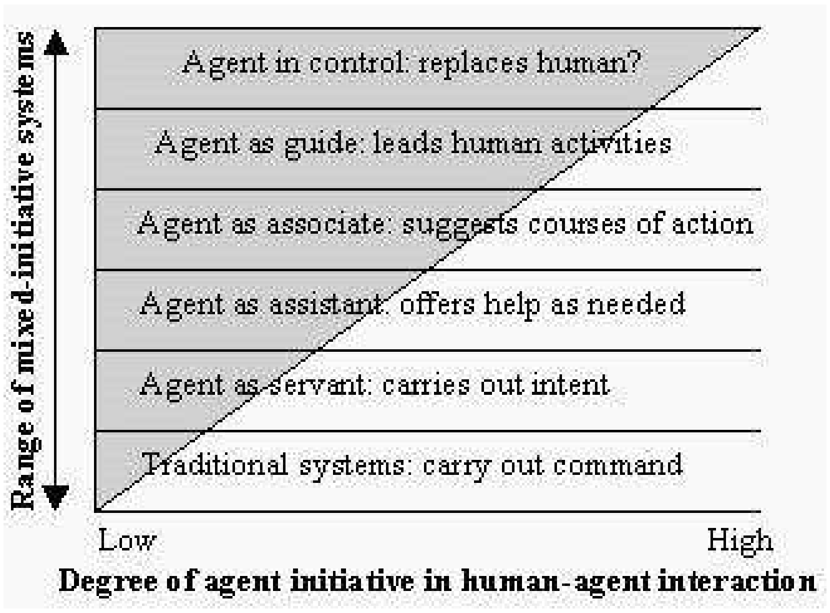

Range of mixed-initiative systems
Agent in control: replaces human?
Agent as guide: leads human activities
Agent as associate: suggests courses of action
Agent as assistant: offers help as needed
Agent as servant: carries out intent
Traditional systems: carry out command
Low
High
Degree of agent initiative in human-agent interaction

Figure 5.1: Spectrum of agent roles in human-agent interaction, showing degrees of agent initiative (adapted from [Bradshaw et al., 2002]).

Considering this idea, if the agents' autonomy is adjusted to a correct degree, this will allow them to exploit human abilities to improve their performance, but without becoming overly dependent or intrusive in their human interaction. Research in *adjustable autonomy* considers this idea, encompassing the strategies by which an agent selects the appropriate entity such as itself, a human user, or another agent, to make a decision at key moments when an action is required [Maheswaran et al., 2004]. These strategies can vary the level of autonomy of agents so that they require a different level of guidance depending on the current situation. The remainder of this section discusses the two different directions to formulate adjustable autonomy (user-based and agent-based autonomy), analysing in depth their features and advantages inside a collaborative planning environment.

5.2.1 Agent-Based Approach

In the *agent-based approach* [Scerri et al., 2001] to adjustable autonomy, each agent explicitly reasons by itself about whether and when to transfer decision-making control to another entity. For that, the transfer control process has to reason about the real advantages of such transference, facing the fact that it may have a high cost of interrupting the user who may be also unable to make and communicate a decision.

The agent-based approach has been used in the context of a multiagent system called *Electric Elves* [Chaulpsky et al., 2001]. In this system, individual user agents act in a team to assist with rescheduling meetings, ordering meals, finding presenters and other day-to-day activities. For that, agents are implemented through *Markov Decision Processes* [Puterman, 1994], which allow an explicit representation and reasoning about uncertainty of the world state and user interactions. Consequently, agents are able to dynamically vary their degree of autonomy, deciding when they need to interact with human users.

To illustrate the use of this approach regarding requirement 6, suppose that, when faced with uncertainty, a fire brigade agent consults its user (e.g., to check whether the user is able to work into the night), but the user, very busy in extinguishing a fire, fails to respond. While waiting for a response, the agent may miscoordinate with its teammates, since it fails to inform them whether the user will work during the night. Note that the agents, following the teamwork requirements, model each operation activity as a team commitment so that agents keep each other informed about progress, completion or failure. This in turn means that other members that depend on the work of the fire brigade waste their time waiting. Conversely if, to maintain coordination, the fire brigade agent tells the other agents that its user is not able to work during the night, but the user does indeed work, the team suffers a potentially serious cost from receiving this incorrect information. So, the fire brigade agent must instead make a decision that makes the best trade off possible between the possible costs of inaction and the possible costs of incorrect information.

The approach used in *Electric Elves* was innovative in considering the use of an agent-based approach to adjustable autonomy in a teamwork context, differently of previous works (e.g., [Dorais et al., 1998, Horvitz et al., 1999]) that have focused such approach on individual agent-human interactions. However, we can note that agents have considerable control of the interaction so that human users would not feel very confident. This feeling could become worse if we consider critical domains such as disaster relief operations or military missions where decisions can put humans in situations of risk. Furthermore, the approach does not consider the scenario where individual users take unacceptable decisions when dealing with their activities (Requirement 7).

5.2.2 User-Based Approach

Differently of the agent-based approach, the user-based approach to adjustable autonomy explores a human-centred perspective, where humans are seen as the crucial elements in the system and agents are fitted to serve human needs. The central issue for this approach is the design of mechanisms by which an user can dynamically modify the scope of autonomy for an agent.

A possible implementation of the user-based approach is oriented around the notion of *policies* [Sierhuis et al., 2003, Myers and Morley, 2003]. A Policy can be considered a declarative statement that explicitly bounds the activities that an agent is allowed to perform without user involvement. Thus, as long as the agent operates within the policy, it is otherwise free to act with complete autonomy. Human users can impose and remove the policies at any time, adjusting in this way the level of autonomy of agents. Consequently the main issue is to know how to specify good polices in accordance with the current scenario.

An example of policy use to implement the user-based approach can be seen in the

TIGER System [Myers and Morley, 2003]. TIGER provides interactive tools to help users to define and manipulate its policy framework. Such a framework assumes a BDI model of agency in which an agent has a predefined library of parameterised plans that can be applied to achieve assigned tasks or respond to detected events. Each class of plan instances in this library is characterised by an *activity specification*, which can be defined as $A =< F^+, F^-, R, C_r >$. The elements in this tuple respectively correspond to:

- A set of required *plan features*. A plan feature designates an intrinsic characteristic of a plan that distinguishes it from other plans that could be applied to the same task;
- A set of prohibited plan features;
- A set of plan *roles*. A plan role describes a capacity in which a domain object is used within a plan. For example, a pathfinder plan may contain variables, whose roles correspond to locations such as START, DESTINATION and so on;
- A set of constraints on how plan roles can be filled.

A *desire specification D* is defined in a similar way, however substituting the concept of plans by goals. Thus D has a set of required and prohibited goal features rather than plan features. Using both concepts, A and D, the formalisation introduces the notion of *agent context*, which is defined by a tuple $C =< B, D, A >$, where:

- B is a set of beliefs;
- D is the current set of desire specifications of μ;
- A is the current set of activity specifications of μ.

Agent context is the mechanism for delimiting the scope of agent reasoning. Via such a mechanism the framework defines two classes of policies:

- Permission Policy $< C, A >$: declares conditions under which an agent must obtain authorisation from the human user before performing activities (e.g., *Obtain* *permission before refilling the fire brigade water tank*). When the agent's BDI state matches the specified agent context C, permission must be obtained in order to execute a plan instance that matches the permission-restricted activity A;

- Consultation Policy: designates a class of decisions that should be deferred to the human user. They are defined into two types:
  - Role-fill Policy $< C, r >$: related to the selection of a value for parameter instantiation (e.g., *Let me choose specific roads to the final route*). The interpretation of a role-fill requirement is that when an agent's BDI executor state matches the agent context C, any instantiation decision for a variable corresponding to the consultation role r should be passed to the human;
  - Plan Policy $< C, D >$: related to the selection of a plan for a goal (e.g., *Consult me when deciding how to respond to requests of help*). Plan policy indicates that when an agent's BDI executor state matches the agent context C, the human user should be asked to select a plan to apply for any goal that matches the desire specification D.

In addition to permission and consultation policies, the proposal also defines the idea of *strategy preferences* that constitutes recommendations on how agents should accomplish assigned tasks. Together, such concepts support some idea associated with requirement 6. For example, if a fire brigade is very busy extinguishing a fire, it can give more autonomy for its agent and specify directions on how it must plan/act. Note, however, that this proposal does not discuss what must be done in case of an agent which asks something of its user and he/she does not reply. Furthermore, this proposal also does not explicitly discuss solutions toward requirement 7.

5.3 The Mixed-Initiative Approach

Mixed-Initiative Interaction refers to a flexible strategy in which each participant of the interaction contributes what it is best suited to do at the most appropriate time [Allen et al., 1999]. In this way, such an approach is useful to combine the abilities of humans and agents so that they synergistically work together. We can say that while 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.

Mixed-initiative interaction is not only used to combine the abilities of humans and agents, but also can be used as a technique for users to keep the control of agents. For example, TRAINS/TRIPS [Ferguson et al., 1996, Ferguson and Allen, 1998] is a mixed-initiative planning assistant that outlines the applicability of dialogue as a possible metaphor for the design of such systems. Figure 5.2 exemplifies part of a dialogue for a train route planning problem.

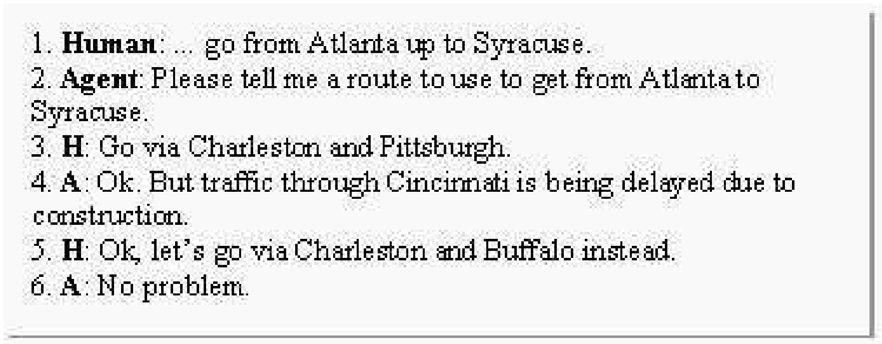
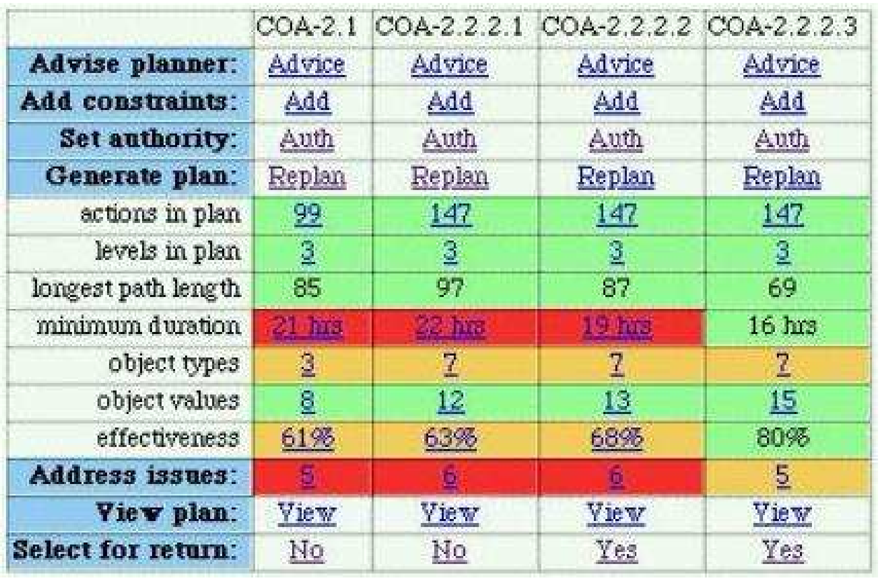

1. **Human**: ... go from Atlanta up to Syracuse.
2. **Agent**: Please tell me a route to use to get from Atlanta to Syracuse.
3. **H**: Go via Charleston and Pittsburgh.
4. **A**: Ok. But traffic through Cincinnati is being delayed due to construction.
5. **H**: Ok, let's go via Charleston and Buffalo instead.
6. **A**: No problem.

Figure 5.2: Example of part of a dialogue (from [Ferguson et al., 1996]).

In this example, humans are able to set high-level goals (such as in the step 1) while agents account for refining these goals. To that end, agents should ask questions to users, which provide details about the plan (such as in the step 3). Note that during this dialogue, the human is adding (step 3) or relaxing (step 5) restrictions on the deliberative process of the agent, and this is a form of controlling its behaviour.

The metaphor used by TRAINS/TRIPS involves some problems associated with natural spoken dialogue research. However, if we analyse a human-agent dialogue in a planning context, such as the dialogue in Figure 5.2, we can note that they are commonly a high-level way of manipulating restrictions.

O-Plan [Tate, 1997], another classical mixed-initiative planner, uses a more direct alternative to manipulate such restrictions. Via the O-Plan interface (Figure 5.3) users are guided through a planning process in which they can set initial assumptions, assign tasking level requirements for a *Course of Action* (COA) and select elements of evaluation, which are shown on a white background in the first column (e.g., *minimum duration* in Figure 5.3). The items in the matrix give the values for these elements, together with colours that specify whether the values raise any possible problematic issue (green = no issues, orange = points to note, red = violated constraints). Users can split any COA into two or more sub-options and explore further within each, adding, for example, new constraints and generating new plans. Together, these facilities allow for incremental development, exploration and evaluation of qualitatively different plan options.



	COA-2.1	COA-2.2.2.1	COA-2.2.2.2	COA-2.2.2.3
Advise planner:	Advice	Advice	Advice	Advice
Add constraints:	Add	Add	Add	Add
Set authority:	Auth	Auth	Auth	Auth
Generate plan:	Replan	Replan	Replan	Replan
actions in plan	99	147	147	147
levels in plan	3	3	3	3
longest path length	85	97	87	69
minimum duration	21 hrs	22 hrs	19 hrs	16 hrs
object types	3	7	7	7
object values	8	12	13	15
effectiveness	61%	63%	68%	80%
Address issues:	5	6	6	5
View plan:	View	View	View	View
Select for return:	No	No	Yes	Yes

Figure 5.3: Example of an O-Plan interface.

In both projects agents and humans can flexibly transfer initiative in decision making. TRAINS/TRIPS uses the idea of *turn-taking* models to address questions of when an agent is obliged to take the turn, when it cannot have the turn, and when it has an option of taking the turn or not. Differently, O-Plan envisages a mixed initiative form of interaction in which users and agents proceed by mutually restricting the plan. During this process, users can delegate to the system by adding suitable entries (with implied constraints) in an *agenda* for parts of the work that agents can handle best. Agents can seek help from the user via the same mechanism.

Mixed-initiative proposals, such as TRAINS/TRIPS and O-Plan, implement mechanisms to transfer control between agents and humans. Such mechanisms are a key aspect of the adjustable autonomy process so that mixed-initiative interaction can be used to adjust the degree of agents' autonomy. The implementation of adjustable autonomy on the mixed-initiative perspective is a suitable alternative because mixed-initiative research has also been investigating two fundamental model requirements of human-computer interaction, which can be expressed as:

> ***Requirement 8:*** *the human-agent model must support the definition of mechanisms that intensify the human user control and enable the customisation of solutions.*

> ***Requirement 9:*** *the human-agent model must support the generation of explanations about autonomous decisions, clarifying the reasons why they were taken.*

Requirement 8 refers to mechanisms that allow users to customise the solution according to their desires. Note that this requirement is likely to be antagonistic toward requirement 7. The idea should be that users can customise their solutions so that such solutions are still in accord with the coalition goals. In the TRAINS/TRIPS example (Figure 5.2), we can see that the user was able to customise the solution by indicating a preferential route (step 5). O-Plan also enables such customisation when users add entries in the agenda together with restrictions (e.g., ordering restrictions) for their performance. Note that the approach used in the TIGER system (Section 5.2.2) also presents an option for customising solutions via the concept of strategy preferences.

Requirement 9 is important because human users have an additional need of understanding what and why something is happening or will be carried out by the agent. This is particularly significant when an agent responds in a specific way to some previous human request. In TRAINS/TRIPS example (Figure 5.2), the agent returns an explanation (step 4) when an activity cannot be performed. In the same way, O-Plan allows users to set elements of evaluation that qualify a plan (COA) showing, for example, why it is not possible to perform it (values in red colour).

5.4 Summary

The involvement of human users in systems that implement the teamwork concepts is a very new experience and several opportunities are being created for research [Siebra and Tate, 2004]. The current proposals are still limited and they can only be deployed in specific cases. In this context we have identified two principal requirements (requirements 6 and 7) that must be considered during the evolution of multiagent teamwork systems to human-agent teamwork systems.

Research in adjustable autonomy is the current trend to deal with these requirements and it can be classified into two principal directions: user-based and agent-based autonomy. However we argue that such requirements can be dealt with the mixed-initiative approach. The principal advantage is that mixed-initiative research has a long history involving the interaction of humans and agents, already considering fundamental requirements for this (requirements 8 and 9).

Part III

Unifying Requirements

Chapter 6

<I-N-C-A>: a Constraint-Based Ontology

Part II of this thesis analysed three different threads associated with human-agent collaborative planning, extracting several requirements and discussing potential solutions for them. This part details how we are mapping such solutions into a unified framework, using constraint-based models and operations on them.

This chapter, in particular, presents <I-N-C-A>, the general-purpose constraint-based ontology that has been extended and adapted to support the definition of our planning models. We start by justifying the use of an ontology as a basis for plan representation. After that, we focus on the <I-N-C-A> components and how <I-N-C-A> can be used to describe plans.

6.1 Reasons to Use Ontologies for Multiagent Planning Representation

Ontologies are formal descriptions of concepts and relationships that can exist for organisations and their agents, creating a representation vocabulary, often specialised to some domain or subject matter. However, as stressed in [Chandrasekaran et al., 1999], it is not the vocabulary as such that qualifies as an ontology, but the conceptualisations that the terms in the vocabulary are intended to capture.

One important feature of ontologies for multiagent planning is that they are designed for the purpose of enabling knowledge sharing and reuse [Gruber, 1995] via agreements to use a vocabulary in a way that is consistent with the theory pre-specified. Note that for problems that require standard planners, where all the planning processes are commonly carried out for a single planning agent (e.g., *Sokoban*, *Towers of Hanoi*), the kind of planning representation can be designed considering features such as efficiency or expressiveness. Differently, the design of a multiagent planning representation must consider the fundamental need of providing a mutual understanding of the domain for all the agents. For that, ontologies are able to provide a shared underlying conceptual model, creating a framework for supporting collaborative processes of planning and execution.

The advantages associated with the use of an ontology for multiagent planning representation can be summarised into three features:

- It facilitates interoperation and communication between agents with common terminology;
- It promotes knowledge sharing between systems, integrating knowledge acquisition and modelling efforts, and;
- It creates a repository for general knowledge about planning to be used across several different application domains (reusability).

An example of ontology use in planning domains is the *Joint Forces Air Component Commander* (JFACC) ontology [Valente et al., 1999]. This project investigates a case study in building and (re)using an ontology for a specific application domain - air campaign planning. According to the authors, the experiments obtained the benefits envisioned, which are similar to those listed above.

6.2 <I-N-C-A> and its Components

<I-N-C-A> (Issues - Nodes - Constraints - Annotations) [Tate, 2003] is a general-purpose ontology that can be used to represent synthesised artefacts, such as plans

and designs, in the form of a set of constraints on the space of all possible artefacts in the application domain. The use of <I-N-C-A>, in this thesis, aims to underpin the representation of collaborative activities, respecting the requirements previously discussed. The next subsections describe each of the <I-N-C-A> components.

6.2.1 Issues

Issues are the <I-N-C-A> components that state the outstanding questions to be handled and can represent unsatisfied objectives or questions raised as result of analysis or other deliberative processes. For example, during the planning of activities associated with the rescue of civilians, an ambulance team can raise an issue representing a question like “Which are the hospital with facilities to treat burn injuries?”. This issue could be, for example, send to another agent with more expertise to answer it.

<I-N-C-A> has adopted the IBIS [Kunz and Rittel, 1970] (*Issue-Based Information System*) orientation of expressing issues as any of a number of specific types of question to be considered [Conklin, 2003]. The types of questions advocated that are likely to be the most common in the <I-N-C-A> task support environment are:

1. Deontic questions: What should we do?

2. Instrumental questions: How should we do it?

3. Criterial questions: What are the criteria?

4. Meaning or conceptual questions: What does X mean?

5. Factual questions: What is X? Is X true?

This use of issues in planning is similar to the *Question-Option-Criteria* (QOC) approach [MacLean et al., 1991]. At a high level, a planning session could be defined by the issues (questions) considered, the alternatives (options) posed and their justifications (criteria for those choices). In fact, the QOC approach was used for rationale capture for plans in earliest works associated with <I-N-C-A> [Polyak and Tate, 1998].

6.2.2 Nodes (Activities)

Nodes describe components that are to be included in an artefact (in our case, in a plan). Nodes can themselves be artefacts that can have their own structure with sub-nodes and other <I-N-C-A> described refinements associated with them (Figure 6.1).

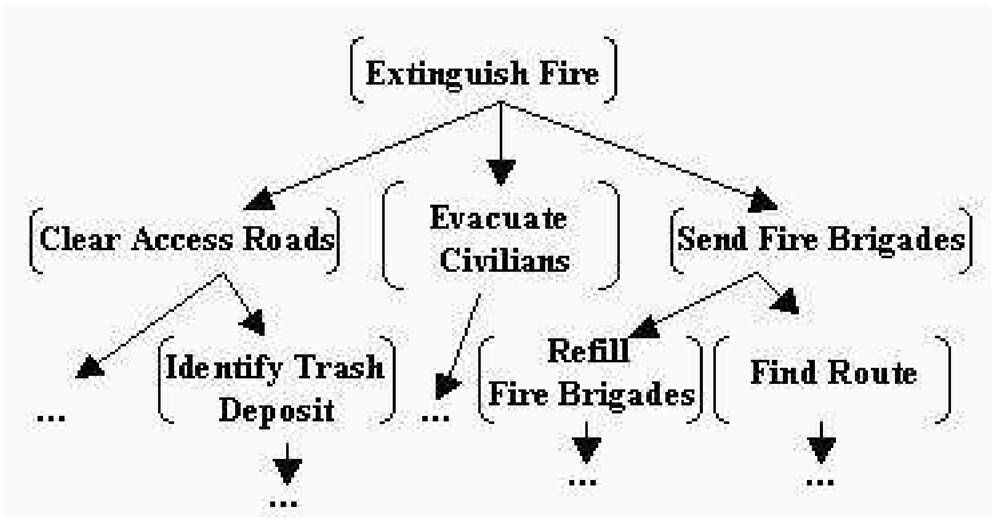


Figure 6.1: Example of hierarchical decomposition of activities.

When <I-N-C-A> is being used to describe plans as processes, the nodes are usually the individual activities and their sub-activities. They are usually characterised by a pattern composed of an initial verb followed by any number of parameter objects, noun phrases, qualifiers or filler words describing the activity. For example:

(**transport** *object* from *start-place* to *finish-place*)

In this example, the activity **transport** is characterised by the qualifiers “from” and “to”. The variables *object*, *start-place* and *finish-place* must be set before the activity’s performance. It is important to note that issues can be transformed to activities, implying further nodes. For example the issue “Which are the hospitals with facilities to treat burn injuries?” could be transformed to the activity: (**find** *object* with *capability*), where *object* = (hospital) and *capability* = (treat (burn injuries)). Details about the <I-N-C-A> syntax are given later.

6.2.3 Constraints

Constraints restrict the relationships between nodes to describe only those artefacts within the artefact space that meet the requirements. The constraints may be split into *critical constraints* and *auxiliary constraints* depending on whether some constraint manager can return them as *maybe* answers to indicate that the constraint being added to the model is okay so long as other critical constraints are imposed/altered by other constraint managers. For example, a fire brigade agent may extinguish a fire if restrictions on routes are made. The *maybe* answer is expressed as a disjunction of conjunctions ("or-tree" of possible solutions) on such critical constraints. The *yes/no/maybe* constraint management approach is detailed in [Tate, 1995] and generalises the *Model Truth Criterion* approach presented in [Tate, 1977, Chapman, 1987].

The choice of which constraints are considered critical and which are considered auxiliary is itself a decision for a particular application. Specific decisions on how to split the management of constraints within such an application are required. For example, a temporal activity-based planner would normally have object/variable constraints (equality and inequality of objects) and some temporal constraints (maybe just the simple "before" constraint: before timepoint- 1 time-point-2) as the critical constraints. But, in a 3D design or a configuration application, object/variable and some other critical constraints (possibly spatial constraints) might be chosen.

6.2.4 Annotations

Annotations account for adding complementary human-centric and rationale information to plans. In a general way, annotations can be seen as notes on plan components, such as nodes (activities) or issues, describing information that is not easily represented via the other <I-N-C-A> components.

6.3 Representation of Plans via <I-N-C-A>

Each plan represented via <I-N-C-A> is made up of a set of issues, a set of nodes and a set of constraints, which relate those nodes and objects in the application domain.

Annotations can be added to the overall plan, as well as specifically on any of its components. Figure 6.2 shows the first level of the <I-N-C-A> specification for plans, where we can see the declaration for such elements.

```
PLAN ::=
    <plan>
        <plan-variable-declarations>
            <list>PLAN-VARIABLE-DECLARATION</list>
        </plan-variable-declarations>
        <plan-issues> <list>PLAN-ISSUE</list> </plan-issues>
        <plan-issue-refinements>
            <list>PLAN-ISSUE-REFINEMENT</list>
        </plan-issue-refinements>
        <plan-nodes> <list>PLAN-NODE</list> </plan-nodes>
        <plan-node-refinements>
            <list>PLAN-NODE-REFINEMENT</list>
        </plan-node-refinements>
        <constraints> <list>CONSTRAINER</list> </constraints>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    <plan>
```

Figure 6.2: First level of the <I-N-C-A> specification for plans.

The first part of the specification is dedicated to the declaration of variables. Variables are characterised by a unique identifier, a name and a scope (local or global). Local variables are only visible by the component that is using them. Thus, we can have, for example, several local variables with the same name in a plan. Differently, global variables must have different names. Names that represent variables begin with the symbol "?" and such names can be used by several other components of the model.

Issues, in this specification, are not directly included in a plan. Instead, each issue is wrapped in a PLAN-ISSUE element (Figure 6.3). A pair of the elements PLAN-ISSUE and PLAN-ISSUE-REFINEMENT is used to relate an issue to its sub-issues.

```
PLAN-ISSUE ::=
    <plan-issue id="NAME" expansion="NAME">
        <issue>ISSUE</issue>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </plan-issue>

ISSUE ::=
    <issue status="STATUS" priority="PRIORITY" sender-id="NAME"
            ref="NAME" report-back="YES-NO">
        <pattern> <list>PATTERN</list> </pattern>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </issue>
```

Figure 6.3: Specification of issues.

The ISSUE element (Figure 6.3) is characterised by a status (blank, complete, executing, possible, impossible, n/a), a qualitative priority (lowest, low, normal, high, highest), an attribute to indicate the source of the issue (sender-id), a reference name for internal use, and a flag to indicate if the issue sender requires report-back.

The declaration of nodes (activities) is similar to the issues, so that nodes are also not directly included in a plan. Using the same idea of issues, nodes are wrapped in a PLAN-NODE element and the pair of the elements PLAN-NODE and PLAN-NODE-REFINEMENT is used to relate an activity to its subactivities. Thus, the specification of the elements PLAN-NODE and ACTIVITY (Figure 6.4) are similar to the elements PLAN-ISSUE and ISSUE respectively. In fact, issues are likely to be transformed in activities during the planning process.

The specification of constraints starts by the element CONSTRAINER (Figure 6.5). Such element is being used at the moment (version 4.0) because the idea of ordering is represented via instances of a separate element called ORDERING. However, ongoing versions tend to consider such ORDERING element as a normal constraint, which implements similar ideas to the "before" or "after" temporal constraints.

```
PLAN-NODE ::=
    <plan-node id="NAME" expansion="NAME">
        <activity>ACTIVITY</activity>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </plan-issue>

ACTIVITY ::=
    <activity status="STATUS" priority="PRIORITY" sender-id="NAME"
            ref="NAME" report-back="YES-NO">
        <pattern> <list>PATTERN</list> </pattern>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </activity>
```

Figure 6.4: Specification of nodes.

A constraint is characterised by a type (e.g., world-state), a relation (e.g., condition or effect) and a sender-id attribute to indicate its source. The constraint itself is described as a list of parameters, whose syntax depends on the type of the constraint. For example, a world-state constraint has as parameter a list of PATTERN-ASSIGNMENT, which is defined as a pair pattern-value such as ((speed wind),35km/h).

```
CONSTRAINER ::= CONSTRAINT | ORDERING

CONSTRAINT ::=
    <constraint type="SYMBOL" relation="SYMBOL" sender-id="NAME">
        <parameters> <list>PARAMETER<list> </parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>
```

Figure 6.5: Specification of constraints.

Finally we can see that annotations can be used as in the high level plan definition, as in each of its components. Annotations are represented by a set of key-value maps in which any object represented in the <I-N-C-A> specification may appear as a key or a value. The complete and current <I-N-C-A> specification for plans can be found in Appendix A.

6.4 Summary

Differently to the JFACC ontology (Section 6.1) example, which focuses on the particular domain of air campaign, <I-N-C-A> aims to be a general-purpose ontology able to represent a broad range of domains related to synthesis tasks such as designs, plans and configurations.

An interesting feature of <I-N-C-A> is that it has a clear description of the different components within a synthesised plan, allowing such a plan (or part of it) to be manipulated and used separately from the environment in which it was generated. This feature enables agents to work individually on different parts of the plan, but without losing the awareness of collaboration.

<I-N-C-A> is an ongoing topic of research so that its specification is likely to be changed as the work progresses. However, we have used the current version presented here as the basis for our thesis. In our case, in particular, we have adapted such a version so that it accommodates the requirements previously discussed (Part II). The next chapter details these adaptations.

Chapter 7

A Unified Representation

This chapter details how we are synthesising the requirements previously discussed so that they can be analysed from the same perspective. For that end, the chapter starts by showing an agent-centred view of the planning process, classifying different sets of constraints and functions that are related to this process. Each set of constraints or function is associated with a requirement, which guides its specification. An extended version of the <I-N-C-A> ontology together with a group of constraint manipulation functions are raised from this investigation, creating a unified representation for collaborative human-agent planning activities.

7.1 Classifying Constraints

An interesting way to understand the planning process is via the elements that have influence on it (e.g., world state, time, human users, etc.). Considering that each of such elements is represented as a set of constraints (C_i), we have the scenario illustrated in Figure 7.1. According to this figure, we can identify the following sets of constraints:

- C_0: represents the set of constraints imposed by the environment such as weather conditions or the state of world objects;
- C_1: represents the set of constraints that restricts the planning process in adding new activities into the agent's agenda (plan). In our representation it is specified via the temporal model (Section 7.2.1) according to *Requirement 1*;

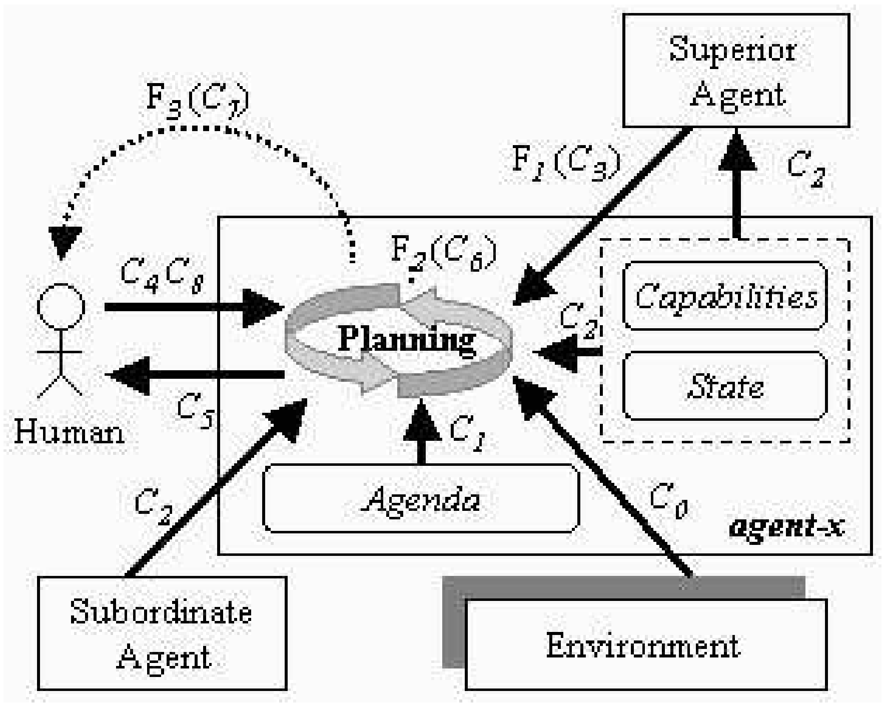


Figure 7.1: Constraints classification according to their roles in the planning process.

- C_2: represents the capabilities and internal state of agents that, together, restrict the kind and amount of activities that they can perform. Note that the set C_2 of subordinate agents also restrict the creation of plans by its superior. In our representation C_2 is specified via the resource model (Section 7.2.2) according to *Requirement 2*;

- C_3: represents the constraints associated with the process of commitment. Each delegated activity has one constraint C_3, whose value indicates if a specific agent is committed to the performance of such an activity. The value of C_3 is generated via the commitment function (Section 7.2.3 and 7.2.4), which is specified according to *Requirement 3 and 4*;

- C_4: represents the set of constraints used by humans to control/customise the behaviour of agents (Section 7.2.8), according to the *Requirement 8*;

- C_5: represents the set of constraints that restricts the options of the human user. Its definition (Section 7.2.7) is in accordance with *Requirement 7*;

- C_6: represents the set of constraints associated with activities of other agents. The function F_2 acts on C_6 to discover possible inconsistencies (Section 7.2.5) in such a set, so that agents can mutually support each other according to *Requirement 5*;

- C_7: represents the set of all constraints monitored by the function F_3, which accounts for generating explanations in accordance with *Requirement 9*;

- C_8: represents the set of constraints that accounts for providing a declarative manner of restricting the autonomy of the planning process (Section 7.2.6), according to *Requirement 6*.

Two simple properties can be defined to the constraint sets $C_0...C_8$. First, while some sets are composed of only one type of constraint (e.g., C_0: world-state, C_1: time and C_2: resource), other sets (e.g., C_6 and C_7) can be composed of several constraint types defined in the model. Second, a constraint c can appear in one or more sets, so that a constraint is not uniquely associated with a specific set.

The figure below (Figure 7.2) shows an example of constraint type definition. In this case for the set C_0 (world-state constraints). Using the general <I-N-C-A> model for constraints (Figure 6.5), we have:

```
CONSTRAINT ::= KNOWN-CONSTRAINT
KNOWN-CONSTRAINT ::=
      <constraint type="world-state" relation="condition" sender-id="NAME">
            <parameters> <list>PATTERN-ASSIGNMENT</list> </parameters>
            <annotations> <map>MAP-ENTRY</map> </annotations>
      </constraint>
```

Figure 7.2: <I-N-C-A> definition for world-state constraints.

The PATTERN-ASSIGNMENT element is represented by the tuple *(*pattern,value), where both are lists of any <I-N-C-A> element (strings, variables, activities, issues, numeric values, symbols, etc.). For world-state constraints we generally use the statement: *((attribute Object [attribute-qualifiers]),value)*. Based on this statement, we can specify the following examples of world-state constraints:

((number Civilians),1000)

((number Civilians disappeared),?amount)

((fieryness Building-x), 25%)

((state ?route (from station to Building-X)), clear)

In the first example, the attribute *number* of the object *Civilians* has the numeric value *1000*. Note that this example does not have the optional *attribute-qualifier* parameter. Differently, the second example qualifies the attribute *number* using the attribute qualifier *disappeared*. Note that the second example uses a variable, which is initialised by the question mark "?", to express that the number is not known yet. Variables can be used anywhere in these statements, such as to represent an object as showed in the last example (*?route*).

7.2 Synthesis of Requirements

7.2.1 Temporal Model

As discussed before, the development of a suitable temporal model for planning in hierarchical coalitions should be based on the following requirement:

> ***Requirement 1****: the temporal planning model must be based on an explicit timeline approach, which must enable the representation of both quantitative and qualitative temporal references as well as relations between them.*

In fact there are several and expressive ways that this requirement could be implemented [Allen, 1991, Freksa, 1992], so that it is not our intention to propose a novel temporal model for planning. Rather, we are using a simple set of the timeline ideas to show how a temporal model could be specified via <I-N-C-A>.

Considering the <I-N-C-A> representation, an explicit timeline approach indicates that each activity (node) has associated a constraint *I*, expressing its interval, with initial (I_i) and final (I_f) moments. Such a constraint could be defined as shown in Figure 7.3, where the relation attribute is set as *interval*. For this type of constraint we are composing the pattern, in the PATTERN-ASSIGNMENT element, by the node identifier; while the value by the tuple (I_i,I_f) where I_i and I_f can be variables if the moments are not known yet. Based on this definition, instances of pattern-assignment for temporal constraints could be specified as: (node-x,(10,20)) or (node-y,(0,?moment)).

```
KNOWN-CONSTRAINT ::=
    <constraint type="temporal" relation="interval" sender-id="NAME">
        <parameters> <list>PATTERN-ASSIGNMENT</list> </parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>
```

Figure 7.3: <I-N-C-A> definition for temporal constraints.

This time representation allows the expression of values for both certain (with numeric values) and uncertain (with variables) times. In case of certain time, the duration of an activity can directly be defined as the difference between the final moment and initial moment. Considering now that we want to set temporal relations between two activities a_1 and a_2, with respective intervals $I(a_1)$ and $I(a_2)$. For that end, first we must specify which are such relations and their meaning. Table 7.1 shows some examples of possible relations. However, several others can be defined using the same idea.

Relation	Meaning
Before	$\forall$ a_1,a_2 Before(a_1,a_2) $\Rightarrow I_f(a_1) < I_i(a_2)$
Equals	$\forall$ a_1,a_2 Equals(a_1,a_2) $\Rightarrow I_i(a_1) = I_i(a_2) \wedge I_f(a_1) = I_f(a_2)$
Meets	$\forall$ a_1,a_2 Meets(a_1,a_2) $\Rightarrow I_f(a_1) = I_i(a_2)$
During	$\forall$ a_1,a_2 During(a_1,a_2) $\Rightarrow I_i(a_2) < I_i(a_1) \wedge I_f(a_1) < I_f(a_2)$
Overlaps	$\forall$ a_1,a_2 Overlaps(a_1,a_2) $\Rightarrow I_i(a_1) < I_i(a_2) \wedge I_f(a_1) < I_f(a_2) \wedge I_f(a_1) > I_i(a_2)$
Finishes	$\forall$ a_1,a_2 Finishes(a_1,a_2) $\Rightarrow I_i(a_2) > I_i(a_1) \wedge I_f(a_1) = I_f(a_2)$

Table 7.1: Temporal relations.

The representation of temporal relations via <I-N-C-A> follows the structure shown in Figure 7.3, however with the relation attribute specifying the temporal relation (before, equals, etc.) and a simple tuple (a_1,a_2) as parameter rather than a

PATTERN-ASSIGNMENT element. a_1,a_2 are the identifiers of the nodes (activities) that are being related. Then, using the notation "relation-attribute(parameter)" to represent example of temporal constraints, we could have: before(*activity*$_1$,*activity*$_2$) that means activity$_1$ before *activity*$_2$. The figure below (Figure 7.4) illustrates a scenario where we can exemplify the use of this model to represent the temporal aspects of hierarchical coalition activities.

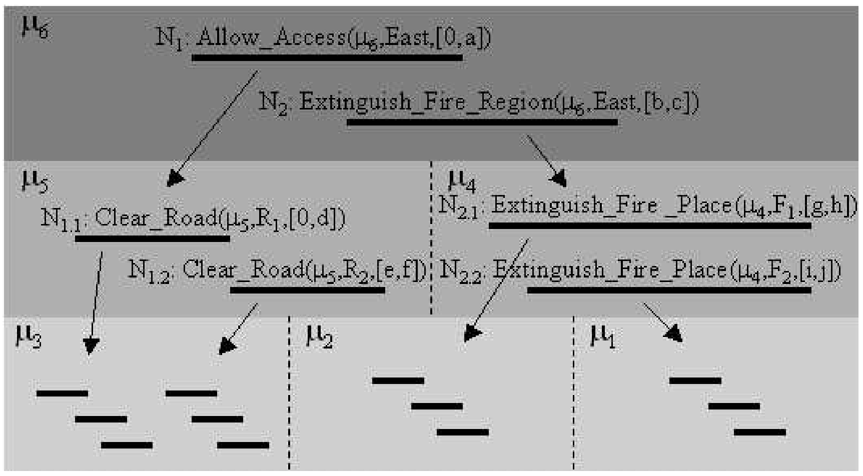

μ6
N1: Allow_Access(μ6,East,[0,a])
N2: Extinguish_Fire_Region(μ6,East,[b,c])
μ5
N1.1: Clear_Road(μ5,R1,[0,d])
N1.2: Clear_Road(μ5,R2,[e,f])
μ4
N2.1: Extinguish_Fire_Place(μ4,F1,[g,h])
N2.2: Extinguish_Fire_Place(μ4,F2,[i,j])
μ3
μ2
μ1

Figure 7.4: Example of activities and their intervals in a hierarchical coalition.

In this example, "East" is the region where the fires F_1 and F_2 are taking place. μ_6 represents the command and control centre (strategic level); μ_5 and μ_4 represent the police office and the fire station respectively (operational level); μ_3, μ_2 and μ_1 represent one police force and two fire brigade respectively (tactical level). Using the "relation-attribute(parameter)" notation again, the following constraints can be specified for each of the activities in the strategic and operational levels:

- Plan of μ_6: overlaps(N$_1$,N$_2$);
  - N$_1$: interval(N$_1$,(0,?a));
  - N$_2$: interval(N$_2$,(?b,?c));
- Plan of μ_5: before(N$_{1.1}$,N$_{1.2}$);
  - N$_{1.1}$: interval(N$_{1.1}$,(0,?d));

  - $N_{1.2}$: interval($N_{1.2}$,(?e,?f));

- Plan of μ_4: finishes($N_{2.2}$,$N_{2.1}$);

  - $N_{2.1}$: interval($N_{2.1}$,(?g,?h));
  - $N_{2.2}$: interval($N_{2.2}$,(?i,?j));

It is very difficult to determine durations for activities related to disaster relief operations. Thus their initial and final moments are likely to be variables. However, suppose that the agent μ_4 knows that the node $N_{2.2}$ will spend 15 time units. In this case, the interval constraint for this node could be specified as interval($N_{2.2}$,(?i,?i+15)).

We can conclude that this simple representation for C_1 supports the requirement 1. First it considers initial and final moments to activities so that they have explicit timelines. Second we can represent the notion of qualitative time, using temporal relations, and also quantitative values to express duration of activities. It is interesting to observe that the temporal relations (Table 7.1) are abstractions on numeric relations between initial and final activities' moments. A more expressive representation could enable relations on any two activities' moments, as for example, to specify exact overlap or during periods between activities. However, this approach increases the manipulation complexity of such a representation so that its implementation could not be justified.

7.2.2 Resource Model

The requirement previously discussed and used as a basis for the resource model specification is:

> ***Requirement 2****: the resource planning model must support the tasks of localising services/agents that provide specified capabilities, and also provide information that enables reasoning on such capabilities.*

In a similar way as discussed for the temporal model, it is not our intention to provide a complete ontology for resources. An interesting effort in this direction is DAML-S (or currently OWL-S) [Ankolekar et al., 2002], whose principles could be reused here. For now we are using a simple idea to specify resources and their features, which provide a basis for our experiments. Furthermore we show how such specification could be extended to provide a more detailed description for resources.

The original version of <I-N-C-A> (Chapter 6) uses a “pattern” in the activity element definition (Figure 6.4) to specify which capability an agent should have to carry out a specific activity. For that, the pattern is composed of an initial verb followed by any number of parameters, qualifiers or filler words. For example: (**transport** ?injured from ?x to ?hospital). Then the system finds agents to perform this activity by matching the verb **transport** with the capabilities (list of verbs) of the available agents. Thus, this simple mechanism supports the task of localising agents.

According to Requirement 2, the resource description should also provide information that enables reasoning on such capabilities. Imagine the scenario where a high building is on fire. To extinguish the fire in this building, the fire brigade should have a suitable ladder to reach the fire. However, using this simple capabilities description, fire brigades with and without ladders can be allocated to this activity because both are able to **extinguish** fires.

Based on this idea, we can define a new type of constraint to represent the features of a required resource as:

```
KNOWN-CONSTRAINT ::=
  <constraint type=“resource” relation=“RESOURCE-TYPE” sender-id=“NAME”>
        <parameters> <list>PATTERN-ASSIGNMENT</list> </parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
  </constraint>
```

Figure 7.5: <I-N-C-A> definition for resource constraints.

This constraint specification follows the same structure of the world-state or temporal constraints specification. Again we must detail the general form of the PATTERN-ASSIGNMENT element, which can be defined as below:

((resource object [resource-qualifier] [resource-range]),value)

In this statement, “object” represents the identifier of the agent that accounts for performing the activity. For example, if the activity is “Extinguish ?fire”, the object could be a “fire-brigade-x”. The attribute “resource” represents some object’s resource necessary for its operation. Such resource can be qualified via the attribute “resource-qualifier” and it can also have a range, which is typically used when the resource is consumable.

The table below (Table 7.2) shows some examples of resource specifications for a fire brigade. Note that the last two constraints have a special meaning and are only used for consumable resources. The “Consume” qualifier indicates the average rate that the resource is consumed. Similarly, the “Produce” qualifier indicates the average rate that the resource is produced. For example, for the fire brigade below the consume qualifier represents its pumping capacity and the produce qualifier represents its refilling rate.

Resource	Object	Qualifier	Range	Value
Water-tank	Fire-brigade-x	Amount	(0-12000000)	138763 ml
Fuel	Fire-brigade-x	Amount	(0,140000)	60000 ml
Fuel	Fire-Brigade-x	Type	-	Diesel
Stairs	Fire-Brigade-x	Height	-	20000 mm
Water-tank	Fire-Brigade-x	Consume	-	100000 ml/s
Water-tank	Fire-Brigade-x	Produce	-	10000 ml/s

Table 7.2: Example of resource specification for a fire brigade.

Then, using both ideas, the resource model can be seen on two levels. The activity pattern provides a simple high-level description of the capability required by the plan, while the constraints provide a more granular way to characterise or restrict the use of such capability. Considering a hierarchical coalition, superior agents can keep only the high-level description of their subordinate agents, mainly because such descriptions are stable. However, if they need more information, a query can be performed so that subordinates return their current resource attributes and respective values. This interaction between agents evinces the influence that the resource specification of subordinates has on the planning process of their superior agent.

7.2.3 Commitment Function

The temporal and resource models defined via <I-N-C-A> provides the essential requirements for the development of planning processes, also providing the basis for the definition of more detailed models. At this point, our aim is to incorporate the notion of collaboration into these planning processes. For that end, the first requirement to be considered is:

> ***Requirement 3:*** *the collaborative model must consider the establishment of commitments to joint activities, enabling consensus on plans or their constituent parts.*

According to <I-N-C-A>, each plan p is composed of a set of plan nodes n_i. If a superior agent, that has p as goal, sends such nodes to its subordinate agents, then a commitment must be made between them. For that end, Figure 7.6 defines a function called *AgentCommitment* that must be implemented by each hierarchical agent.

```
01. function AgentCommitment($sender$,$n_i$)
02.       static: $subplan$, list of nodes to be generated, initially empty
03.               $subordinates$, list of subordinates involved in $subplan$
04.       $subplan \leftarrow$ GenerateNodes($n_i$)
05.       if($\exists$ $subplan$)
06.             if (hasNodesToBeDelegated($subplan$)) then
07.                   Delegate($subplan$,$subordinates$) $\wedge$ WaitCommits()
08.                   if $\exists$ $s$ ($s \in subordinates$) $\wedge$ ($\neg$ Commits($s$)) then go to step 04
09.             REPORT($sender$,$n_i$,COMMITTED)
10.       else
11.             Report($sender$,$n_i$,NO-COMMITTED)
12.             $\forall$ s ($s \in subordinates$) $\wedge$ HasCommitted($s$,$subplan_s$)
13.                   Report($s$,$subplan_s$,NO-COMMITTED)
14. end
```

Figure 7.6: AgentCommitment function.

Based on this function, we can discuss some of its implications and features. First, n_i has a set of constraints associated with it. Thus the "GenerateNodes" function (step 04) considers such a set to return a subplan (list of nodes) to perform n_i. If there is a subplan (step 05) and it does not depend on anyone else (step 06) then the agent can commit to n_i (step 09). However, if the option depends on the commitments of subordinates, then the agent must await their answers (step 07). This implies that commitments are made between a superior agent and their subordinates and, starting from the bottom, an "upper-commitment" only can be done if all the "down-commitments" are already stabilised.

Second, if some subordinate is not able to commit (step 08), the agent returns to generate a new subplan (invalid commitments to the old subplan must be cancelled) rather than sending a no-committed 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 (e.g., a deadline for such generation is a possible reason) it reports a NO-COMMITTED to its superior (step 11). However the agent must also warn all their subordinates that *subplan* has failed and consequently their subnodes ($subplan_s$) can be abandoned (step 13).

This discussion has considered that agents only report simple commitment indications. However, extensions for this idea could enable the following situations: COMMITTED, when an agent commits to the performance of a node $n_i \in p$, it returns the new conditional constraints necessary for the performance of n_i; and NON-COMMITTED, if an agent does not commit to the performance of n_i, it returns the constraints that are avoiding the performance of n_i. Such constraints must be used by its superior during the generation of a new node list (subplan).

The principal feature of <I-N-C-A> that supports this function is its explicit representation for activities (nodes). That is, an activity can be seen as a complete subplan, which can be sent to agents others than the one(s) that generated it. However, to support this function <I-N-C-A> must also have a kind of constraint (C_3, Figure 7.1) associated with commitments. If we say that the activity of a superior agent is only valid if the subordinates agents commit on the performance of such activity's subnodes, then a commitment setting is, in fact, a conditional constraint to the activity. In this way, Figure 7.7 shows the <I-N-C-A> specification for commitment constraints.

```
KNOWN-CONSTRAINT ::=
    <constraint type="commitment" relation="COMMIT-TYPE" sender-id="NAME">
        <parameters> <list>PATTERN-ASSIGNMENT</list> </parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>
```

Figure 7.7: <I-N-C-A> definition for commitment constraints.

If a plan p has n nodes $node_{[1..n]}$ that are delegated to m agents $agent_{[1..m]}$, the PATTERN-ASSIGNMENT element for the commitment constraint can be defined as: ((node$_i$ agent$_j$),true/false). Then p is possible if every commitment constraint for each node$_i$ delegated to agent$_j$ is true. Considering this approach, p can assume three different status:

- Possible: indicates that all constraints (including commitment constraints) are met so that p is currently ready to execution;
- Not-Ready: indicates that p is not currently ready for execution because some conditional constraint might not be met yet;
- Impossible: indicates a failure (e.g., a false commitment constraint) for which recovery planning steps might be initiated.

We can note that the principal advantage of this approach is that commitments are naturally considered during the constraint processing. In other words, the failure in a commitment is interpreted as a problem in satisfying a constraint, which will trigger a common process of recovery (e.g., replanning).

7.2.4 Report Function

The next requirement associated with collaboration is:

> ***Requirement 4:*** *the collaborative model must provide ways to the dissemination of information associated with progress, completion and failure of activities.*

We are implementing this requirement via a function that monitors the constraints' values during the execution of activities. The principal questions that this function must answer is when to send a report and which information should be reported. There are two obvious cases in which an agent *ag* performing an activity n_i must report: first when *ag* finishes the performance of n_i and second when happens an execution failure and *ag* is not able to deal with the failure by itself. In the first case the agent only needs to send an activity completion status as information, while in the second case an activity failure status must be sent optionally together with the failure reasons.

Reports associated with progress are a more complex case. In order, agents do not have to communicate each step of their execution progress. Previous works already have identified communication as a significant overhead and risk, mainly in hostile environments [Tambe, 1997b]. 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 update. For example, if *ag* informs that it has just started the execution of n_i, it does not need to send other messages informing that it is still executing n_i. At this point we must think on which could be the information generated during the execution of activities and that is likely to help superior agents during the process of monitoring. For that end, let us use the following scenario: when *ag* commits to the performance of n_i, it informs the conditional constraints of the *subplan* that it is going to use. The constraints of such a *subplan* can have the following classes of values:

- *Concrete values*: represent known values, or estimated values if *ag* has a good idea about the process. For example, the temporal constraint interval(n_i,(30,60)) indicates that *ag* will (or is likely to) finish the activity n_i in 30 time units, or the resource constraint ((Fuel *ag* Amount),35000) to indicate that *ag* will (or is likely to) need 35000 millilitres of fuel to complete its activity;

- *Variables*: used during unpredicted situations. For example, interval(n_i,(?s,?f)) to indicate that *ag* does not have an idea about the duration of n_i yet, or ((Fuel *ag* Amount),?x) to indicate that *ag* does not have idea about the fuel it will need.

Using constraints with concrete values, the superior agent of *ag* can perform better reasoning. For example, if it knows that the activity of *ag* will finish in 30 time units, it can allocate *ag* to a new activity after 30 time units. Thus, if during the execution of *subplan* a concrete value of *subplan* is changed or its variables are instantiated, a progress report must be sent.

The report function is based on this discussion and can be seen as an extension of the commitment function (Figure 7.6). Thus, both ideas (commitment and reports) are represented by the same function (F_1 in Figure 7.1) that is defined below (Figure 7.8).

```
01. functionAgentCommitment($sender$,$n_i$)
02.     static: $subplan$, list of nodes to be generated, initially empty
03.             $subordinates$, list of subordinate agents involved with $subplan$
04.     $subplan \leftarrow$ GenerateNodes($n_i$)
05.     if($\exists subplan$)
06.         if (hasNodesToBeDelegated($subplan$)) then
07.             Delegate($subplan$,$subordinates$) $\wedge$ WaitCommits()
08.             if $\exists\ s\ (s \in subordinates) \wedge (\neg$ commits($s$)) then go to step 04
09.         Report($sender$,$n_i$,COMMITTED)
10.         while ($\neg$ Complete($subplan$))
11.             if (JustReady($subplan$) $\vee$ HasChanged($subplan$)) then
12.                 Report(sender,$n_i$,EXECUTING)
13.             if (Violated($subplan$) $\vee$ Receive(FAILURE)) then go to step 4
14.         end while
15.         Report(sender,$n_i$,COMPLETION)
16.     else
17.         Report(sender,$n_i$,FAILURE)
18.         $\forall$ s ($s \in subordinates$) $\wedge$ HasCommitted($s$,$subplan_s$)
19.             Report($s$,$subplan_s$,FAILURE)
20. end
```

Figure 7.8: AgentCommitment function extended to support reports.

According to this redefinition of the commitment function, commitments (and non-commitments that here are considered failures) are treated as a type of report, together with execution and completion reports. Then, after reporting a commitment (step 09) the agent keeps monitoring the constraint processing to identify when its subplan has been completed (step 10), changed (step 11) or violated (step 13). For each case, respectively, the agent reports a completion, reports an execution, or restarts the process until the subplan is completed or there is not a possible subplan. Note that a failure reported by subordinates is considered as a simple failure in the plan (step 13).

The function also expands from three (possible, not-ready and impossible) to five (complete and executing) the possible status values of a plan (or subplan). The <I-N-C-A> definition for activities (Figure 6.4) shows that each activity has a status attribute, which is filled with one of these status values.

The principal point of this function is that the reasoning associated with commitments and reports are based on results of constraint processing. This fact is illustrated by the functions "Complete" (step 10), "JustReady" (step 11), "HasChanged" (step 11) and "Violated" (step 13) that are all applied to the constraints of the agents' subplan. In this way, we still having the same basis for working, which was also used to support the planning mechanisms.

7.2.5 Mutual Support

The last requirement associated with collaboration is, in fact, the requirement that accounts for providing a real sense of collaborative behaviour to the members of a coalition.

> ***Requirement 5:*** *the collaborative model must underline the idea of mutual support, providing ways to the specification of useful information sharing mechanisms and creation of supportive activities.*

However, despite its importance, only the SharedPlans formalism discusses direct efforts toward this requirement, as was discussed in Section 4.3. In this thesis we are exploring the notion of mutual support via a function (F_2, Figure 7.1) that verifies the constraints (C_6, Figure 7.1) associated with activities of other agents. However, before characterising F_2 and C_6, it is important to understand the idea of mutual support that we are exploring here.

The principal idea behind mutual support is to enable that one agent has knowledge about the needs of other agents. For example, an agent *ag* knows that a specific road is clear so that it uses this constraint in its plan. However, as the world is dynamic, the road becomes blocked. If any other agent finds out that such road is no longer clear, it must inform this fact to *ag*. Thus, this informer agent is supporting the performance of *ag*. An easy option to implement this feature is to force that agents 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 a good approach in terms of communication and agents will also receive useless information.

Consider now that each agent ag_i, where $ag_i \in \Theta_x$, has a plan p_i with a set of conditional constraints C, which ag_i desires that hold so that p_i is still valid. In this case, each $c_i \in C$ is a constraint that ag_i believes to be true and hopes that it is still being true. Then ag_i broadcasts C for every agent $ag_j \in \Theta_x$ so that other agents of its sub-coalition know what it needs. A function based on this idea is defined below (Figure 7.9).

```
01. function MutualSupport($ag_i$,$C$,$mybeliefs$)
02.     static: $newactivity$, activity that can be created during the process
03.     while ($\exists c_j \; c_j \in C$)
04.         if ($\exists c_j c_k \; c_j \in C \wedge c_k \in mybeliefs \wedge$ Contrast($c_j$,$c_k$)) then
05.             $newactivity \leftarrow$ CreateActivity(Goal($c_j$))
06.             if ($\neg\exists$ $newactivity$) then Inform($ag_i$,$c_k$)
07.             Retire($c_j$,$C$)
08.         if ($\exists c_j \; c_j \in C \wedge \neg$Valid($c_j$)) then Retire($c_j$,$C$)
09.     end while
10. end
```

Figure 7.9: MutualSupport function.

This function follows the same idea of the AgentCommitment function (Figure 7.8) so that it is still based on constraint manipulation and verification. Another similarity is that the function also needs the id-sender attribute, which identifies the agent source ag_i of the constraint set C to be monitored.

According to the function, which is applied by agents that receive C from ag_i, agents must compare their *beliefs* (that is also a set of constraints) with C (step 04). If they find some *contrast*, they must try to create a new activity whose goal is to turn c_j true. If this is not possible, they must inform ag_i that c_j is not more holding and its new value is c_k. The idea implemented by this function is simple, however there are two more complex points: the “Contrast” and “Valid” functions.

The Contrast function is an extension of the Violated function (step 13, Figure 7.8). A violation is a type of contrast between two constraints. It says that two constraints, which are supposed to match, are not matching. However we are also considering as contrast the situation where two constraints have the potential to be equal. For example, ((colour Car),?x) and ((colour Car),blue). In this case the two constraints are in contrast because they have the potential to be equal if the variable ?x assumes the value “blue”. This type of contrast is very useful in the following class of situations. Suppose that one of the activities of ag_i is to rescue injured civilians. For that end, ag_i firstly needs to find such civilians so that it has 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 team agents that have or discover a set of constraints that contrast with the set sent by ag_i, they must inform ag_i about this new knowledge (note that it is not logical to create a new activity in this case).

The Valid function accounts for eliminating the constraints that no longer represent conditions to ag_i. This is important to avoid that ag_i still receives useless information and also to decrease the number of messages in the coalition. A practical way to do that is to consider that all $c_j \in C$ have a timestamp that indicates the interval where such a constraint is valid. At this point we are able to characterise the constraints for mutual support (C_6, Figure 7.1) as being a kind of world-state constraint with a pre-defined timestamp (t_i,t_f). In this way, its pattern assignment would be specified as:

((attribute object [attribute-qualifier$_1$ ... attribute-qualifier$_n$] (t$_i$,t$_f$)),value)

Using the timestamp (t$_i$,t$_f$) and considering that t$_i$ and t$_f$ are ground values, the Valid function only needs to compare if the condition ($t_f <$ current-time) is true to eliminate the respective constraint. However, as discussed before, it is likely that agents do not know when their activities finish so that such a temporal value will be a variable. Note that the principal advantage that we are looking for in using timestamps is to avoid that agents (C's senders) need to broadcast the information that they no longer need that a group of constraints holds. Rather, timestamps enables agents (C's receivers) to 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 [Siebra and Tate, 2005] because the sending of information is guided by the constraint-based knowledge that each agent has of the activities of its partners. Chapter 9 shows some experiments associated with this function. so that we can justify its real relevance.

7.2.6 Agents' Autonomy

As outlined previously, the first requirement associated with the involvement of human users during the collaborative planning process is:

> ***Requirement 6:*** *the human-agent model must enable the definition of adjustable methods that complement the decision making process of human users.*

As better detailed in the next chapter (Chapter 8), the transfer of control between agents and humans follows a *generate-evaluate-choose* sequence. First agents generate possible plan options to deal with the current set of activities, presenting such options to their users. Then users evaluate these options and, if they want to make changes, the process returns to the first step where agents generate a new set of options. Otherwise, users choose one of the options to be performed.

An alternative to decrease the impact of human delays in this process is to define contexts where agents choose by themselves the option to be performed. Such an approach was used in past projects, as discussed in Section 5.2.2, and it is very suitable because contexts can be redefined by users so that they are still in control of the situation. We define contexts associated with activities so that if such a context is true, a degree of autonomy is applied avoiding or allowing agents to take autonomous decisions. Constraints (C_8) that implement this idea have their type attribute instantiated with the key word *autonomy*, and the relation attribute with a *degree* representing the agents' level of autonomy (Figure 7.10).

```
KNOWN-CONSTRAINT ::=
    <constraint type="autonomy" relation="DEGREE" sender-id="NAME">
        <parameters> <list>PATTERN-ASSIGNMENT</list> </parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>
```

Figure 7.10: <I-N-C-A> definition for autonomy constraints.

Currently we are considering two degree values for these constraints: *permission* and *in-control*. The first is the default value so that if no autonomy constraint is specified, agents consider that they have to ask permission to insert (and perform if the agent is also an executor) the associated activity. The second means that agents can choose and perform the activity by themselves. The PATTERN-ASSIGNMENT element is defined as ((*attribute object*),*value*) so that a list of pattern assignments creates a context for possible values where the degree must be activated. The scenario below (Figure 7.11) illustrates the use of such type of constraint.

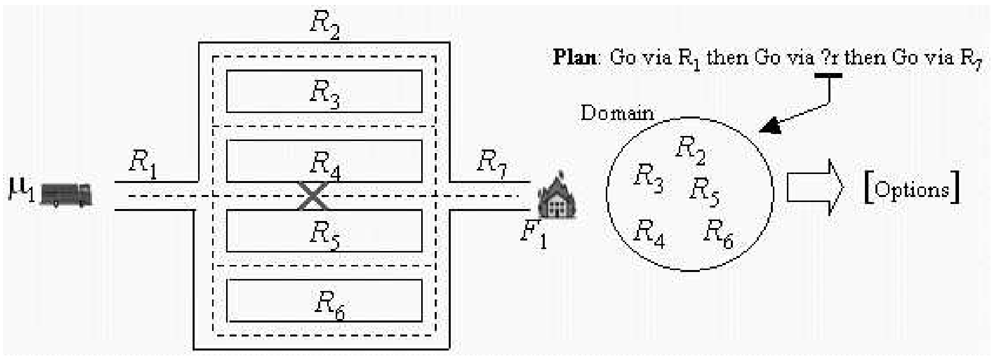


Figure 7.11: Scenario example for autonomy constraints.

In this scenario the agent μ_1 has the goal of extinguishing the fire F_1. For that, μ_1 must firstly create a plan to go to F_1. The plan illustrated in Figure 7.11 has three activities: Go via R_1, Go via ?r and Go via R_7. Suppose that the second activity has an autonomy constraint, whose degree is *in-control* and the pattern-assignment is ((position ?r),South). Such constraint means that if the agent planning process decides to instantiate ?r with a road of the South region (R_5 and R_6), it does not need to ask permission to the component identified in the sender-id attribute (normally its human user) to add the activity in its plan. Otherwise, if the agent decides for any other option (R_2, R_3 or R_4), it needs a permission.

A possible expansion of this approach is to implement the *consult* (degree attribute) autonomy constraint. The idea is to enable that users specify context where agents must consult humans about the instantiation of specific variables. Note that using permission constraints, agents only ask humans to confirm the choice of one activity. Differently, using consult constraints the process becomes more interactive and granular because humans interfere during the configuration of activities.

7.2.7 Users' Restriction

The second requirement associated with human interaction, as outlined before, is:

> ***Requirement 7:*** *the human-agent model must provide ways to restrict user options in accordance with the global coalition decisions.*

The idea here is to avoid that users take decisions that are prejudicial to the coalition as a whole. Agents can do that by restricting the options of users in creating their plans, or performing their activities.

The natural way to implement such restrictions, considering our constraint-based framework, is to define a set of constraints that cannot be changed by users or their planning agents. During the planning process users are able to manipulate constraints as a way to customise the outcome solutions, as discussed in the next section. However, it is clear that there is a group of constraints that users cannot change. This group is represented by constraints whose source is an external component such as the superior agent, or other team members. Based on this idea, the whole set of constraints can be divided into two classes:

- Internal constraints, which users can manipulate (read/write) and that are generally created by themselves during the process of customisation of solutions;
- External constraints, which users cannot change (read-only) because they are assigned by other components.

<I-N-C-A> enables a very direct support for such classification. As we can see in the constraint definition (Figure 6.5), constraints have an optional attribute (sender-id) that specifies the constraint source, the component that accounts for their creation. Thus the semantics associated with such attribute is very simple. If the sender-id attribute is assigned with the own agent/user identifier, the constraint is internal. Otherwise the constraint is external.

Despite the simplicity of such approach, it also presents a serious limitation. For example, suppose that a superior agent sends an activity to one of its subordinates. Associated with the activity, the superior agent also sends a group of constraints with some preferences on how to perform such activity. Note that these constraints are only preferences so that the subordinate agent is not obliged to follow the options expressed by such constraint group. It must do so if it is able to. Otherwise it can try other options. However this is not possible using our initial approach. As the constraints are external, they cannot be changed and planning must respect them.

This feature can be relaxed by implementing the idea of *weak constraints*. Weak constraints are, at first, used as normal constraints. However, if agents are not able to find any solution, they can eliminate weak-constraints, relaxing the restrictions on the plan options. Following this approach, world-state, resource and temporal constraints should also have their weak versions. Thus, the ontology needs to have ways to discriminate between normal and weak constraints. The next section shows how this discrimination is specified in our model.

7.2.8 User Control

The next requirement is:

> ***Requirement 8:*** *the human-agent model must support the definition of mechanisms that intensify the human user control and enable the customisation of solutions.*

As discussed before, this requirement seems to be antagonistic to Requirement 7. In fact we are looking for a mutual process of restriction so that while agents have constraints (C_5) to restrict the options of human users, users can also set constraints (C_4) to restrict the behaviour/reasoning of agents.

The approach defined in the last section provides the basis for user control. According to that approach, users can set normal constraints to force some result, or weak constraints to try such preferential solution if they are possible. To implement this approach (weak constraints) we have defined a new class of constraints (C_4) whose type attribute is specified as *preference* (Figure 7.12) and the relation attribute receives the type of a pre-defined constraint such as temporal or resource type constraints.

```
KNOWN-CONSTRAINT ::=
<constraint type="preference" relation= "PREF-TYPE" sender-id="NAME">
      <parameters> <list>PARAMETER</list> </parameters>
      <annotations> <map>MAP-ENTRY</map> </annotations>
</constraint>
```

Figure 7.12: <I-N-C-A> definition for preference constraints.

At this point we must investigate if the pre-defined constraints specified so far are able to express preferences. For that end, consider the same scenario illustrated in Figure 7.11. In this scenario, users can set preferences during the creation of plans using constraints to restrict durations (temporal constraints), use of resources such as maximum quantity of fuel (resource constraint) and so on. However it is not easy, using constraints in this way, to tell μ_1 that there is a preferential path, via R_5 for example. An option could be to set world-state constraints to say that the other roads $R_{2,3,4,6}$ are "blocked" so that the agent is forced to choose R_5. However, this is not logically an attractive solution.

A more practical way to deal with preferences is to associate them with the process of variable instantiation. Consider that such instantiation can be guided by a specific kind of constraint, whose function is to restrict values of constraint variables. This new type of preference constraint has its relation called *instantiation* and its PATTERN-

ASSIGNMENT element defined as: ((*set-definer-attribute variable*),*value*).

According to this pattern-assignment definition, the attribute *set-definer-attribute* defines the relation between *variable* and *value*, delimiting the space of options to instantiate *variable*. Possible examples are:

- ((> ?x),10) - indicates preference for values greater than 10;
- ((set ?r),R_5) - this is the most basic example for discrete values, where ?r should be instantiated with the value R_5;
- ((in ?r),[R_4,R_3,R_5]) - in this case ?r should be instantiated with any value that belongs to the set [R_4,R_3,R_5];
- ((in-ordered ?r),[R_4,R_3,R_5]) - in this case ?r should be instantiated with any value that belongs to the set [R_4,R_3,R_5], but considering that the values are listed in order of preference.

The last example is particularly important because it demonstrates the expressiveness of the syntax. In this case the user is not only setting a specific preference, but a set of discrete and disjunctive values listed by order of preference. Then if R_4 is not possible, the agent must firstly try the next option that is R_3 and so on. Based on these examples, the idea of *set-definer-attribute* can be expanded to consider diverse kinds of delimiters.

Note that autonomy and instantiation preference constraints have different semantics for their parameters, if we compare them with the other constraint types. While the autonomy constraint parameter characterises a context in which a specific degree of autonomy is valid, the instantiation preference constraint parameter indicates preferential values to be assigned to variables.

7.2.9 Explanations Function

The last requirement that must be considered is:

> ***Requirement 9:*** *the human-agent model must support the generation of explanations about autonomous decisions, clarifying the reasons why they were taken.*

This process of providing explanations is closely associated with the reasoning process that agents perform [Sqalli and Freuder, 1996]. For example, tracing through some problem solver techniques we could have: “I tried this and then I tried that, and it did not work so I backed up and tried the other...” (for backtrack searchers); “I tried this and then I tried that and it did not work but I decided to use it anyway...” (for simulated annealing local searchers); and “X had to be v because ...” (for inference-based solvers).

The idea here is to define a function (F_3, Figure 7.1) that produces explanations based on specific events triggered during the constraint processing. For example, if an activity becomes impossible due to a constraint, this event generates an explanation to the user saying which constraint is blocking the plan. In this case the user can, for example, relax or delete such constraint (only if it is an internal constraint). In this case the set of constraints C_7 associated with F_3 represents all constraints that are being manipulated by the planning process. Considering this idea, we must answer three questions: which are the agents' decisions that humans would like to have explanations for, which are the events that can trigger such explanations, and how can such explanations be produced.

To exemplify the explanation function design, consider one of the most common events that happens during planning and execution process: the invalidation of an activity in a plan. Users are generally interested to know the reasons for such invalidation. In other words, explanations for this case must be able to answer questions like: “why is the plan/activity x impossible?” or “why are we not able to perform x?”.

The second step is to identify the constraint processing events that can be used to generate explanations for such questions. We know that the invalidation of plan options is caused by conflict between constraints. Considering that the plan is in a stable state, some event must happen to cause a conflict. Such an event could be:

- Activity addition: constraints associated with new activities may be in conflict with current constraints;

- State changes: acquisition of new knowledge can change constraint values or instantiate constraint variables. This new configuration can generate conflicts, invalidating activities;

- User decisions: constraint manipulation by users can also create new configurations that generate conflicts and invalidate activities.

The approach designed to produce explanations it to use templates with variables, which are instantiated by such events. The idea is to translate the meaning of each event via a specific template. A template for the case of activity invalidation is:

activity is not valid because $c_{activity}$ is in conflict with $c_{trigger}$ set by *source*.

For this template F_3 must instantiate four variables: *activity* representing the activity that was invalidated, $c_{activity}$ representing the activity constraint in conflict , $c_{trigger}$ representing the constraint that has triggered the conflict and *source* representing the source of $c_{trigger}$. Generalising the idea used in this example, we can define F_3 as follows (Figure 7.13):

```
01. function CreateExplanation($p$)
02.     static: $event$, event that generates some explanation
03.             $template$, template for a specific event
04.     while (Valid($p$))
05.         $event \leftarrow$ CaptureEvent($p$)
06.         if ($\exists$ $event$) then
07.             $template \leftarrow$ GetTemplate($event$)
08.             $explanation \leftarrow$ InstantiateTemplate($template$,GetFeatures($event$))
09.             SetTemplate($p$,$explanation$)
10.     end while
11. end
```

Figure 7.13: Explanation function.

The idea of this function is to monitor the activity/plan p while it is valid. Then if some event associated with the constraint processing of p happens, the function captures such event (step 05) and, if it is an explanation trigger, the function gets a predefined template for it (step 07). Then the variables of this template are instantiated in accordance with several features (type, sender-id, constraint associated, etc.), creating the explanation for this event. Finally the function sets the explanation to p.

We are adding explanations for activities in the form of annotations. As defined in Chapter 6, every <I-N-C-A> component can have one or more annotations associated with it. We are using annotations, in particular, to keep the explanations for <I-N-C-A> activities. As annotations can be set for every component, the explanation function could be extended to automatically create several explicative forms of annotations to other components.

Another interesting feature of this explanation function is associated with the "Get-Template" function (step 07). Such a function can be implemented to find out the template in the solver class rather than using a default template. For example a pathfinder solver could provide specific templates associated with situations where the activity is impossible, such as the path was blocked or the vehicle does not have sufficient fuel to complete the path. Consider the two explanations below where the first uses the default template and the second a specialised template to the path clear condition.

> **Go from school to hospital** is not valid because **((status road-x),clear)** is in conflict with **((status road-x),blocked)** set by **world-state**.

> **Go from school to hospital** is not valid because **status** of **road-x** is **blocked** rather than **clear** according to the current **world-state**.

The specialised template for this example in particular is:

> *activity* is not valid because $c.pattern_{activity}$ of $c.object_{activity}$ is $c.value_{trigger}$ rather than $c.value_{activity}$ according to current *source*.

It is easy to see that the explanation function can produce more intelligent and readable explanations using specialised templates like that. Chapter 9 shows how we could implement this feature.

The explanation function defined here is able to generate high-level explanations for simple decisions taken by agents. However, the idea provides the basis to support the elaboration of more detailed explanations for questions like: "Why has the plan/activity x been chosen?", "Why is this solution better than that?" or "which are the possible values for a specific domain variable?". Note that while an explanation function must capture the notion of comparative parameters in the first two questions, the last question requires a better understanding of the planning domain.

7.3 Summary

This chapter has proposed a unified framework for the development of planning processes, which considers requirements of different research areas: multiagent planning, collaboration and human-agent interaction. Such a framework is underlined by <I-N-C-A>, a constraint-based ontology which was extended/adapted to support the implementation of these requirements. The principal ideas presented here are given in the form of constraint representations, summarised below (Table 7.3), and functions to manipulate these constraints.

C_i	Type	Principal feature (PAR means Parameter)
C_0	world-state	PAR: *((attribute object [attribute-qualifiers]),value)*
C_1	temporal	PAR for intervals: (*node-id*,($t_{initial}$,t_{final})) or PAR for temporal-relation: (*node-id-x*,*node-id-y*)
C_2	resource	PAR: ((*resource object* [*res-qualifier*] [*res-range*]),*value*)
C_3	commitment	Manipulated by F_1, PAR: ((*node-id agent-id*),*true/false*)
C_4	preference	Represents weak-constraints that can be eliminated in case of conflicts.
C_5	no specific	sender-id attribute corresponds to an external source
C_6	no specific	Set of external constraints used by F_2 during the mutual support process
C_7	no specific	Set monitored by F_3 to create explanations
C_8	autonomy	Implements *in-control* and *permission* as relation values. The constraint parameters are used to define contexts

Table 7.3: Summary of the constraint definitions.

Appendix B brings the complete proposed extension for the <I-N-C-A> syntax. Furthermore, Chapter 9 contains several practical demonstrations on how the ideas discussed here could be used to implement a real coalition support system.

Chapter 8

Customisation of Planning Mechanisms

The last chapter has showed how we can use a constraint-based ontology to express several ideas associated with the development of coalition support systems. This chapter focuses on the practical aspects of this approach, discussing how planning processes use such a representation during the creation and execution of plans. In this way, Section 8.1 details the planning architecture and the principal components associated with it. Section 8.2 describes the process of building plans using such an architecture. Section 8.3 explains how we can use the idea of planning interfaces to customise handlers according to the activities carried out at each decision level (strategic, operational and tactical). Finally, Section 8.4 presents the agents that implement such ideas together with their auxiliary tools.

8.1 Planning Architecture

The specification of the <I-N-C-A> ontology has been done in parallel with the development of an open planning architecture, called I-X [Tate, 2004], whose principal objective is to enable the manipulation of models based on such an ontology. This section details the principal concepts and components associated with the I-X architecture, showing how we are using them to implement our ideas.

8.1.1 The Planning Process Abstraction

The I-X 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> approach. The first cycle tries to create candidate nodes to be included into the agent's plan, respecting the current constraints of the activities, which are already in the plan. If the agent is able to create one of more candidate nodes, one of them can be chosen and its associated constraints are propagated, restricting the addition of future new nodes.

A simple way to understand this process is to follow the example below (Figure 8.1). The agent's plan contains the set of activities that it intends to perform. If the agent receives a new activity, it must generate planning *actions* that include this new node in its plan. Each action is a different way to perform this inclusion so that different actions generate different nodes' configurations.

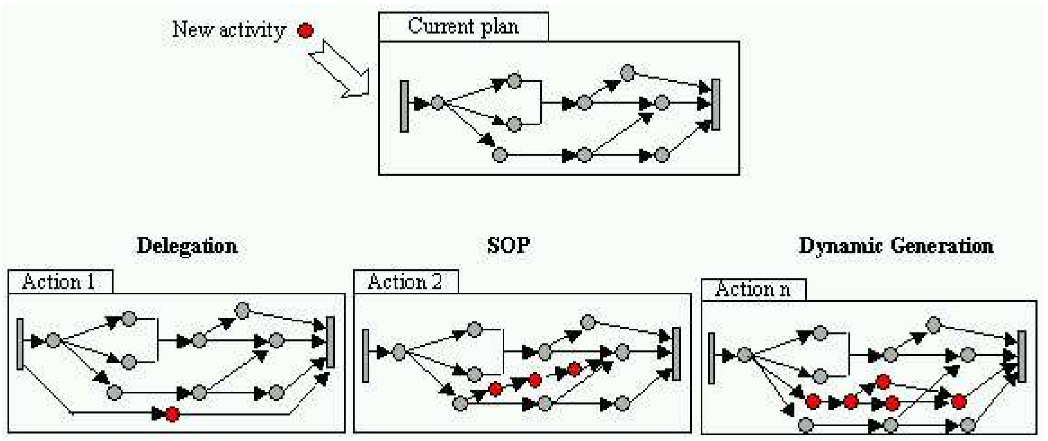


Figure 8.1: The activity-oriented planning approach

We call this process of *activity-oriented planning* because agents provide context sensitive actions to perform activities in specific. Common types of actions implemented under this perspective are:

- Delegation: action that simply sends a node to other agent that has the capability to handle it. Thus the unique work of the sender agent is to wait for the result;

- *Standard Operating Procedures* (SOPs): SOPs are sequences of pre-defined activities based on experiences, lessons learnt or carefully pre-designed. Depending on the context, agents turn available one or more SOPs as actions that decompose nodes. Appendix C shows an example of search patterns defined as SOPs;

- Dynamic Plan Generation: this action creates a dynamic plan, providing more assistance with a "How do I do this?" action associated with each node;
- Specific solver: actions can invoke specific solvers, such as a pathfinder, which are available as plug-ins.

We can say that the role of agents is to provide actions to decompose nodes until there are only executable nodes. The important point in this discussions is to know that each action is implemented by an activity (node) handler, which propagates the components through constraint managers to validate their constraints. For example, the action of applying a SOP is a handler that decomposes an activity according to the SOP specification. For this, the handler also causes constraint managers to check the conditions in which the SOP can be applied, indicating any conflict.

8.1.2 Activity Handlers and Constraint Managers

All agents have a set of activity handlers that they use to refine or perform their activities. The I-X architecture allows the definition of handlers as plug-ins, which can be declared in a configuration file so that agents load them during their initialisation. Each handler has a method (appliesTo) that indicates at runtime if it can be applied to a specific activity. In a general way, the process follows the steps below:

1. When an activity a is received, the agent's controller component selects (appliesTo method) a set of handlers H, which matches the description of a;
2. Each handler $h \in H$ uses one or more constraint managers to return the status of the action (possible, impossible or not ready);
3. Users choose one of the proposed actions, committing to the performance of a;
4. During the execution, constraint managers are still monitoring the constraints of a, warning in case of problems.

Note that the steps 1 and 2 correspond to the first cycle of the planning process, where constraints associated with each handler are tested against constraint managers so that they have to respect the current model. Meanwhile, steps 3 and 4 correspond to the second cycle, where the constraints of the chosen action are propagated and monitored by the managers.

The role of constraint managers in this process is to maintain information about a plan while it is being generated and executed. The information can then be used to prune search where plans are found to be invalid as a result of propagating the constraints managed by these managers. The principal advantage of using constraint managers is their modularity. We can design managers to deal with specific types of constraints, such as temporal, resource or commitment. In fact a guideline for the provision of a good constraint manager is the ability to specify the calling requirements for the module in a precise way. According to the constraints defined in Chapter 7, we could have the following constraint managers (CM):

- World-state CM: manages the constraints associated with the state of the environment and its objects;
- Temporal CM: manages the constraints associated with time, implementing the temporal relations defined in Table 7.1;
- Resource CM: manages the constraints associated with the description of agents, their capabilities and limitations;
- Commitment CM: manages the constraints associated with commitments, implementing the ideas of the *AgentCommitment* function;
- Autonomy CM: manages the constraints associated with autonomy;
- Preference CM: managers the constraints associated with preference, mainly guiding the instantiation of variables.

Together the constraint managers form the *model manager* of the agent. Each constraint manager considers a set of specific constraints in a well-defined syntax, based on the <I-N-C-A> formalism. In brief, constraint managers are intended to provide support to a higher level of the planner where decisions are taken. However, they do not take any decision themselves. Rather, they are intended to maintain all the information about the constraints they are managing and to respond to questions being asked of them by the decision making level [Tate, 1995].

8.1.3 Planning Interfaces

The idea of I-X in using planning interfaces is to provide a standardised connection between each constraint manager and other agents' components. For that end, it is important to understand the inputs that constraint managers need to manage their set of constraints, and which possible outputs are generated by them.

Inputs for each manager are represented by any constraint entry that must be tested against existing managed constraints to see what the impact of making the entry would be, and then a commit or abort can be done to add it or not. However the manager must also be informed about changes in its constraints, such as variable instantiations and deletions, because these events have impact during the constraint processing.

According to the I-X approach, all standard constraint managers return as output one of three results when they receive a new (or changed) constraint:

- **yes**: indicates that the constraint is now under management;
- **no**: indicates that the constraint cannot be added;
- **maybe**: indicates that the constraint can be added if plan entities are altered as specified via an *or-tree*, a data structure representing the alternative ways in which the plan may be altered[1].

The use of a well-defined interface has the advantage of allowing an easy implementation of new handlers and also the design of new constraint managers that can be dynamically added to an agent's model manager. For example, a pathfinder handler can add a constraint manager that monitors road states. This manager could return specific information to the handler according to the reasoning that it is performing.

[1]In this thesis we are not considering this option. More details can be seen in [Tate, 1995]

8.2 Building Plans

Following our extended version of the <I-N-C-A> ontology, a plan instance could be represented as follows (Figure 8.2):

```
01. <plan xmlns="http://www.aiai.ed.ac.uk/project/ix/">
02.     <plan-nodes> <list>
03.         <plan-node id="node-0" expansion="refinement-0">... </plan-node>
04.     </list> </plan-nodes>
05.     <plan-node-refinements> <list>
06.         <plan-node-refinement id="refinement-0" expands="node-0">
07.             <plan-nodes> <list>
08.                 <plan-node id="node-0-0">...</plan-node>
09.                 <plan-node id="node-0-1">...</plan-node>
10.                 <plan-node id="node-0-2">...</plan-node>
11.             </list> </plan-nodes>
12.             <constraints> <list>
13.                 <constraint type="temporal" relation="before">
14.                     ... (node-0-0,node-0-1) ...
15.                 </constraint>
16.                 <constraint type="temporal" relation="before">
17.                     ... (node-0-1,node-0-2) ...
18.                 </constraint>
19.             </list> </constraints>
20.         </plan-node-refinement>
21.     </list> </plan-node-refinements>
22. </plan>
```

Figure 8.2: Simplified example of an <I-N-C-A> plan.

This is a simplified example of an <I-N-C-A> plan, which omits several syntactical elements represented in the form of XML tags. The important point in this example is to understand its structure and the function of each element. Starting at step 02, the plan nodes element allows the association of several nodes (activities) with this plan, where each of these nodes will have a unique identifier. In this example we represent only one plan-node (step 03), whose identifier is “node-0”, and that also has an expansion attribute called “refinement-0”. Together, both attributes are used to relate nodes with their refinements to create hierarchical plans. In this case, the refinement of “node-0” is represented by three sub-nodes: node-0-0 (step 08), node-0-1 (step 09) and node-0-2 (step 10).

The two constraints (steps 13 and 16), associated with the “refinement-0”, indicate that the sub-nodes need to be performed sequentially. In the same way we could add temporal constraints (between step 21 and 22) to indicate some temporal restriction on the high-level node (node-0) of the plan. Other examples of constraints that could be in this list, between steps 12 and 19, are: ((node-0-0,?agent),?value) to indicate that a commitment to the performance of node-0-0 is required (commitment constraint) and ((in-ordered ?agent),[a_1,a_2,a_3]) to indicate that there is a preference on the agent (a_1, a_2 or a_3) that will be selected to perform node-0-0.

An important observation is that each plan-node inside the plan-nodes list (step 07) can also be decomposed in more nodes via refinements. In this way we are able to create a hierarchy of nodes so that each level has its associated constraints. For example, the decomposition of node-0-0, which is represented as <plan-node-refinement id=“refinement-1” expands=“node-0-0”>, could generate two new plan-nodes: <plan-node id=“node-0-0-0”>...</plan-node> and <plan-node id=“node-0-0-1”>...</plan-node>. For each of these nodes we can associate a new list of constraints, which are only valid in the corresponding level of the node.

8.3 The Customisation Design

The idea of planning interfaces provides a practical and uniform way to develop activity handlers and constraint managers. This is more useful when we think of a hierarchical organisation where the agents have to be customised to support different activities. This section shows the design of three handlers, one for each level of decision making, using our disaster relief domain as scenario. There are two important points to be noticed here. First, independently of their functional aspects, handlers have the same structure because they extend a super class that defines the essential methods for constraint checking and manipulation. Second, as the requirements discussed throughout the thesis are all represented via constraints, it is possible to control any aspect associated with planning via handlers. The remainder of this section stresses such points.

8.3.1 Planning Customisation for Strategic Agents

As discussed before (Section 2.3.1), the strategic level accounts for developing plans in a high-level abstraction, so that its principal tasks are related to the analysis of information and definition of directions.

Considering a disaster relief scenario where several fires are spreading over the city, one of the strategic level functions is to decide where to concentrate the coalition resources to avoid such spreading. Based on this idea, the planning model could contain an activity called "Avoid fire-spreading", which has a sub-activity called "Predict fire-behaviour". Then, for this activity we could design a handler (action) that receives the positions of the fires and, using information of the world-state module such as the speed and direction of the wind, this handler could return positions about where to place the fire brigades.

Based on this idea, let us define the handler in a more formal way. First, the activity that the handler can be applied to is specified as:

(**Avoid-fire-spreading** ?*region*)

The variable ?*region* specifies the area where the agent wants to avoid the fire spreading. Using a SOP or any other action we can decompose such activity in the sequential sub-activities below:

(**Localise** ?*list-of-fires* in ?*region*)
(**Predict-behaviour** ?*list-of-fires* ?*list-of-positions* ?*time*)
(**Protect** ?*list-of-positions*)

Following this order, the first activity intends to instantiate *list-of-fires* with the fires localised in *region*. Then, the second activity predicts the behaviour of the fire instances in ?*list-of-fires* during a specific *time* and returns the locations (*list-of-positions*) where the fire is going to. Finally, the last activity is associated with the protection of such positions and the agent can send it to a fire station, for example. The handler to be applied to "Predict-behaviour" activity can be defined as (Figure 8.3):

```
01. handler CalculateFireBehaviour(list-of-fires, time)
02.     static: list-of-positions, save positions that must be protected
03.     AppliesTo("Predict-behaviour");
04.     Handle()
05.         do ($\forall f \in$ list-of-fires)
06.             $s$ = modelManager.query(windSpeed(Position($f$)))
07.             $d$ = modelManager.query(windDirection(Position($f$)))
08.             pos = FireBehaviourPrediction(Position($f$), v, d, time)
09.             list-of-positions = list-of-positions + pos;
10.         end do
11.     return list-of-positions
12. end
```

Figure 8.3: Example of strategic level handler.

In this example we can see that this handler defines the method "AppliesTo" (step 03), specified in the activity interface, to indicate in which kind of activity it can be used. Another important method of this interface is "Handle", which is called when the handler is chosen. This handler, in particular, neither creates nor uses any constraint manager. Rather it only uses the "Query" method (steps 06 and 07), which is defined in the modelManager class, to acquire information about the environment. Then, the information about wind direction and speed for each position is used in a special function, which returns positions where the fires are likely to evolve during the specified time.

This is a typical handler at the strategic level, whose principal function is to analyse information about the scenario (wind's speed and direction), producing important knowledge (fire behaviour) that is used to support high-level decisions (in which areas the coalition must concentrate resources).

8.3.2 Planning Customisation for Operational Agents

The operational level accounts for refining the plans produced in the strategic level, mainly providing the logistical resources for them via processes as resource scheduling and load balancing (Section 2.3.2). Continuing in the disaster relief scenario, such level could be represented by fire station agents, which have groups of fire brigades as subordinate agents.

Consider that a fire station receives the (**Protect** ?*list-of-positions*) activity from the strategic level. Then one of its functions is to allocate its resources (fire brigades) based on the variable ?*list-of-positions*. One possible handler to deal with such activity is exemplified below (Figure 8.4):

```
01. handler AllocateResources(list-of-fire-positions)
02.     static: list-of-activities,list-of-resources,list-of-attributions
03.     AppliesTo("Protect");
04.     Handle()
05.         list-of-activities ← GenerateActivities(list-of-fire-positions)
06.         list-of-resources ← contactManager.query("available resources")
07.         list-of-attributions ← Schedule(list-of-resources,list-of-activities)
08. end
```

Figure 8.4: Example of operational level handler.

In this example we can note that the structure of the handler is similar to the strategic level handler, or any other handler developed on this architecture. In this case, however, there is a considerable number of interactions between the handler and the temporal module (constraint manager) in particular. This happens because the Schedule method (step 07) will try to add several temporal constraints into the current agent's plan that represent the new activities associated with the protection of each position. Thus, the application of this handler on the (**Protect** ?*list-of-positions*) activity will decompose it in several sub-activities of the form "**Protect** ?*position*" with receiver agents already associated with them.

As the definition of this handler is at a high level, several details are hidden. One of these details is related to the commitment constraints. When handlers create activities to be delegated, they automatically associate commitment constraints with such activities. In this case, the function that accounts for such association could be “GenerateActivities” (step 05).

8.3.3 Planning Customisation for Tactical Agents

The tactical level is where the execution of activities actually takes place. In our example, the tactical level is populated by fire brigades, which will receive the ”**Protect** ?*position*” activity produced in the operational level. However, as such an activity is not executable yet, the fire brigade must decompose it. For that end, it can apply a SOP that produces the following sub-activities:

(**Refill** water-tank)
(**GoTo** ?*position*)
(**Extinguish** ?*fire* in ?*position*)

At this point the three sub-activities are executable. However we can implement a handler to improve the efficiency of the “GoTo” activity. This handler could account for finding a suitable route between the current position of the fire brigade and ?*position*. This handler can be specified as follows (Figure 8.5).

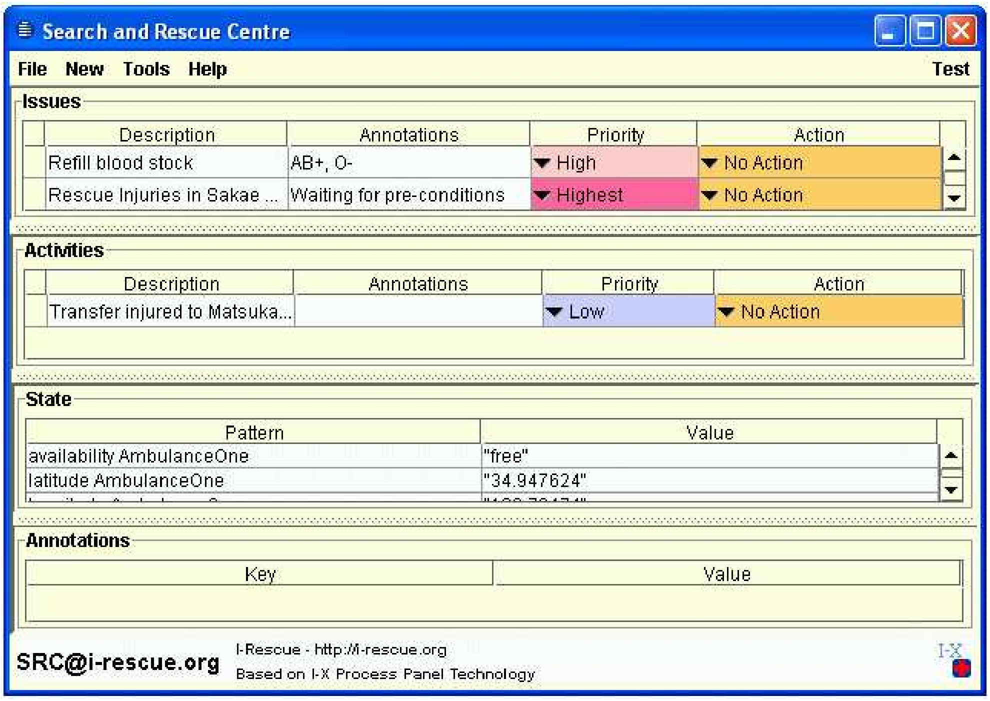
```
01. handler Pathfinder(position)
02.     static: list-of-roads, set of roads that the agent must use
03.     AppliesTo(“GoTo”);
04.     Handle()
05.         list-of-roads ← Router(position)
06. end
```

Figure 8.5: Example of tactical level handler.

According to this definition, if the “Pathfinder” handler is applied to the “GoTo” activity, a list of roads from the current position of the agent to the specified position is generated. The Router function (step 05) can implement any pathfinder algorithm according to the route representation used by the application. Note that while the Router function is creating a route, it interacts several times with the constraint managers to know if this route option is valid. For example, it has to ask the status of roads to the world-state module before including them in its outcome.

8.4 Planning Tools

The I-X project seeks to deliver useful functionality based on the <I-N-C-A> ontology via *I-X Process Panels* (I-P^2) [Tate et al.,2002]. A panel shows the current state of collaborative planning (from the perspective of the panel's user) through the presentation of the current items of each of the four set of entities comprising the <I-N-C-A> model (Figure 8.6).



Search and Rescue Centre

File New Tools Help Test

Issues

Description	Annotations	Priority	Action
Refill blood stock	AB+, O-	High	No Action
Rescue Injuries in Sakae ...	Waiting for pre-conditions	Highest	No Action

Activities

Description	Annotations	Priority	Action
Transfer injured to Matsuka...		Low	No Action

State

Pattern	Value
availability AmbulanceOne	"free"
latitude AmbulanceOne	"34.947624"

Annotations

Key	Value

SRC@i-rescue.org I-Rescue - http://i-rescue.org Based on I-X Process Panel Technology I-X

Figure 8.6: I-X Process Panel and its 4 sub-panels.

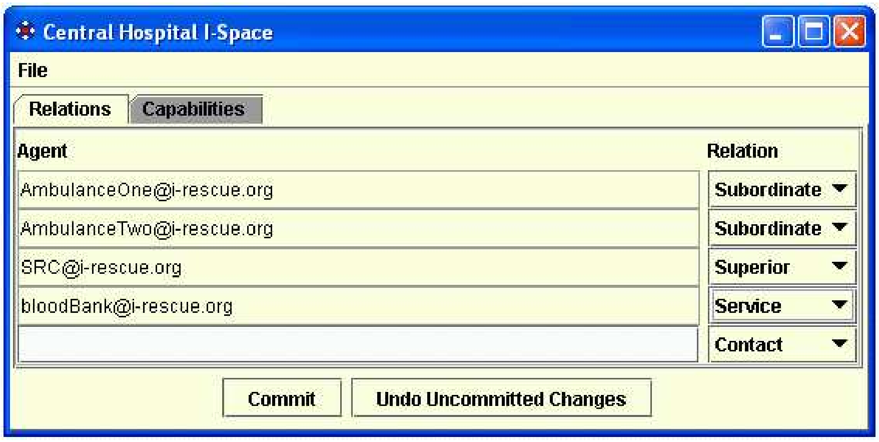
The principal objective of I-P^2 is to provide support to joint planning and execution activities of a team or coalition. Each agent of such a coalition receives its activities via an I-P^2, whose contents, along with the current context and state of the collaboration, are used to dynamically generate possible planning actions. For example, as discussed before (Section 8.1), associated with a particular activity might be suggestions for refining or executing it using known SOPs, for invoking a specific solver, or for delegating this activity to some other agent in the coalition.

For any activity on the panel, an I-P^2 panel row shows its current execution status and an "Action" menu with the available options to handle the activity. Colours indicate the readiness of the specific action for use:

- White: indicates that the item is not currently ready for execution (i.e., some precondition defined by some constraint is not met yet);
- Orange: indicates that the action is ready to perform and that all preconditions and constraints are met;
- Green: indicates that the item is currently being performed;
- Blue: indicates successful completion;
- Red: indicates a failure for which failure recovery planning steps might be initiated.

Activities and other panel items can be passed from one panel to another (or to capable services or other agents). These can pass back reports of success or failure to the original sender of the item. This provides a way to monitor activity progress, receive back milestone reports and check off the completion of activities.

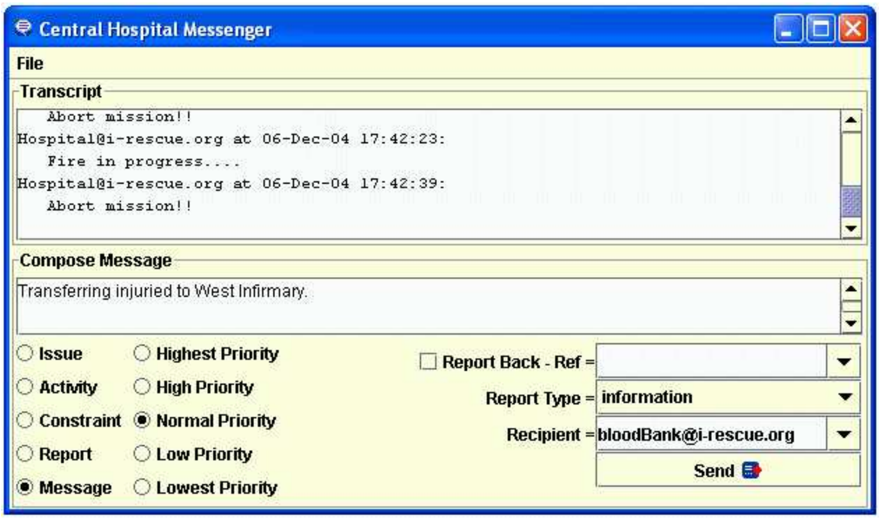
I-P^2 has additional tools that also support the process of planning and execution of human users. Examples are: *I-Space*, *I-Messenger*, *I-Domain Editor* and *I-Viewers*. The I-Space tool (Figure 8.7) supports the arrangement of coalitions, allowing the management of organisational relationships such as superior-subordinate or peer-peer. Considering an agent *ag*, I-Space shows the kind of relationship that *ag* has with other agents of the coalition (superior, subordinate or peer). For each of these relationships we can associate specific forms of interaction, which characterise each relationship in specific. In addition, I-Space also shows the capabilities of each agent that composes the contact list of *ag*.



Agent	Relation
AmbulanceOne@i-rescue.org	Subordinate
AmbulanceTwo@i-rescue.org	Subordinate
SRC@i-rescue.org	Superior
bloodBank@i-rescue.org	Service
	Contact

Figure 8.7: I-Space tool.

I-Messenger (Figure 8.8) supports the change of knowledge and manual monitoring of activities via reports of progress, success or failure. The knowledge sharing can be done via formal (pre-formatted messages) and informal (chat-like) messages. This tool, and the whole architecture, operates independently of communication protocols[2].



```
   Abort mission!!
Hospital@i-rescue.org at 06-Dec-04 17:42:23:
   Fire in progress....
Hospital@i-rescue.org at 06-Dec-04 17:42:39:
   Abort mission!!
```

Transferring injuried to West Infirmary.

Issue, Activity, Constraint, Report, [✓ Message]

Highest Priority, High Priority, [✓ Normal Priority], Low Priority, Lowest Priority

☐ Report Back - Ref =

Report Type = information

Recipient = bloodBank@i-rescue.org

Figure 8.8: I-Messenger tool.

The I-Domain Editor (Figure 8.9) enables the definition of refinements (e.g., SOPs) and creation of libraries of pre-defined plans. Rather than being used during missions, such an editor is commonly used off-line, preparing robust and well defined-plan com-

[2]I-X can use a wide variety of communication methods via plug-ins.

ponents or complete plans. Then, during missions, I-X agents are able to automatically turn available the procedures/plans that can be applied into a specific context.

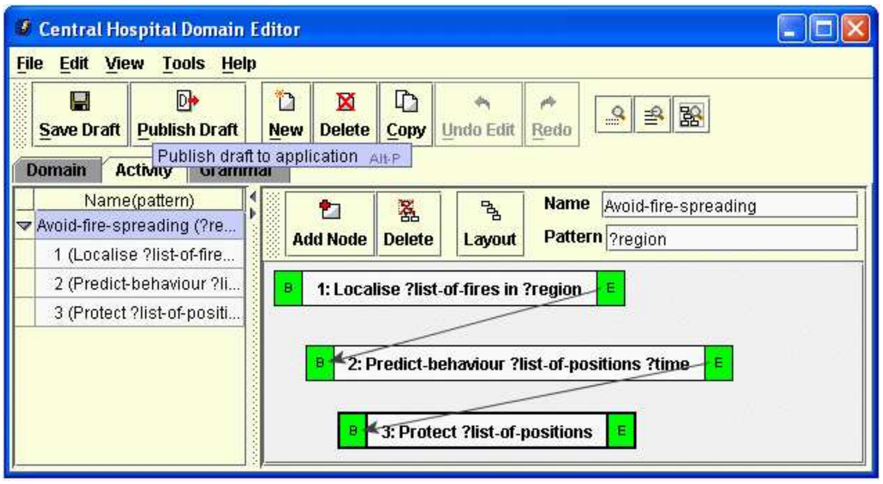

Central Hospital Domain Editor
File Edit View Tools Help
Save Draft
Publish Draft
New
Delete
Copy
Undo Edit
Redo
Publish draft to application Alt-P
Domain
Activity
Grammar
Name(pattern)
Avoid-fire-spreading (?re...
1 (Localise ?list-of-fire...
2 (Predict-behaviour ?li...
3 (Protect ?list-of-positi...
Add Node
Delete
Layout
Name
Avoid-fire-spreading
Pattern
?region
B 1: Localise ?list-of-fires in ?region E
B 2: Predict-behaviour ?list-of-positions ?time E
B 3: Protect ?list-of-positions E

Figure 8.9: I-Editor tool.

Finally the I-Viewers represent a set of tools that explores appropriate (sometimes domain specific) ways for showing some kind of information generated by the I-X agents. Two current examples developed during this thesis are the *Map Viewer* and *3D viewer*. The need to develop a Map Viewer is justified by the kind of domain that we are working with. Generally, members involved in disaster relief operations or military missions, for example, require a visual representation of the scenario where they are performing, which shows positions of resources and issues. For that end, a map-based representation is the most natural way to bring this kind of information in a bi-dimensional plan. The 3D Viewer can be used in missions that require a more detailed description of the scenario. For example, in a disaster relief operation where a rescue team is looking for injured civilians inside a large building.

At the moment the I-X Process Panels only can be used in a PC or laptop based environment. However research associated with the I-X aims to adapt this technology to more limited platforms (e.g., PDAs, mobile phones, etc.) so that users on the move, such as rescue team members, can use it. A first step in this direction [Lino et al., 2003] is the development of planning information delivery processes that look for the most "legible" way to present information based on several features such as profile of devices, user preferences and information content.

8.5 Summary

The I-X architecture defines the practical aspects to use the <I-N-C-A> ontology. The principal advantage of I-X for our work is its proposal of openness, which enables an easy extension of its capabilities (e.g., activity handlers, constraint managers and viewers) and configuration for different domains (e.g., disaster relief, space applications, etc.). In fact, the implementation of such ideas are being developed in the form of a JAVA *Application Program Interface* (API), which already supports the openness and extensibility features.

The I-X Process Panels, together with their auxiliary tools, have been demonstrated in different scenarios such as Coalition and Multinational Forces Command and Control [Tate et al., 2004], and Search and Rescue Coordination [Siebra and Tate, 2003]. Such demonstrations were based on the original <I-N-C-A> version, presented in Chapter 6. The next chapter discusses experiments that include the ideas presented through this thesis, mainly in Chapter 7.

Part IV

Experiments, Results and Conclusion

Chapter 9

I-Kobe: A Practical Application

I-Kobe[1] is a prototype of a hierarchical coalition system that supports disaster relief operations in the RoboCup Rescue Kobe scenario. This chapter details how I-Kobe has considered several ideas discussed through this thesis during its implementation. For that end, Section 9.1 introduces the Kobe domain, explaining the motivations for its use. Then, Section 9.2 presents the *RoboCup Rescue Simulator*, a disaster relief simulator that was configured to represent a scenario similar to a specific region of Kobe city. Section 9.3 describes the architecture of our application, showing the interaction between its components. Section 9.4 discusses several experiments based on this architecture. Such experiments can be divided into two groups: the first focuses on collaborative aspects, while the second on user interaction aspects. Finally, the last section discusses the results together with advantages and limitations of the system.

9.1 Urban Disaster Relief Domains

The occurrence of disasters in urban areas is the most critical event in terms of human lives due to the population concentration and considerable number of buildings. The *Great Hanshin Earthquake* or *Kobe Earthquake* is an example of how disasters have tragic effects in urban areas. On Tuesday, January 17th 1995, at 5.46 a.m. (local time), an earthquake of magnitude 7.2 on the *Richter Scale* struck the Kobe region of south-

[1] I-Kobe is part of the *I-Rescue Project* (http://i-rescue.org), an effort toward the development of knowledge-based tools applied to search and rescue or disaster relief domains.

central Japan. This region is the second most populated and industrialised area after Tokyo, with a total population of about 10 million people. The ground shook for only about 20 seconds, but in that short time over 5,000 people died, over 300,000 people became homeless and damage worth an estimated £100 billion was caused to roads, houses, factories and infrastructure (gas, electric, water, sewerage, phone cables, etc).

It is not only Kobe, but the whole Japan is a region prone to earthquakes due to the number of tectonic plates (*Philippines*, *Eurasian* and *Pacific* Plates) that converge below the country's surface (Figure 9.1a). To lighten the damage of earthquakes in the future, scientists are studying ways to predict the occurrence of quakes more accurately. One of the results of that study was to appoint the Chubu region (Figure 9.1b) to be a candidate for a large quake in the near future.

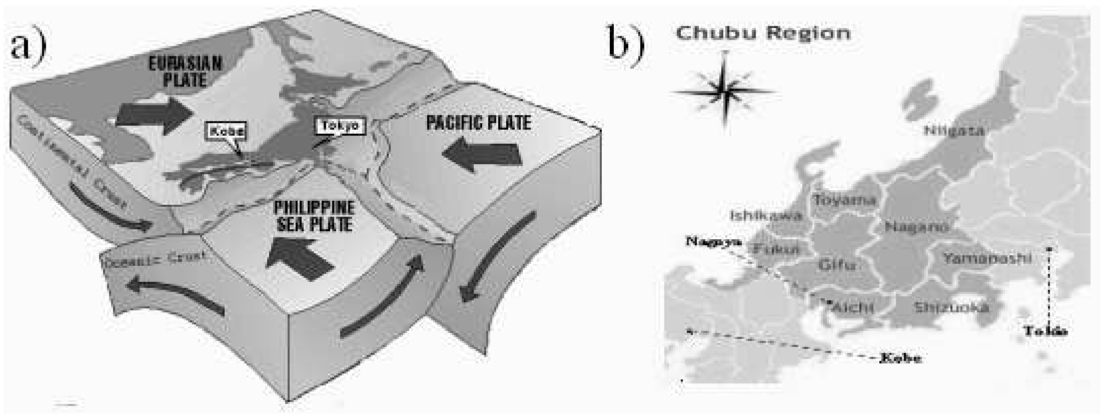


Figure 9.1: (a) Tectonic plates and (b) Chubu map (http://www.georesources.co.uk).

Based on this context, we have chosen the Kobe domain as a motivational scenario for our application. Such a domain represents a small but central part of Kobe, which will certainly be a problematic region in case of a new natural disaster. Similarly, such an application could be adapted to Nagoya, the principal city of the Chubu region.

9.2 The RoboCup Rescue Simulator

The *RoboCup Rescue* (RCR) simulator [Kitano and Tadokoro, 2001] is a real-time distributed simulation system that is built of several modules connected through a network via a central *kernel*, which manages communications among such modules. Each module can run on different computers as an independent program, so that the computational load of the simulation can be distributed across multiple processors. Each disaster phenomenon, such as collapse of buildings or fire spread, is simulated by a dedicated simulator-module, while a *Geographical Information System* (GIS) provides the initial condition of the disaster space.

The RCR simulator recognises six different types of agents: ambulance team, fire brigade, police force, ambulance centre, fire station and police office. Such agents act as several independent modules, which receive information about the environment and send commands to be performed by the kernel. In our experiments some of these agents are represented by I-X agents as detailed soon.

The simulation proceeds by repeating a specific cycle of one second (default value), which corresponds to one minute in the real disaster space. However such rate can be configured according to the application that we intend to test. During each cycle the following steps are carried out:

1. The kernel sends individual sensory information to each agent;
2. Each agent individually submits an action command to the kernel;
3. The kernel sends the action commands of agents to all sub-simulators;
4. Sub-simulators submit updated states of the disaster space to the kernel;
5. The kernel integrates the received states, sending the result to the viewer;
6. The kernel advances the simulation clock of the disaster space.

The RCR disaster space is not fixed so that we can model and use customised scenarios. For that end, there are appropriate graphical tools that generate geographical data in the format used by the RCR simulator. We have used one of these tools to create a scenario representing part of the Nagoya city, for example. Together with this capability of configuration, the principal advantage of using the RCR simulator is that it provides a dynamic and unpredicted environment, which enables the evaluation of our ideas in a more real way. A visual evolution of one of our experiments in the RCR environment is showed in Appendix D.

9.3 Application Architecture

A complete I-X coalition for the I-Kobe application could be defined as illustrated bellow (Figure 9.2). As introduced in the last section, the RoboCup Rescue simulator is the component that accounts for generating sensory information and processing the agents' commands. Sensory information from the simulator are indexed with the identification of agents and each of these agents only receives the perceptions associated with the scenario around it. In general, geographically fixed agents (e.g., Police Office) do not receive new information about the environment because they do not change their positions. Action commands (e.g., rescue, extinguish, move, etc.) are only generated by the movable agents such as fire brigades (FB).

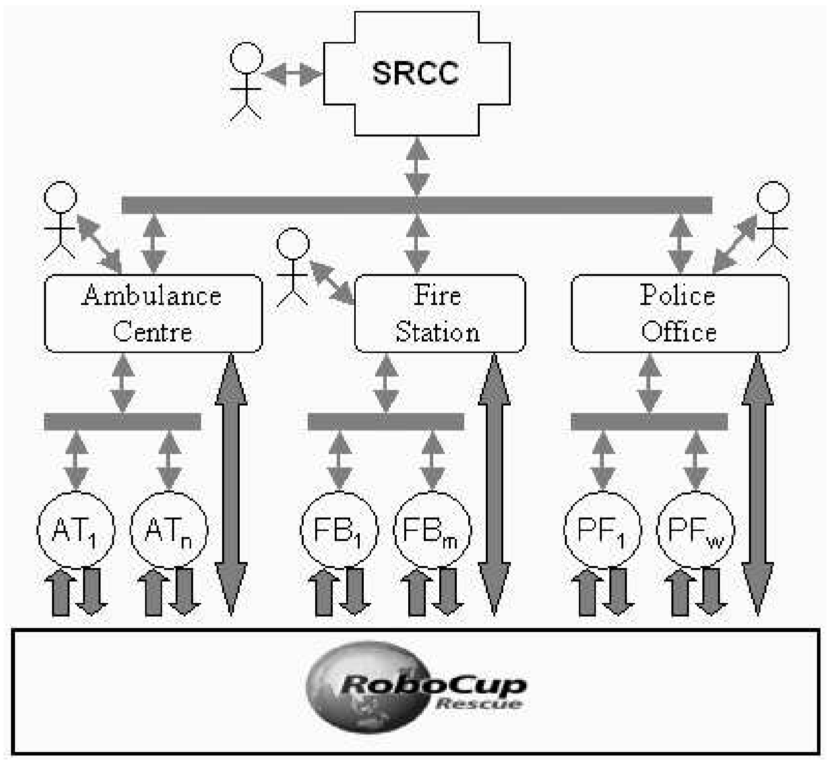


Figure 9.2: Application architecture.

The grey bars and arrows (Figure 9.2) represent the communication network, which defines which agents can interact. For example, fire brigades (FB_i) can only interact among themselves and with the fire station. For the experiments discussed here, the architecture uses a communication strategy in which each agent is mapped to a host and port number, and messages are sent by writing their serialisations to a socket. A thread that acts as a name-server on a specified port accounts for the role of distributing messages between agents. Note, however, that this architecture is independent of the communication strategy, so that other options could be used (e.g., *Jabber Technology* for XML messaging [Saint-Andre, 2001] or its recent formalisation XMPP[2]).

The coalition represented in Figure 9.2 is divided into the following levels of decision-making:

- **Strategic Level** - composed of one *Search and Rescue Command Centre* (SRCC) agent, represented by an I-X Process Panel. Its principal function is to coordinate the operational agents;

- **Operational Level** - composed of three agents called *Police Office*, *Ambulance Centre* and *Fire Station*, which are also represented by I-X Process Panels. Their main functions are, respectively, to coordinate the activities of police forces, ambulance teams and fire brigades;

- **Tactical Level** - composed of *n* ambulance teams (AT), whose function is to rescue buried civilians; *m* fire brigades (FB), whose function is to extinguish fires; and *w* police forces (PF), whose function is to clear roads. All these components are represented by I-X agents.

Agents of the operational and tactical levels have special functions to translate sensory information and actions between them and the simulator, because the internal formats used by agents and simulator are different. For example, consider the geographic information format. While the simulator uses the orthogonal distance (x,y) from a pre-defined point to a specific object, in millimetres, to indicate the position of such an object; I-X agents use latitude/longitude values for that. For this case, in particular, the functions to translate values between these two formats were implemented using the *regression technique*[3].

[2]*Extensible Messaging and Presence Protocol* (http://xmpp.org).

[3]Regression is the numerical technique of fitting a simple equation to real data points.

9.4 Evaluation and Results

This section describes a set of experiments, which demonstrate the implementation of the features discussed through this thesis. We must note that all the mechanisms developed in this section are based on constraint manipulation and visualisation of the effects of such manipulation. The first part (Section 9.4.1) focuses on mechanisms that support a better notion of collaboration. The second part (Section 9.4.2) discusses some practical directions associated with human-agent interaction mechanisms.

9.4.1 The Collaboration Aspect

The experiments of this section focus on the performance of a sub-coalition Θ_p composed of one police office (operational level) and ten police forces (tactical level) during a period of 150 cycles, which corresponds to 150 minutes in the real world. The objective of Θ_p is to clear the roads that are blocked by collapsed buildings. A good performance of Θ_p is very important to the fire brigades, for example, because they need clear paths to quickly reach the fire points and water refill places.

For the experiments detailed here, the tactical agents use a simple plan. Each police force has a list of blocked roads, indicated by the police office, that is ordered by the closest distance from the blockage to the current agent position. Then, if an agent is clearing a road, it remains doing that until one of the passable lines becomes clear. Otherwise, it accesses its list to know the next blocked road. If the list is empty, the agent tries to find (search action) other blockages around the scenario. This plan is used as a basis for the following experiments. The initialisation file for the experiments, defining numbers and initial features of agents and objects (refuges and fire points) is shown in Appendix E.

9.4.1.1 Experiment 1: Coalition without Coordination/Collaboration Models

This first experiment was carried out using RCR agents, which are provided by the simulator package and represent plain components that can be used as a basis for more complex implementations. The police force agents (tactical level) implement the plan previously discussed, while the police office has the function of receiving requests to clear roads from other operational agents and broadcasting such requests to its police forces. In other words, the police office ensures that all its subordinate agents have the same knowledge that it has about the roads. Table 9.1 shows the results[4] for this experiment.

Cycle	PF allocated time (%):			FB Allocated time (%):			Buildings
	Move	Search	Clear	Move	Refill	Exting.	on fire
10	26.7	52.2	21.1	87.0	2.0	11.0	7
20	49.5	24.7	25.8	71.6	1.1	27.4	10
30	55.2	16.2	28.6	70.3	0.7	29.0	11
40	61.0	12.0	27.0	67.9	5.6	26.4	14
50	66.7	9.6	23.7	60.2	13.9	25.9	16
60	70.3	8.0	21.7	54.7	14.9	25.3	16
70	72.6	6.8	20.4	52.6	18.3	29.1	15
80	75.9	5.9	18.1	50.2	20.9	28.9	18
90	78.5	5.3	16.2	47.0	22.7	30.3	22
100	80.4	4.8	14.8	44.6	24.8	30.6	21
110	82.1	4.3	13.6	43.8	25.0	31.2	23
120	83.6	4.0	12.4	42.4	25.8	31.8	25
130	84.9	3.6	11.5	41.7	26.4	31.9	30
140	86.0	3.4	10.6	40.4	27.6	32.0	34
150	86.9	3.2	9.9	39.6	28.0	32.4	34

Table 9.1: Simulation results for a coalition without collaborative/coordination models.

The table presents three types of information. First, the time that police forces (PF) have spent in their possible actions: move (when agents are moving to some position indicated by the police office), search (when agents are looking for blocked roads by themselves) and clear (when agents are clearing roads). Second the time that fire brigades (FB) have spent in their possible actions: move (when agents are moving along the scenario), refill (when agents are stopped in a refuge refilling their water

[4]Results on all tables in this section represent the joint performance of ten police forces during a simulation of 150 cycles. Each experiment was repeated three times.

tanks) and extinguish (when agents are trying to extinguish fires). Finally, the number of buildings on fire. According to this experiment, the coalition is not able to avoid the fire spread so that the number of buildings on fire increases during the simulation. Thus we need to understand what is the problem of this coalition and its approach.

The simulator classifies fires in three different levels: initial, medium and severe. At the beginning of the simulation (cycle 10), there are 7 buildings in 5 different locations that are in an initial level of fire. The idea is that the fire brigades quickly find the fire points while such fires are still at an initial stage because such fires are easier to be extinguished. However this is only possible if the paths along the route are clear.

Using the information in the table, we can observe that the police forces spend a great part of the simulation moving, rather than clearing roads. This mainly happens because agents in this experiment are self-coordinating their own activities according to the features of the environment. Thus, it is common that an agent moves to a position that must be cleared, however such a position was already cleared by another agent. In other words, many such move activities are a waste of time. Figure 9.3 illustrates this problem and its effects.

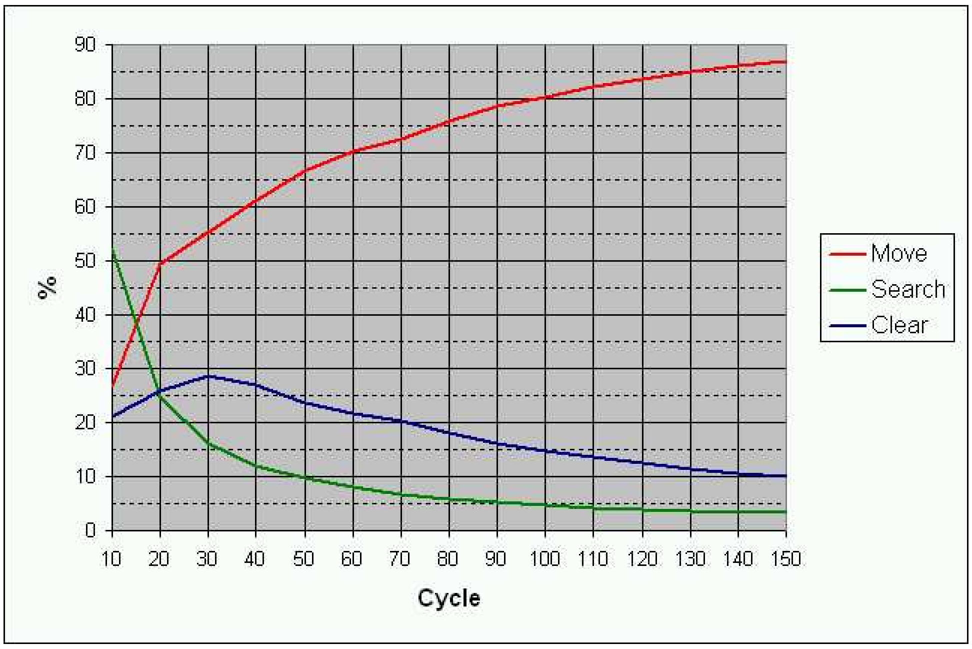


Figure 9.3: Activity allocation for police forces during the experiment 1.

In the graphic we can see that the *Move curve* is always increasing. This means that the police forces have requests to clear roads from the police office during all the simulation. In an efficient coalition we could expect that this rate starts to decrease at some moment. We can say that the sooner this rate decreases, the better the coalition performance is.

It is also important that the police forces quickly deal with incoming requests so that they start to search for blocked roads by themselves. Thus, differently of the *Move curve*, we could expect that the *Search curve* starts to increase at some moment. In the case of the current approach, the inefficient behaviour affects the performance of the fire brigades so that in the final stage of the simulation we have a high number of buildings (34 buildings, Table 9.1) on fire, together with other 12 buildings partially burnt (buildings where fires were extinguished).

Another analysis that we can carry out here is associated with the number of messages exchanged in the sub-coalition Θ_p. Consider that R_{clear} is the number of requests that a police office *po* receives from other operational agents. If we have a function SubordinateNumber(*po*) that returns the number of subordinates of *po*, the number of messages generated in Θ_p is given by:

$$\mathrm{N}_{messages} = R_{clear} \text{ x } \mathrm{SubordinateNumber}(po)$$

For our experiment, for example, we have 10 police forces (subordinates) and an average of 40 clearance requests. Thus, a total of 400 messages are generated in Θ_p.

9.4.1.2 Experiment 2: Adding Coordination via the I-X Approach

Based on the results of the first simulation, we have implemented a new coalition version that uses a model of coordination based on the I-X approach. For that, the police office was implemented as an I-P^2 and the police forces were implemented as I-X agents (partially representing the architecture in Figure 9.2).

I-X provides a coordination structure where each agent can report execution, completion or failure of activities. In addition, we can implement handlers to deal with the “clear activity” in particular. For this experiment we have implemented a handler called “SimpleAllocation” that uses the reports and information about the environment to generate an efficient delegation of activities to police forces. In other words, it intends to increase the number of clear actions during a period of time. This handler is based on three ideas:

- Closest distance: the handler tries to allocate the closest agent to a blocked road. If this is not possible, it tries the second closest agent and so on;
- Loading balancing: the handler firstly tries agents with least activities;
- Multiple allocations: the handler allocates more than one agent to each activity if there are agents with less executing activities than a constant Y.

Table 9.2 shows the results for this new simulation.

Cycle	PF allocated time (%):			FB Allocated time (%):			Buildings
	Move	Search	Clear	Move	Refill	Exting.	on fire
10	13.3	67.8	18.9	86.7	2.2	11.1	7
20	30.0	41.6	28.4	72.1	1.1	26.8	9
30	35.2	33.4	31.4	71.4	0.7	27.9	14
40	36.9	33.1	30.0	60.8	8.2	31.0	13
50	36.9	33.9	29.2	52.2	19.4	28.4	18
60	38.8	34.9	26.3	48.8	21.2	30.0	15
70	39.9	35.5	24.6	46.4	19.8	33.8	15
80	39.7	37.1	23.2	43.4	22.5	34.1	16
90	37.0	41.5	21.5	39.4	27.9	32.7	20
100	35.0	45.2	19.8	38.2	28.0	33.8	20
110	35.1	46.1	18.8	36.6	26.8	36.6	18
120	32.9	49.5	17.8	35.2	29.1	35.7	16
130	30.6	52.3	17.1	34.7	31.3	34.0	18
140	28.7	54.9	16.4	36.0	30.2	33.8	16
150	26.9	57.3	15.8	37.6	29.7	32.7	16

Table 9.2: Simulation results for a coalition using a coordination model.

According to Table 9.2, police forces are able to deal with their delegated activities faster than in the previous experiment. This fact is mainly represented by the rate of the move actions that increases until the cycle 80 and after that starts to decrease. The principal reason for this improvement is that the superior component (police office) is selecting specific (and best) agents to activities rather than sending all the activities to all agents. Thus the police office avoids that one police force moves to a place that another police force is already working on. Note in Table 9.2 that a better performance of the police forces also improves the performance of fire brigades. In the final round of the simulation (cycle 150) we have 16 buildings on fire (against 34 from experiment 1), together with 18 partially burnt (against 12 from experiment 1).

The idea of reports, supported by I-X, is fundamental in this process. If an agent does not report that it is executing an activity, the allocation handler considers that such an agent is still available. In the same way, agents must report the completion of activities so that the police office can update its allocation table. Agents can also report failures. In this case the activity must be reconsidered by the allocation process. The graphic in Figure 9.4 gives a better idea about the police forces' behaviour.

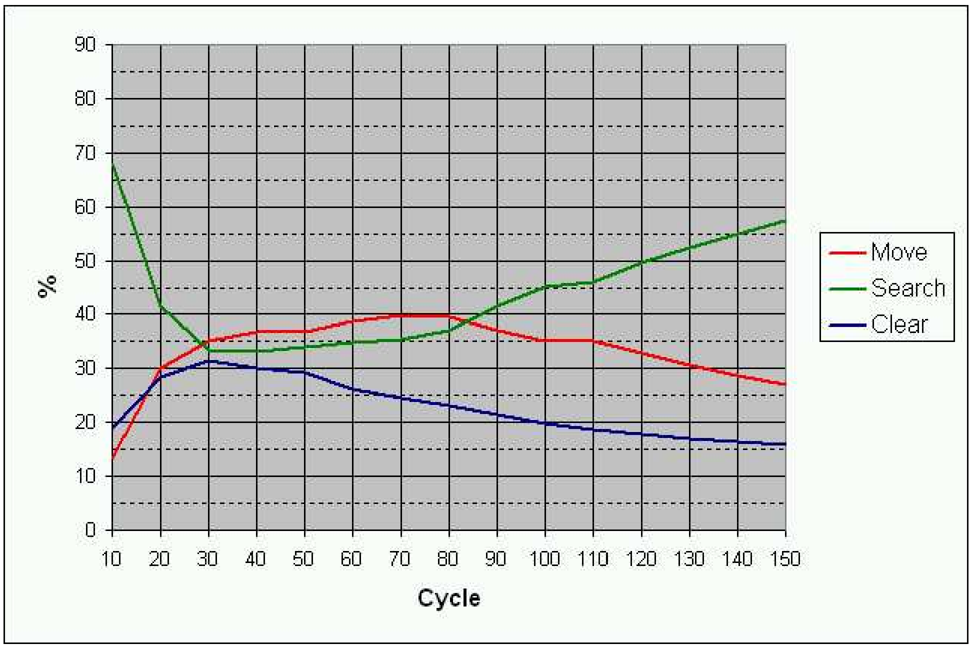


Figure 9.4: Activity allocation for police forces during the experiment 2.

The curves in this graphic represent the behaviour that we are expecting from the police forces. The *Move curve* has a peak around the cycle 70 and after that starts to decrease. The *Search curve* has the opposite behaviour, showing that the police forces have actually finished the delegated activities and they are going back to the search actions. Finally the *Clear curve* also demonstrates a better performance so that if we calculate the integral of this curve the resultant value will be bigger than the same curve in the previous experiment.

Despite the improvements presented by this coalition, it still has some limitations. The principal examples are:

- Police forces only report completion or failure of activities. Reports associated with activity commitments and progress are also important because they provide, for example, useful information to be used by the handlers;
- In situations where the police office allocates a clear activity to *n* agents, *n* subnodes are created to represent such allocations. These nodes are typically examples of or-activities where only one of them needs to be completed for the overall clear activity be finished. However this does not happen in this version.

Finally, considering that $\mathrm{N}_{subnodes}(a)$ returns the number of subnodes created to an activity a, we can define the number of messages in this coalition using the following expression:

$$\mathrm{N}_{messages} = 3 \text{ x } \sum_{i=1}^{R_{clear}} N_{subnodes}(activity_i)$$

The number 3 in the expression represents the three basic messages (delegation, execution start and completion/failure). For our experiment, in particular, we set $\mathrm{N}_{subnodes}$ to the constant value 2. Thus, every $activity_i$ has two subnodes (two police forces allocated to each activity). In this case we can simplify the expression to:

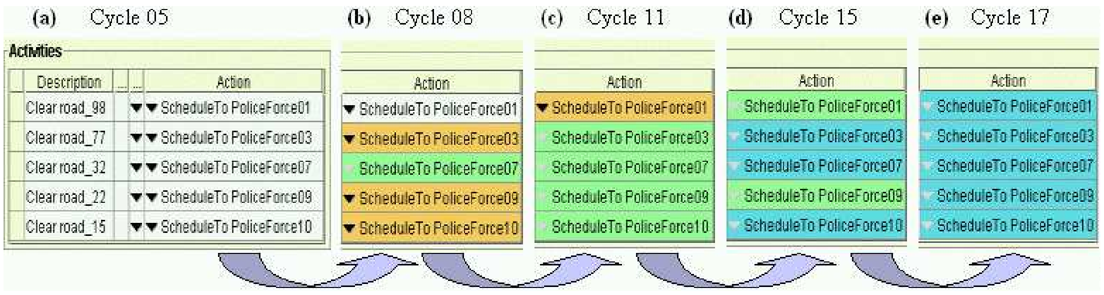
$$\mathrm{N}_{messages} = 6 \text{ x } R_{clear}$$

Then, considering again that the average of clear requests for this coalition is 40, we have a total of 240 messages changed by the coalition members.

9.4.1.3 Experiment 3: Adding Collaboration (Commitments and Mutual Support)

For this experiment we are still using an I-P^2 to implement the police office, and I-X agents to implement the police forces. However we have incorporated the notion of collaboration into such components via the commitment and mutual support ideas, discussed in Sections 7.2.3 to 7.2.5. It is important to stress that these mechanisms are incorporated as part of the I-X architecture, rather than as part of specific plans or processes. Thus, they intend to support a broad space of coalition planning problems, acting on abstract concepts of <I-N-C-A> plans such as activities and constraints.

Starting by the commitments idea, Figure 9.5 illustrates how such notion works in this experiment.



Description	Action (a)	Action (b)	Action (c)	Action (d)	Action (e)
Clear road_98	ScheduleTo PoliceForce01	ScheduleTo PoliceForce01	ScheduleTo PoliceForce01	ScheduleTo PoliceForce01	ScheduleTo PoliceForce01
Clear road_77	ScheduleTo PoliceForce03	ScheduleTo PoliceForce03	ScheduleTo PoliceForce03	ScheduleTo PoliceForce03	ScheduleTo PoliceForce03
Clear road_32	ScheduleTo PoliceForce07	ScheduleTo PoliceForce07	ScheduleTo PoliceForce07	ScheduleTo PoliceForce07	ScheduleTo PoliceForce07
Clear road_22	ScheduleTo PoliceForce09	ScheduleTo PoliceForce09	ScheduleTo PoliceForce09	ScheduleTo PoliceForce09	ScheduleTo PoliceForce09
Clear road_15	ScheduleTo PoliceForce10	ScheduleTo PoliceForce10	ScheduleTo PoliceForce10	ScheduleTo PoliceForce10	ScheduleTo PoliceForce10

Figure 9.5: I-P^2 using commitments and reports.

Figure 9.5a shows the initial stage where the allocation handler has assigned the task of clearing 5 different roads to 5 different agents. At this point no commitment has been taken (white colour). Figure 9.5b shows that 4 police forces have committed (orange colour) and one of them (PoliceForce07) already have started the execution (green colour). Figure 9.5c shows that the last agent (PoliceForce01) has committed and all the others have started their execution. Figure 9.5d shows that some agents have completed their tasks (blue colour), and the last figure shows that all agents have completed their activities successively. Note that while the handler improves the task division and load balancing, the commitment function provides an appropriate way to monitor the subordinates' performance.

In this example we do not have a failure. However consider that the activity "Clear road_77" has a temporal constraint Interval("Clear road_77",(09,14)). However the agent PoliceForce03 is not able to complete this activity, reporting a failure to the police office. In this case, the item “ScheduleTo PoliceForce03” in Figure 9.3d becomes red and, following the commitment/report function (Figure 7.8), the I-P^2 user or the I-P^2 itself must start a replanning process (e.g., delegate the task to more police forces so that it can be completed in less cycles). The police office will only report a failure to the activity’s original source if a replanning is not possible.

In the experiment 2, the first report is sent when agents start the execution of their activities. In this new version, the first report is generated as soon as a plan is created. Note that if there is a long period between the plan generation and the plan execution, the police office will also spend a long period unsure about the status of this activity.

This new version also compels police forces to send progress updates, if some plan information has changed. In this experiment, when a police force *pf* commits to the performance of an activity *ac*, it also sends the cost of *ac* to its police office. The cost here is given by the time, in cycles, that *pf* will spend to reach the blockage place, plus the time to clear such blockage. However this cost can change due to, for example, problems in the path and wrong estimations (e.g., *pf* usually does an estimation of the time to clear blockages in the moment of the commitment because it has not seen the blockage yet). As the allocator handler uses the cost values during the process of delegation, progress updates help it in keeping its allocation table in accordance with the real situation of the police forces, improving the process of allocation.

Together with the commitment mechanism, we have also introduced the notion of mutual support into this experiment. The mutual support function plays an useful role during the simulation. When a police force receives an activity to clear a road, it shares (according to Section 7.2.5) the conditions to clear this road. One of these conditions is that the road is actually blocked. If other agent of the coalition has an information that contrasts with this condition, it must inform to the police force.

This process indirectly resolves the problem of or-activities discussed in the last experiment. If two police forces pf_1 and pf_2 receive the same activity to clear a road and pf_1 finishes such activity before pf_2 has started its execution, the new status of the road (status road = clear) will contrast with the conditional constraint sent by pf_2 to pf_1, so that pf_1 informs this new status to pf_2 (note that both agents have received the conditional constraints of each other). Then, pf_2 automatically reports the completion of its activity to its police office. Using this mechanism, the police forces become available faster and the allocator has more options to perform its allocations.

On the other hand, this experiment highlights a potential problem. According to the mutual support function (Figure 7.9), before pf_1 informs of the new status of the road, it must try to create an activity that turns the condition true (status road = blocked). This does not happen in this experiment because police forces do not have this capability. But, in a general way, conditions that are negations of goals can generate problems, so that the CreateActivity function (line 05, Figure 7.9) must consider this exception[5].

The results of the practical use of such ideas are illustrated in Table 9.3.

Cycle	PF allocated time (%):			FB Allocated time (%):			Buildings
	Move	Search	Clear	Move	Refill	Exting.	on fire
10	21.1	61.1	17.8	86.7	2.2	11.1	7
20	41.0	31.6	27.4	71.6	1.0	27.4	10
30	47.9	23.8	28.3	69.0	0.7	30.3	8
40	49.5	20.3	30.2	62.8	7.2	30.0	11
50	49.6	21.6	28.8	55.3	16.9	27.8	10
60	47.3	25.8	26.9	52.2	17.1	30.7	8
70	41.9	33.6	24.5	49.6	18.4	32.0	7
80	36.6	40.1	23.3	46.1	23.5	30.4	6
90	32.5	46.3	21.2	42.9	24.9	32.2	6
100	29.5	50.5	20.0	45.3	23.3	31.4	4
110	27.8	53.6	18.6	45.4	24.5	30.1	5
120	25.6	57.2	17.2	44.6	26.9	28.5	5
130	23.6	60.1	16.3	43.7	29.3	27.0	5
140	21.9	62.7	15.4	41.9	31.8	26.3	5
150	20.5	65.0	14.5	40.2	35.0	24.8	5

Table 9.3: Simulation results for a coalition using a coordination/collaboration model.

According to the table, this coalition has a better performance as a whole so that

[5]This is a common problem in AI planning and planners often use "constraint types" to indicate which constraints are intended to be tested and which are intended to be achieved [Tate, 1995].

at the final round of the simulation (cycle 150) we have 5 buildings on fire (against 16 from the experiment 2), together with 14 partially burnt (against 18 from experiment 2). The behaviour of the police forces is better represented in the graphic below (Figure 9.6).

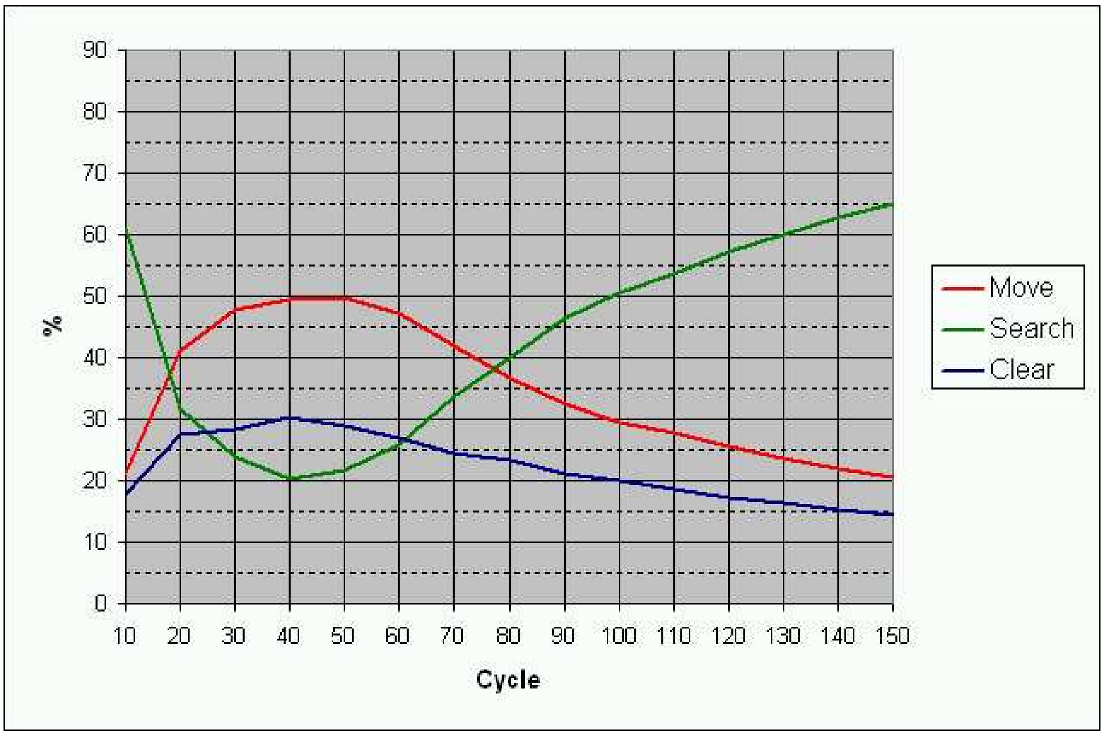


Figure 9.6: Activity allocation for police forces during the experiment 3.

The graphic shows that the *Clear curve* is almost the same as in the last experiment. However in this case we are sure that the clear actions are associated with the requests of the police office because such a curve follows the behaviour of the *Move curve*. In other words the police forces are moving to the blockages indicated by the police office. Note that there are two perspectives in which we can analyse the efficiency of Θ_p. From the Θ_p's perspective, such sub-coalition is efficient if it is able to clear a big number of roads. From the perspective of the coalition as a whole, which is the focus of this experiment, Θ_p is efficient if it is able to clear the necessary roads. Thus, rather than a quantitative result, we are interested in a qualitative measure on the performance of clear actions.

If we compare this graphic with the graphic of the experiment 2 (Figure 9.4), we can also notice that the *Move curve* and *Search curve* are more regular and narrower. This indicates that the police forces finish their delegated activities faster than the last experiment, returning to their original action of searching blockages by themselves.

The analysis of the number of messages changed in Θ_p for this case is more complex than for the last experiments. Considering $TSN = \sum_{i=1}^{R_{clear}} N_{subnodes}(activity_i)$ and NS the number of subordinates in Θ_p, the number of messages is given by the expression below:

$$\mathrm{N}_{messages} = 3 \text{ x } TSN + \sum_{i=1}^{TSN} f(subnode_i) + TSN \text{ x } (NS\text{-}1) + \sum_{j=1}^{NS} g(po_j)$$

The terms of this expression have the following meaning:

- 3 x TSN, represents the three basic messages generated by each subnode: the activity delegation, the commitment and the final message of completion or failure;
- $\sum_{i=1}^{TSN} f(subnode_i)$, represents the sum of all execution reports generated to each subnode$_i$. The abstract function f returns the value for each subnode, however such value cannot be precisely defined because it depends on the dynamic (variation degree of the initial state) of the system;
- TSN x (NS-1), represents the messages that an agent sends to its peers, containing the conditional constraints associated with each subnode;
- $\sum_{j=1}^{NS} g(po_j)$, represents the update messages related to the conditional constraints shared among the peers. The abstract function g returns the value for each peer, however such value cannot be precisely defined because it also depends on the dynamic of the system;

Note that we are considering an ideal case where all agents commit on the performance of the subnodes delegated. In cases where an agent does not commit, there are only two messages changed between it and its superior: the delegation and the non-commit messages. Applying the expression to the current experiment and setting the value of N$_{subnodes}$ to the same constant value 2, we obtains the following result:

$$N_{messages} = 3 \text{ x } 80 + \sum_{i=1}^{80} f(subnode_i) + 80 \text{ x } (10\text{-}1) + \sum_{j=1}^{10} g(x)_j$$
$$N_{messages} = 960 + \sum_{i=1}^{80} f(subnode_i) + \sum_{j=1}^{10} g(x)_j$$

According to the expression, Θ_p changes at least 960 messages. Considering that the values of the undefined terms ($\sum f$ and $\sum g$) do not have a great influence in this value (if this is not true, the plans developed by the agents are not robust or do not have a good quality), the principal problem is that each agent needs to receive conditional constraints of every subnode. Directions of this work could consider algorithms to filter the set of agents that really need to receive such kind of information. The big challenge is to create a general solution rather than one associated with a particular domain.

Finally, for this experiment we have designed both the commitment and mutual support ideas as I-X constraint managers. The commitment manager accounts for monitoring the commitment constraints, signalising the changes of activities' status and, consequently, when reports must be sent to. The mutual support manager accounts for monitoring the external conditional constraints and, according with the constraint processing, signalising the report of information or creation of new activities. Appendices F and G present the pseudocode in Java for these constraint managers, explaining how they interact with other modules of the I-X system.

9.4.1.4 Experiment 4: Avoiding Conflict via Mutual Support

This additional experiment is associated with the mutual support notion, however it is not related to the past experiments so that it uses a different context and sub-coalition to better express the problem of conflicts . The objective is to verify if our planning model is able to avoid conflict between activities. As commented in Section 2.2, there are two principal approaches (centralised and distributed) to avoid conflict during hierarchical planning. The mutual support function indirectly implements the distributed approach because each agent shares its conditional constraints, which other agents must considerer during their planning process.

The first step of this experiment is to identify a common scenario for potential activity conflict inside the RoboCup Rescue scenario. For that end consider the sub-coalition Θ_a composed of one ambulance centre and five ambulances. The objective of

Θ_a is to rescue injured civilians and move them to refuges. The scenario has a total of four refuges (R_a, R_b, R_c and R_d) and each of them can receive a maximum number of injured civilians[6]. A conflict of activities appears when two or more ambulances have planned to unload injured civilians in a refuge with only one place left.

Considering this scenario, the following activities (Figure 9.7) are commonly created during the planning process of ambulance agents. Note that if the current availability of R_b is 1, then the two activities will raise a conflict. This is more usual in the final part of the simulation. However, the mutual support function avoids this conflict because the constraint $\neg$ ((availability R_b),0), first created at the cycle 139, is sent to AmbulanceTeam2 (and all other agents of Θ_a). Thus such agent knows that its activity cannot be executed because it generates a state, represented by the effect constraints, that has collateral damage for the coalition.

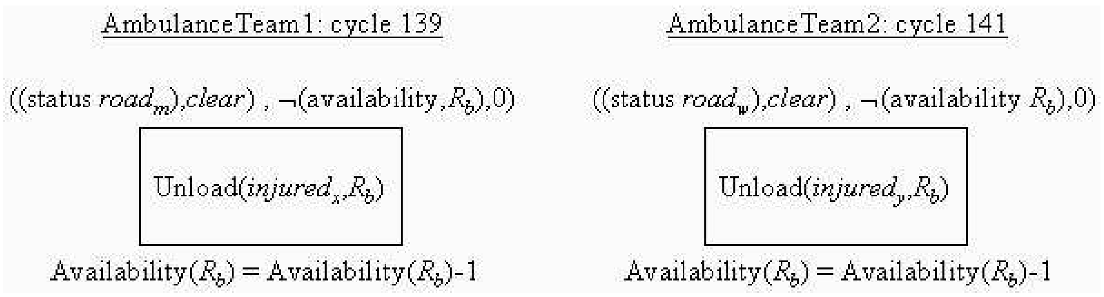

AmbulanceTeam1: cycle 139
((status $road_m$),$clear$) , $\neg$(availability,R_b),0)
Unload($injured_x$,R_b)
Availability(R_b) = Availability(R_b)-1
AmbulanceTeam2: cycle 141
((status $road_w$),$clear$) , $\neg$(availability R_b),0)
Unload($injured_y$,R_b)
Availability(R_b) = Availability(R_b)-1

Figure 9.7: Diagrammatic notation for the activity Unload(*?injured*), where conditions appear above the box, and the effects below.

However this approach for conflict resolution is dependent and limited by the sub-coalitions arrangement. In this example the approach works properly because all the agents involved in the conflict are in the same sub-coalition Θ_a. But consider the other constraint ((status $road_m$),clear). The ideal case is to send it to Θ_p (Section 9.4.1) because agents of Θ_p have the capacity of creating activities to keep this constraint true. This limitation could be avoided if all agents of the tactical level receive the broadcasted messages. However this solution increases the number of messages inside the coalition and the majority of messages will probably be out of the logical scope of

[6]This feature is not implemented in the current RCR simulator yet (Version 0.47), however it was discussed during the last RoboCup Symposium as one of the suggestions to the ongoing simulation development. Such a feature was motivated by the real limited capability of the health care centres.

each sub-coalition processing. Thus a better investigation must be considered on this approach so that we could decide the communication boundaries.

9.4.2 The Human Interaction Aspect

The second part of the experiments focus on two concepts associated with human-agent interaction: adjustable autonomy and generation of explanations. The idea is to show how such ideas can be implemented and integrated in the I-X architecture.

9.4.2.1 Experiment 5: Adjusting Autonomy

As discussed before, we have introduced a special kind of <I-N-C-A> constraint (autonomy) to define if an activity needs permission to be added in the plan. Thus we must also implement the I-X constraint manager that understands the meaning of such constraints. At this point it is important to remember the idea of an I-X constraint manager. In a practical way, the role of each constraint manager is to answer if a constraint does (or does not) hold. Then if the planner tries to add the activity a with a conditional constraint c into a plan p, the planner must firstly send c to a specific constraint manager. If such a constraint manager says that c holds, then a can be added into p.

Based on this explanation, the implementation of an autonomy constraint manager is simple. If the context defined by the parameters is true and the relation attribute is *permission*, then the constraint passes the decision to the sender-id value, generally its human user. For example, consider the constraint in Figure 9.8.

```
<constraint type="autonomy" relation="permission" sender-id="me">
    <parameters> <list>
        ((speed Wind),(>,20mph))
        ((water-tank FireBrigadeOne),(<,6000000mm)
    </list> </parameters>
</constraint>
```

Figure 9.8: Example of autonomy constraint (relaxed syntax representation).

This constraint means that if the wind speed is greater than 20mph and the amount of water in the FireBrigade1 tank is less than 6000000mm, then the activity associated with this constraint must ask permission to be performed by the agent. Note that such a constraint tries to configure an inappropriate scenario (low amount of water and violent wind) for fire brigade performance. Then the agent must pass the final decision to its human user, which can be done via a simple confirmation window. The solution is elegant because the constraint manager just needs to identify the scenario and pass its decision process to the constraint sender. However there is an inconvenient detail in terms of implementation. If we observe the two constraint parameters in Figure 9.8, the first is a world-state constraint, while the second is a resource constraint. Thus, the processes of dealing with such constraints were already implemented by the respective constraint managers, so that a considerable part of the code is repeated in the autonomy constraint manager. A possible solution is to modify the autonomy constraint syntax as showed in Figure 9.9.

```
<constraint type=“autonomy” relation=“permission” sender-id=“me”>
      <parameters> <list>
           <constraint type=“world-state” relation=“condition”>
                <parameters> <list>
                     ((speed Wind),(>,20mph))
                </list> </parameters>
           </constraint>
           <constraint type=“resource” relation=“condition”>
                <parameters> <list>
                     ((water-tank FireBrigadeOne),(<,6000000mm))
                </list> </parameters>
           </constraint>
      </list> </parameters>
</constraint>
```

Figure 9.9: Possible redefinition for autonomy constraints.

According to this redefinition, autonomy constraints allow constraints as parameters, so that the autonomy constraint manager should just ask other managers if the parameters of its constraints hold. Using such a definition we ensure that the autonomy constraint manager can respectively use the world-state and resource constraint managers to validate its constraint parameters. However note that we must keep the ontology simple and intelligible so that we ensure an easy development of processes to manipulate such an ontology. In this case (Figure 9.9) we are trying to modify the ontology to facilitate the implementation of a particular module. This is not good practise in terms of software engineering and a better investigation on the representation of scenarios must be done so that we are able to find the balance between a clear syntax and an easy implementation.

A second possible solution also involves a constraint redefinition, as illustrated below (Figure 9.10).

```
<constraint type="autonomy" relation="permission" type="world-state"
subtype="condition" sender-id="me">
      <parameters> <list>
            ((speed Wind),(>,20mph))
      </list> </parameters>
</constraint>

<constraint type="autonomy" relation="permission" type="resource"
subtype="consumable" sender-id="me">
      <parameters> <list>
            ((water-tank FireBrigadeOne),(<,6000000mm))
      </list> </parameters>
</constraint>
```

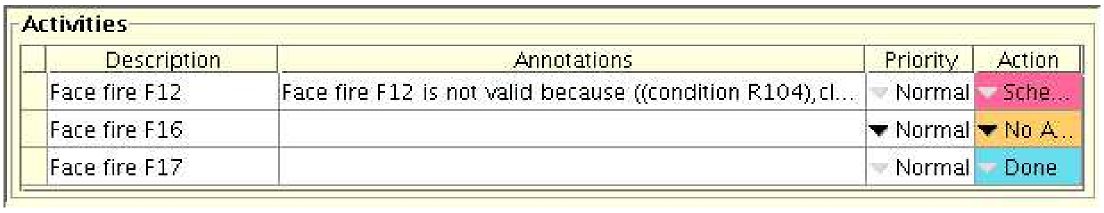
Figure 9.10: Another possible redefinition for autonomy constraints.

According to this redefinition, the autonomy constraint is wrapping other types of constraints. For that, two additional attributes, type and subtype, are needed to respectively receive the type and relation of the wrapped constraints. Then, similarly to the previous case, the autonomy constraint manager could just ask to other managers if the constrains are holding.

The inconvenient point of this option is that a scenario will be usually composed by several sparse constraints. If we want to define more that a scenario, where the autonomy constraint must be applied, we need to relate the constraints of each scenario. For example, we could add a scenario identifier attribute to the constraints. Note that we are still doing several changes in the general/original definition of constraints, so that this approach does not avoid the disadvantages of the previous redefinition.

9.4.2.2 Experiment 6: Creating Explanations

For this experiment we have focused on the generation of explanations to cases where activities become invalid. In other words, we try to show appropriate explanations to users about the motives that an activity cannot be included or executed. For example, consider that the I-P^2 below (Figure 9.11) represents the fire station component. One of its activities a_2 is "Face fire F_{12}", which is delegated to a fire brigade *fb*. Note that this activity is invalid (red colour) and there is an annotation[7] that, in this case, provides an explanation about this impossible status.


Activities

Description	Annotations	Priority	Action
Face fire F12	Face fire F12 is not valid because ((condition R104),cl...	Normal	Sche...
Face fire F16		Normal	No A...
Face fire F17		Normal	Done

Figure 9.11: Example explanation as an activity annotation.

A possible implementation based on our proposal and supported by the I-X architecture works in the following way. Suppose that when *fb* receives a_2, it decomposes such activity to:

- $a_{2.1}$: Refill *water-tank* in *refuge*;

[7]The annotation is: Face fire F_{12} is not valid because ((condition R_{104}),clear) is in conflict with ((condition R_{104}),blocked) set by world.

- $a_{2.2}$: Go from *refuge* to F_{12};
- $a_{2.3}$: Extinguish F_{12}.

Then, *fb* applies a path finder handler to $a_{2.2}$, which returns a solution with the following constraints: ((status R_{76}),clear), ((status R_{98}),clear) and ((status R_{104}),clear). However, R_{104} is blocked during the simulation and *fb* is not able to find another plan (route) for this activity. At this point the explanation must be generated.

We have implemented the explanation module as a special kind of handler that is automatically applied to every activity in I-P^2. Rather than decomposing or executing the activity, this handler just adds a listener to each activity so that the change of status is captured and, if such status becomes invalid, the handler starts the explanation process. After capturing the event, we can access the associated template and bind its variables. For the conflict template we must instantiate four variables: the activity identifier (*activity*), the activity constraint in conflict ($c_{activity}$), the current constraint that has triggered the conflict ($c_{trigger}$) and the source of such constraint (*source*). The instantiation of such variables is performed in the following way:

- *activity*: the activity identifier can be directly extracted from the event;
- $c_{activity}$: as discussed before, we are considering that constraint managers only return a *yes* or *no* to indicate that the constraint is (not) holding. In this way there is not a direct way to know which constraint was violated. However, using a loop we can retest all the activity constraints and know which one accounts for the problem;
- $c_{trigger}$: if we have the problematic constraint, we can use its pattern to ask the model manager for the current $c_{trigger}$ value ($c_{activity}$ and $c_{trigger}$ logically have the same pattern);
- *source*: the source of $c_{trigger}$ can be directly extracted from the definition of such a constraint (sender-id attribute).

Note that in this case we are using a default template. As discussed before, a more interesting explanation could be generated if we use specialised templates. In this case each handler should provide a method so that the explanation process could access it. The problem is to decide who will bind the template (the explanation process or the handler itself). The problems for each of the options are:

- Explanation process: in this case the process needs to know the meaning of each variable of different templates. For example, in the template " *activity* is not valid because $c.pattern_{activity}$ of $c.object_{activity}$ is $c.value_{trigger}$ rather than $c.value_{activity}$ according to current *source*", the process should know that $c.pattern_{activity}$ needs to be instantiated with the pattern of the activity constraint that is in conflict and so on;

- Each particular handler: in this case the handlers should access the information that they need. Certainly we will have a considerable repetition of code in each handler. However the most inconvenient problem is that we are transferring to the developers of the handlers the responsibility of implementing a function that must be provided by the architecture. Handlers must only account for dealing with an activity. Thus, this is a solution that is not in accordance with our proposal.

Again, as in the implementation of autonomy (Section 9.4.2.1), we can be forced to change or restrict some aspects of our theoretical model so that we can facilitate the implementation of such model. In this case, for example, a restriction could be set saying that every explanation associated with conflict uses the same four variables introduced by the default explanation.

Finally, the discussion of this section was focused on the generation of explanations for one case in particular: conflict and consequent invalidation of activities. However the implementation for other kinds of explanations follows the same steps, which are defined in the explanation function (Figure 7.13): capture the event, choose a template, bind its variables creating the explanation and associate such explanation to the activity (as an annotation). Note that in our initial example, the explanation was generated in the tactical level. However when the agent reports the failure of an activity, it can also return the explanation as an annotation of this activity. Thus, it is possible to visualise such explanation, as showed in Figure 9.11.

9.5 Summary

This chapter tries to stress the facility of thinking about practical aspects of a system if we consider all its issues on a unified perspective, in our case on a constraint-based framework. Then, starting from a planning architecture, we have integrated collaborative mechanisms using the same basis of constraint representation and manipulation. We must highlight two important concepts that support this process of integration: the activity handlers and the constraint managers. Both mechanisms have proved to be very practical ways to develop planning mechanisms to decompose/perform activities and validate constraints.

The search and rescue scenario associated with earthquakes was very appropriate to demonstrate the system features. Using such a domain we were able to create different and unpredicted situations. Initial experiments in such a domain have shown how the use of collaborative mechanisms improve the efficiency of the system as a whole. In fact the better results were already expected and other works have demonstrated that systems using a collaborative model have a better performance. The important point to be stressed here is the easy way to integrate collaboration via the development of new constraint managers and extensions in the representational model. Such an approach does not have an influence on the development of planning mechanisms (handlers), which are still independent of this collaborative framework.

It is also important to stress the role of the mutual support function. The idea of mutual support is not commonly found in other works, however we can notice that it brings several advantages such as better information sharing, conflict resolution and the basis for agents to perform activities to help other agents. However we still need to fix some matters. For example, if an agent abandons an activity, all the constraints associated with this activity that were shared inside the sub-coalition have to be cancelled.

Some initial experiments, associated with human-agent interaction (HAI), were also discussed in this chapter. The idea was to show that, as for the collaborative mechanisms, the HAI mechanisms can also be naturally integrated if we consider the same constraint-based framework. To that end we have developed simple mechanisms associated with adjustable autonomy and generation of explanations. The autonomy module is particularly interesting because it works as any other I-X constraint manager (returning yes or no), however its decisions are passed to users or other components depending on the current scenario. The generation of explanations is simple if we consider the use of default explanations. However some restrictions must be set if we intend to use specialised templates provided by different handlers.

A last implementational aspect that was not commented on here, and is likely to be one of our future directions, is associated with the implementation of preferences. The principal problem associated with this feature is the development of a general mechanism that enables the influence of preferences during the reasoning of different handlers, so that the implementation of such handlers is still independent of this mechanism.

Chapter 10

Potential Applications in Space-related Domains

The past chapters have used disaster relief scenarios as a basis for examples and an evaluation domain. However, the technology proposed here is not domain-restricted so that other domains can make use of it. This chapter demonstrates such a feature, addressing the potential use of our approach in two future space-related applications: the satellites' constellation and the human-agent Mars mission. Most of the information contained in this chapter resulted from discussions which took place at the *Fourth International Workshop on Planning and Scheduling for Space*. This chapter summarises such discussions in the following way: Section 10.1 brings up a review about the use of planning in the space domain, showing that the long-term agenda of the space agencies supports the efforts toward the definition of a collaborative human-agent planning framework. Section 10.2 introduces the motivations for the development of a system that supports the operation of satellites' constellations. Section 10.3 discusses the challenges of such systems, showing how the technology presented in this thesis could be used to face them. Section 10.4 introduces a futuristic but realistic scenario, whose specification is likely to be addressed in future space-related works. Finally, Section 10.5 concludes the chapter.

10.1 Planning in Space Domain

The use of intelligent planning in long-term space missions has mainly been focused on providing levels of autonomy to spacecraft (e.g., orbiters) and, more recently, rovers (e.g., the MER rovers). In fact, the use of a planner/scheduler (PS) in space is a very new experience if we consider that the first real application was in 1999, during the Remote Agent Experiment (RAX) [Muscettola et al., 1998], on board of the *Deep Space One* spacecraft. The purpose of RAX-PS [Jonsson et al., 2000] was to generate plans that could be executed on board to achieve specified high-level goals. Its principal differences from classical STRIPS planning were that: actions can occur concurrently and can have different durations, and goals can include time and maintenance conditions.

ASPEN, *Automated Scheduling and Planning Environment* [Rabideau et al., 1999], was intended to explore approaches complementary to RAX-PS. The principal focus was on classifying and repairing conflicts in the spacecraft models, which are described via the ASPEN Modelling Language (AML) [Smith et al., 1998]. The Continuous Activity Scheduling Planning Execution and Replanning (CASPER) system [Chien et al., 2000] is an evolution of ASPEN that integrates repair planning with execution. The idea is to continuously replan around updated information coming from execution monitoring. CASPER was used in *Earth Observing-1* [Chien et al., 2003], the first satellite in NASA's *New Millennium Program Earth Observing* series.

The beginning of rover missions to Mars created a new scenario for planning technology. This technology, however, is still based on earlier approaches. The OASIS system [Estlin et al., 2003], for example, uses CASPER as a planner in its proposal of mixing techniques from both machine learning and planning to rover control.

Efforts, as in the OASIS system, aim to provide more autonomy to rovers. *Sojourner* (the first rover to operate on Mars), for example, travelled about 100 meters during its 90-day lifetime [Mishkin et al., 1998]. However the Mars Exploration Rovers (*Spirit* and *Opportunity*) were designed to travel up to 100 meters per day. Autonomy is important because the rovers have intermittent and delayed communication with Earth. In fact the time of travel of a radio signal from Mars to Earth (about 10 minutes) precludes any real-time idea of continuous human operator control. Furthermore, capacity issues of the *Deep Space Network* (DSN), an international network of antennas that supports interplanetary spacecraft missions, and planetary dynamics (position and rotation of the planets) also impose restrictions on communication.

According to [Ball et al., 2002], the risk to human health during missions beyond Earth orbit (as exposure to high levels of radiation) is the greatest challenge to human exploration of deep space. Despite this threat, human interplanetary missions have seriously been discussed and some dates were already presented[1] as suitable to such missions.

The beginning of human interplanetary missions may lighten the communication delay problems. In this new scenario, astronauts will carry out joint experiments with robots on planetary surfaces, so that several high-level goals and decisions could be taken into the work environment rather than made on Earth. This scenario will bring new requirements regarding joint human-agent planning [Allen and Ferguson, 2002, Sierhuis et al., 2003], which differ from the current approaches to space missions.

Figure 10.1 illustrates this evolution of the planning technology for space applications. At the moment, planning has been applied to provide autonomy to spacecraft such as *Deep Space One* and *Earth Observing-1*. The technology is also being extended to rovers, but until now only in an experimental way.

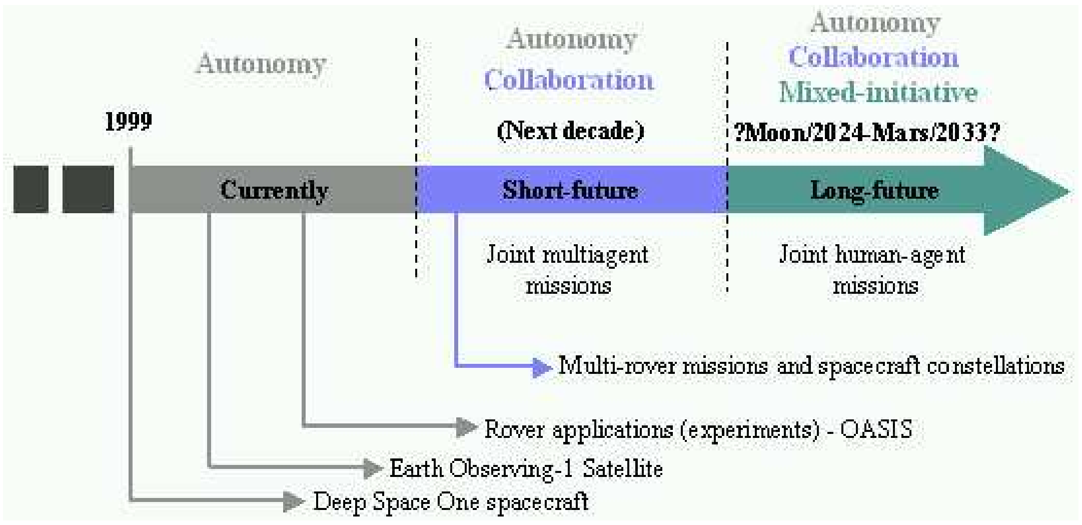

Autonomy
Autonomy
Collaboration
Autonomy
Collaboration
Mixed-initiative
1999
(Next decade)
?Moon/2024-Mars/2033?
Currently
Short-future
Long-future
Joint multiagent missions
Joint human-agent missions
Multi-rover missions and spacecraft constellations
Rover applications (experiments) - OASIS
Earth Observing-1 Satellite
Deep Space One spacecraft

Figure 10.1: Evolution of planning technology in space applications.

[1]In the XVI Congress of the Association of Space Explorers in 2000, for example, 6-May-2018 was presented as a suitable date for the first human mission to Mars due to the planets' orbits.

In the near-future we will not only need autonomy, but also notions of collaboration to support multi-rover missions (e.g., MISUS project [Estlin, 2004]) and spacecraft constellations, which are the focus of this chapter. Then, the next step will be the involvement of humans (astronauts) in the planning process to support missions such as that specified in the *Aurora* schedule, which dates a human mission to the Moon in 2024 and to Mars in 2033.

10.2 The Satellites' Constellation Domain

The satellites' constellation domain (Figure 10.2) consists of a group of low-Earth orbiting satellites that have cross-link communication capability, each carrying nearly the same suite of instruments. Each satellite receives high-level goals from ground station operators, or other satellites. Then, it will perform its own planning by decomposing a goal into a set of sub-goals to be achieved with its onboard subsystems (satellites' resources) and in cooperation with other satellites.

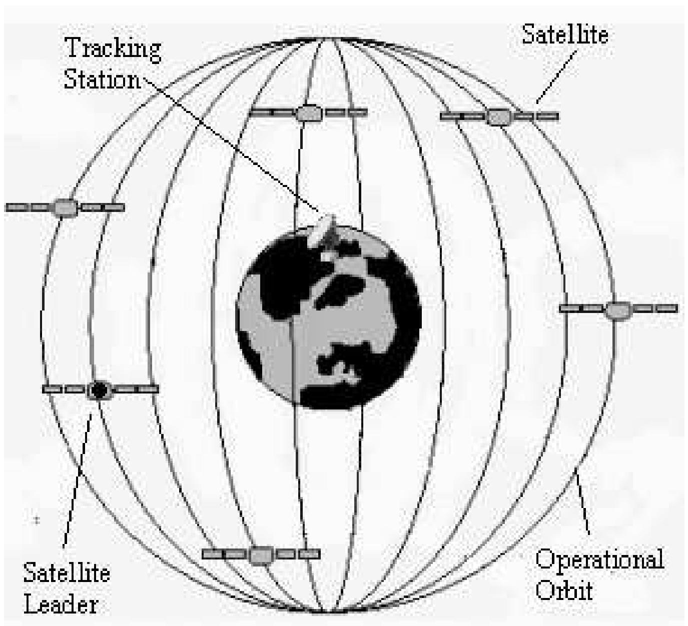


Figure 10.2: Perspective of a constellation.

The onboard resources of satellites are: *Ozone Mapping Spectrometer* (OMS), *Microwave Scanning Radiometer* (MRS), *Moderate Resolution Imaging Spectrora-*

diometer (MRIS), *Hyperspectral Imager* (HI) and *Advanced Microwave Sounding Unit* (AMSU). A request from the ground station operators is defined as a goal that requires use of one or more these resources, together with associated data memory storage, containing start/stop times and operational parameters specified here as constraints.

We have used the idea of hierarchical coalitions to specify a possible configuration for this domain, based on the *Europe Space Agency* (ESA) resources. In order, the strategic level is represented by the *European Space Operations Centre* (ESOC), localised in Darmstadt (Germany), whose function is in accordance with the idea of providing directions and analysis of information to missions. The tactical level is represented by the five ESA tracking stations (Kourou-French Guiana, Maspalomas-Canary Island, Perth-Australia, Redu-Belgium and Villafranca-Spain) and their respective satellite leaders, which control a group of six satellites and whose principal function is to decompose and schedule the given goals. Note that for a large constellation, it is not efficient to command satellites individually. In this case it is more efficient to only send commands to the satellite leader and it either parses the command string and forward the commands to the appropriate satellites, or makes some intelligent decision by itself. Finally the satellites compose the tactical level, accounting for task execution and returning of results.

A wide variety of missions can be better implemented with satellites working together to meet a single objective. Reasons cited for using constellations include lower mission cost, the need for coordinated science and special coverage of survey requirements. For example storms and other phenomena observed from multiple angles can be used to generate 3-D views; or a cluster of satellites flying in formation and working together can form a virtual lens hundreds of miles across.

However the most widely used application of satellite constellations is for reasons of extended area coverage. Low earth orbiting constellations such as *Globalstar* use dozens of satellites to provide continuous global or near-global coverage. The GPS system uses a constellation to provide global coverage and spatial diversity of the multiple satellites in view. Earth mapping missions can use multiple satellites to shorten the time between successive observations of the same area. NASA is planning missions of up to 100 satellites for magnetosphere research with orbits nearly as distant as the moon. A single-satellite approach would require many years of data collection to match what the constellation can survey in a short amount of time [Verfaillie et al., 2003].

10.3 Issues and Solutions

The first issue to the development of systems for satellites constellations (henceforth SC) control is the number of satellites that could compose the constellation. The domain described in the last section is represented by 30 satellites divided in 6 subgroups. However, future missions will tend to use more satellites so that the idea of scalability must be considered. According to previous works in this area [Richards et al., 2001], the management role of the Earth operation centre could be simplified by a hierarchical organisation, since the operation centre needs to interact with a few entities in order to direct the activities of several satellites. This affirmation is in accordance with our work, which was developed to support such a kind of organisation. However, there is an important feature that has not been discussed yet. Architectures for SC are likely to employ a *dynamic* hierarchical organisation to combine the efficiency of hierarchical activity delegation with the robustness afforded by dynamic reorganisation.

The idea behind dynamic organisations is that they are able to reorganise themselves to adapt to changes and minimise, for example, the impact caused by the introduction of new members or the loss of old members due to failures. To include dynamic organisations in our approach, we should extend it so that the organisation model could be represented via constraints, and agents could be able to reason on relations and their features.

A second issue is associated with the resource and temporal reasoning process. In fact, the idea of resources is very important because decisions of employing satellites must consider the availability of onboard resources (e.g., memory, power, downlink accesses and bandwidth) as well as the satellites' current status (e.g., viewing geometry, reference trajectory, sensor states, etc.). For example, in low Earth orbit, micro-satellites are in view of a ground station for approximately four minutes at a time, about four times a day [Mohammed, 2001]. If we consider 35 Gigabytes the total amount of data generated during a 2 minutes observation, and that such data could be continuously downlinked at the bandwidth of 150 Mbps, this process will take about 32 minutes. However, with the previous restriction (16 minutes a day), this process can take two days to downlink the data from a short two-minutes observation session.

We can note that resources such as onboard memory, downlink accesses and bandwidth capabilities, are elements that in fact limit the number of experiments that can be performed, so that they should be explicitly represented as constraints in the planning process. Our model supports this idea, providing a specific kind of constraint to represent the satellites' resources and their features. Thus, specific resource constraint manager modules can be implemented to reason on such constraints according to the requisites of the application.

Similarly to resources, time is also fundamental to SC. For example, an image of a certain region of Earth can only be taken within a certain time frame while a satellite is over that location. In a general way, satellites whose missions involve observations and/or communication with the ground must take into account access windows, which are intervals of opportunity defined by when the satellite is in view of the target on the ground. This case can be easily specified using our model. For example, to restrict the time of an observation we could use: interval("Observe region ?x",(?start,?finish)).

However, for SC the problem is more complex because the relative position of individual satellites must be taken into account when assessing viewing geometry. In other words, all the satellites need to be in a specific position during a specific time. Our model also supports this case via the temporal relations. For example, we could use the temporal relation constraints *start* and *finish* to indicate that the satellites must start and finish the observation activity at the same time. Similarly, we could define more complex cases using other temporal constraints.

Note that if a satellite leader is trying to form a specific 6-size viewing geometry, it must delegate activities and receive back commitments of the participants so that it can confirm the experiment. Approaches in SC control system are already considering this issue, so that knowledge bases of satellites are being developed to store information and to support reasoning about commitments of other satellites [Richards et al., 2001].

Together with commitments we must also stress the importance of reports. Consider the situation where all satellites commit on the observation activity. However when one of them reaches the pre-defined position, it notes that its vision is blocked by clouds. In this case, such a satellite must report this fact so that the leader decides what must be done, such as changing (replanning) or cancelling the previous commitments.

The idea of mutual support can also be used in this domain to generalise the concept of collaboration. Consider the following example: a satellite *sat* moves out of the range of a ground station for a specified time period, such that it can lose important messages. Using our proposal, if the acquisition of such messages is declared as conditional constraints to the performance of *sat*'s plan, the system will account for informing this fact to other satellites so that the required messages may be relayed through the system using other agents in the constellation to still allow messages to reach their destination. Note that we do not need to define specific processes to treat this case.

A last issue is associated with the autonomy of SC control systems. In general, important uses of autonomy in SC are: to enable that satellites fly within specified tolerance levels; avoid collisions; address fault detection, isolation and resolution; share knowledge, and plan and schedule activities [Zetocha et al., 2000]. Autonomy entails a higher level of risk, so that safeguards must be put in place to ensure that no adverse conditions arise. Note that our proposal also considers this problem and provide mechanisms for human users to restrict scenarios where autonomous components are allowed to take decisions by themselves, or ask permission to other components. Furthermore, the use of preference constraints could also be important in such situations as a way to reflect the preferences of users in the autonomous reasoning of satellites.

In brief, the principal idea of this section is to show that the SC domain presents several opportunities to apply the technology developed during this thesis. Another interesting advantage of the SC domain is that it can be used to validate collaborative planning and schedule approaches for space, being a first step for the development of more challenging applications as the scenario discussed in the next section. A short introduction to the SC research area and current efforts, such as the *TechSat 21 Program*, is presented in [Zetocha et al., 2000].

10.4 Future Space Scenario

Section 10.2 has described a possible application that mainly covers the second phase of the planning evolution to space applications (Figure 10.1). This section shows, via a descriptive example, how we can apply the technology to cover the third phase of such evolution, where human users are actively involved during the process of planning and execution [Siebra et al., 2004].

For that end, we exemplify the features of I-P^2 via a fictitious one-sol (one Martian day) mission. The duration of daylight is itself a constraint to the mission, because some rovers have devices that only work with solar energy (e.g., APXS spectrometer). The mission is realised during the summer in the Southern hemisphere on Mars because of higher temperatures. However this temperature may change significantly due to perihelic dust-storm activity, requiring continual replanning.

Figure 10.3 shows the mission scenario displayed via two I-X viewers. At the left hand-side the I-X Map Viewer uses a JPG image as the surface, however the viewer enables the plug-in of a PDS (Planetary Data System) layer to manipulate real surface data. In the I-X 3D Viewer (right-hand side), objects are modelled via VRML (Virtual Reality Modelling Language) and I-P^2 imports them using the Java 3D API.

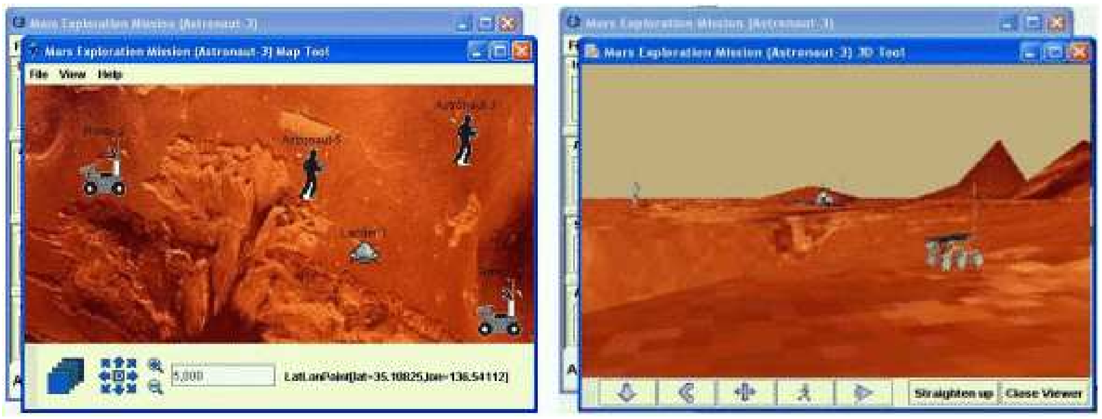

Mars Exploration Mission (Astronaut-3) Map Tool
File View Help
5,000
LatLonPoint[lat=-35.10825,lon=136.54112]
Mars Exploration Mission (Astronaut-3) 3D Tool
Straighten up
Close Viewer

Figure 10.3: Futuristic scenario for human mission displayed via two I-X viewers.

The mission has several decision-making levels. The ground team on Earth sets the macro goals, sharing the activities with the Mars-Habitats. Each Mars-Habitat has one or more exploration teams, which are composed of a lander, two rovers (r_1 and r_2) and two astronauts (a_1 and a_2). Orbiters provide some auxiliary services. In our example there are two principal objectives for each exploration team: studying the surface of Mars (activity assigned to rovers) and looking for some sign of life (activity assigned to astronauts). The lander provides a higher bandwidth communication channel (e.g., for high-quality images transmission) and a mobile micro-laboratory.

Supposing an exploration team has a sol-mission set to start at 06:00h and to finish at 18:00h. During this period, information on the current state of the environment can be passed to I-X Process Panels via "world state" constraints. These might come from sensors directly, or from some analysis or reporting system. Additional I-X viewers (Figure 10.3) can display the world state in a more natural way to astronauts, and also work as a data input mechanism.

At 06:00h the team members begin their work together with their I-X Process panels, which represent what they need to do (activities), directions of how to do these (sub-activities), along with constraints on those activities. Issues will appear during the performance of the activities. For example, at 10:00h the temperature quickly starts to decrease, so that the exploration activities of a_1 and a_2 are paused. This fact also generates a new issue: "What should we do about the fall in temperature?". Thus the I-X agent, using its own capability or external services, returns possible actions to deal with this new issue.

At 12:00h the Mars-Habitat notices a failure in rover r_2. Then it sets a new activity with highest priority to astronaut a_1: "Fix r_2". For that, a_1 makes use of the library of plans, which contains standard operating procedures about how to discover and fix rover problems. This new activity delays a_1. However Mars-Habitat knows this fact because a_1, via its I-P^2, is always sending reports about its performance. In this way, Mars-Habitat is able to predict global failures and replan new schedules.

At 14:30h r_1 finds a possible sign of life during its soil analysis, whose investigation is not part of its activities. However, as the rovers implement the idea of mutual support, a_1 and a_2 are informed about this information when r_1 adds it into its current knowledge as a world state constraint. In this case, the location of the event could determine which astronaut/panel will create a new activity ("examine event"), in accordance with the area assigned to the astronauts.

At 16:05h the orbiter reports a strong solar activity, which increases the emission of solar energetic particles. Thus all outdoor human activities are cancelled and a_1 and a_2 are not able to complete all their activities. In this case the Mars Habitat team, supported by its I-P^2, must provide a new plan that involves the remaining activities of the day. If this plan is not possible, the team reports the situation to the Earth team.

10.5 Summary

This chapter has presented a potential domain where we can apply the technology discussed through this thesis, demonstrating in this way its generality. A particular advantage of this approach for space domains is that it considers the requirements of current missions, but also provides the required support for joint human-agent activities in future interplanetary missions. In addition, the open style of architecture that has been adopted enables the use/integration of our approach with current space-related technologies. For example, a specific schedule already in use for satellites could be adapted to be used as an activity handler.

There are several other domains where we could use this collaborative human-agent planning approach. The main examples are military applications, which are naturally in accordance with this approach because they already consider hierarchical coalitions as organisational option.

Finally, we can note that the basic work in creating a new domain is associated with the definition of new operations to decompose plans/activities, and maybe additional activity handlers together with constraint managers, which deal with particular reasoning related to the domain. The remaining processes such as the making of commitments, information sharing and conflicts associated with default constraints (temporal, resource, etc.) are automatically provided by the architecture.

Chapter 11

Final Remarks

This final chapter starts by tracing a comparative discussion of our approach with systems that could also be used to support coalition operations. Then, we highlight important points discussed through this thesis, such as its contributions, limitations and problems. We also discuss possible future directions, which have the principal objective of investigating and overpassing the limitations of our approach.

11.1 Related Projects and Comparative Discussion

Several systems have been developed with similar objectives to this thesis: supporting the planning and execution performance of joint groups. This section summarises the main ideas of some of these systems, comparing such ideas with our approach.

11.1.1 Coalition Agents eXperiment (CoAX)

As already discussed in Chapter 1, the CoAX Project [Allsopp et al., 2002] demonstrates the use of the agent-based paradigm as a way to deal with the technical issue of integrating different technologies in a coalition organisation. In other words, heterogeneous components are seen as a set of distributed agents that are able to share understanding and information among themselves.

The principal proposal of this project is to use agents to wrap different systems, enabling their integration via a common infrastructure. Differently, we have used the concept of ontology to integrate components. According to this approach, every external component that needs to be integrated to the system must respect ontological commitments when receiving and sending information to the coalition. For example, the RoboCup Rescue Simulator uses a specific method provided by the I-X architecture to send information about the disaster space to the coalition. However, first of all it needs to translate its spatial representation (x and y orthogonal points) to the I-X representation (latitude and longitude). Another example is when we use external systems as handlers. As discussed in Chapter 10, rather than developing a new scheduler for space applications using the I-X approach, we could use the CASPER scheduler that already involves considerable know-how in this area. In this case, we also need a bidirectional translation between the two computing environments.

11.1.2 CplanT System

CplanT [Pechoucek et al., 2002], a multiagent system that belongs to the area of *war avoidance* operations, developed a formal knowledge based approach to the coalition formation problem. The principal issue in this approach is that agents may agree to collaborate, but they are often reluctant to share their knowledge and resources. Thus, negotiation mechanisms are necessary to support the various levels of collaboration.

In our approach we do not discuss any kind of negotiation process associated with information sharing. In fact, at the current stage, agents must share any kind of information that should be important to the performance of other coalition members. Negotiation is very suitable at pre-operation moments, when a coalition is involved in discussions related to which role will be played by sub-coalitions or members. However we are aware that in more complex kinds of coalitions, such as multinational coalitions, the idea of classifying and restricting the information access is appropriated and must be considered.

11.1.3 DIPART System

DIPART [Pollack, 1996], acronym for “Distributed, Interactive Planner’s Assistant for Real-time Transportation planning”, is a prototype simulation system that includes a network of agents, each of which assists a human user. The internal architecture of each DIPART agent is based on a generic model of process scheduling, where incoming messages are stored on a queue, indicating events that may require attention. The architecture provides a component (*Locus of Meta-Level Control* - LMC) that determines which processes should be invoked in response to each message, scheduling these selected processes.

DIPART and I-P^2 agents are based on similar ideas. While users can submit goals to the agents and receive current status information, agents are responsible for generating, dispatching and monitoring the execution of a plan to carry out those goals. In this context, we could say that DIPART messages, processes and LMC are respectively similar to <I-N-C-A> activity messages, activity handlers and constraint managers. However DIPART has been mostly concerned with the investigation and experiment of scheduling algorithms, which could be applied by the LMC. In other words, the LMC is a real scheduler. Differently, constraint managers do not take decisions, but provide information about the constraints that they are managing and respond to actions' requests about the possibility of including activities into the agent's plan.

11.1.4 Coordination of distributed Activities (CODA) System

CODA [Myers et al., 2001] is a system that proposes to improve the coordination process using targeted information dissemination among distributed human planners. According to the CODA approach, each planner declares interest in different kinds of plan changes that could impact his/her local plan development. Thus, CODA is based on plan authoring tools, which are able to monitor the activities of users so that changes that match awareness are forwarded automatically to the person who declared interest in them. An interesting feature of CODA is related to the selectivity of information. Because local planners exactly declare which they are interested in, there is a balance between sharing too little and too much information. This information is sent in real-time so that plans can be updated in a timely manner to ensure adequate time for users to consider impact on their local plans and develop appropriate repairs.

Our approach implements a similar idea, however without the need of users to declare which information they want. This information is directly extracted from the conditional constraints of each activity and analysed by the mutual support function. In the same way that happens in CODA, such a function provides an adequate way to share information, supporting conflict detection and resolution.

11.1.5 DSIPE

DSIPE [DesJardins and Wolverton, 1999] is a distributed planning system that provides decision support to human planners in a joint planning environment. DSIPE uses a hierarchical model where each agent is a separate DSIPE planner instance that supports a human user. The top agent of the hierarchy accounts for the development of high-level plans and distribution of sub-goals among lower-level agents.

DSIPE uses the same hierarchical structure for agents that we are using, however with a different approach to plan decomposition. In DSIPE each planning agent has a complete representation of its own subplan as well as a partial representation of the subplans being developed by other planning agents, with explicit dependencies and relationships with its local subplan. Thus the project implements a specific algorithm for information sharing where each agent knows the kind of information that could be important to other agents so that they can update their partial representations. In brief, the idea is to automatically filter the information that is sent from one planning agent to another by eliminating the information that are provably irrelevant to the latter agent's decision-making process.

The DSIPE approach certainly increases the system complexity because the filter algorithm needs to have specific information about other agents and their activities. However such algorithm could be interesting for us. Note that our proposal for mutual support is based on a broadcast of conditional constraints to a sub-coalition. Probably, some agents from this sub-coalition will not use these constraints in their process. Thus we could use a filter algorithm to change the broadcast for a peer-to-peer process.

11.1.6 Anticipatory Planning Support System (APSS)

APSS [Hill et al., 2000] is a proposal of decision support system that seeks to merge planning and execution, and replaces reaction to events with anticipation of events.

For that, rather than choosing a single course of action (COA) and following it to conclusion, the system maintains many possible COAs so that the plan is considered to be a tree. Nodes of this tree represent states and decision points in the plan. The branches represent the transition to a new state based on a particular action. As new branches are developed, the system will continue planning along those branches. Thus, anticipatory planning for a branch can be done well in advance, rather than reactive planning once the branch occurs.

The point stressed by the APSS project is the importance of the plan information collection to quickly confirm or deny the viability of branches. In our approach we do not try to create and maintain a tree with several branches, however we also consider fundamental the use of information to anticipate possible failures. Such an idea is mainly implemented via reports on execution progress, which try to capture the information that could support the reasoning of superior agents in detecting and resolving possible failures in a sub-coalition plan.

11.1.7 Advisable Planning (AP) System

The aim of the AP system [Myers, 1996] is to make planning technology more accessible to users via the metaphor of *advice-taking*. This approach is close to the ideas presented by the O-Plan and TRAINS systems. Such works support interactions on the part of humans, however AP proposes interactions between users and the planning process at a higher level. Basically, advice is modelled as task-specific filters on the set of solutions to a specific planning problem. The AP system does not require advice for its operation. Instead, advice simply influences the set of solutions that the planning process could provide for a task.

We can note that the concept of advice is very similar with the idea of preferences discussed in this thesis. Both approaches try to set recommendations that must be initially considered by the planning process. However the AP system does a more in deep investigation of this problem, identifying kinds of influences required by users of planning systems. Such influences are: *Task Advice* - designates specific goals to be achieved and actions to be performed; *Strategic Advice* - consists of recommendations on how goals and actions are to be accomplished, and; *Evaluational Advice* -

encompasses constraints on some metric defined for the overall plan, such as resource usage, execution time and solution quality. This study could be very useful for guiding a redefinition of our preferences model in future works.

11.2 Contributions

The principal contribution of this thesis is the specification of a computational framework that supports the implementation of important requirements to the development of hierarchical coalition support systems. Such requirements were collected from three different research areas (multiagent planning, collaborative systems and human-agent interaction) so that the definition of such requirements and posterior identification of theoretical solutions are themselves also contributions of this thesis.

The contribution associated with the framework specification can be divided into two parts. First, the extension and adaptation of a constraint-based ontology (<I-N-C-A>). Second, the definition of a set of functions to manipulate the elements of this ontology. The advantage of our approach is that the implementation of all requirements can be understood from the same perspective because all of them can be implemented on the same basis. In other words, the implementation of the requirements was based on the same principles: constraints representation and manipulation. In a sequential order, we can list the relevant set of contributions of this thesis as:

- Formalisation of hierarchical coalitions, showing that this kind of organisational structure brings several advantages for the development of collaborative processes;

- Discussion about the design of hierarchical coalition support systems, stressing the several types of requirements associated with joint human-agent planning and appropriated theoretical solutions for them;

- Proposal of a unified platform, underlined by an existent constraint-based ontology, which was extended and adapted to support the implementation of the pre-defined requirements;

- Specification of collaborative functions that provide agents with a general model of teamwork, enabling that such agents reason by themselves about their responsibilities as members of a coalition. In this context we could highlight the mutual support function that is a simple way to support several capacities such as conflict detection, information sharing and joint activity support;

- Proposals on how to integrate important mechanisms associated with human-agent interaction (adjustable autonomy, generation of explanations, user control and control on users) using the same constraint-based framework;

- Use of several of these ideas in a practical experiment, which demonstrates how they can be implemented using the I-X Architecture and mainly the concepts of constraint managers and activity handlers. Potential scenarios based on space applications were also discussed, showing that our proposal is not a domain specific technology;

One of the practical advantages of I-X is that its concepts are being implemented as a Java API so that coalition systems' developers can use all the advantages of object-oriented programming. Furthermore, the I-X architecture is domain independent so that several different domains can be configured without changes in the functions or tools. The users' role during the preparation of systems is restricted to describe domains and possible standard operating procedures using the <I-N-C-A> ontology. The inclusion of new activity handlers or constraint managers is optional and depends on the necessity of additional abilities required by the application.

11.3 Limitations and Problems

The use of our approach in practical experiments was also important to highlight some of its problems. A considerable part of such problems are derived from processes associated with the sharing and maintenance of *distributed knowledge*. In a more specific way, these problems are associated with our approach for mutual support. Furthermore, our approach still needs a deeper investigation in some issues associated with human-agent interaction mechanisms, mainly when we consider their practical implementation. The next sections discuss such aspects.

11.3.1 Distributed Knowledge

An interesting deficiency in the current planning literature is the lack of discussions about the amount and kind of knowledge that each agent of a coalition must maintain about the coalition's activities so that they mutually support one another. According to our approach to mutual support, we are arguing that the knowledge about the conditions required by each agent is appropriate to enable such support. However, based on our experiments, it is not possible to demonstrate the real efficiency of such an approach. A detailed investigation of this issue could be done via measures of the usefulness and usability of the knowledge, in our case the activities' conditions shared through the coalition.

Other practical matters associated with the mutual support approach are the elimination of useless information, the number of messages and the post-conflict decision process. As discussed previously, the solution applied to eliminate conditions is to stamp a timeline in each constraint saying the period that it should be considered valid. However such a solution was not very useful in our application because the majority of the activities did not have a defined timeline (start and finish times). Another pertinent problem appears when an agent abandons an activity. In this case its conditional constraints are no longer valid, but as they were shared into the coalition, they are still generating unnecessary reasoning and performance of activities. Thus, the development of a process like a *coalition garbage collection*, applied to unnecessary constraints, could be appropriate to avoid collateral effects.

Concerning the number of messages, the experiments have demonstrated that the mutual support function is likely to require considerable communication. The idea of filter algorithms, as discussed in DSIPE (Section 11.1.5), could be applied to this problem, avoiding that an agent sends its conditional constraints for all agents of its (sub-) coalition.

The *post-conflict decision process* is another possible reason for low efficiency. Consider the following scenario: an agent a_1 generates a plan p_1 with a conditional constraint c_1, which is shared into the coalition. Meanwhile, an agent a_2 is trying to generate a plan p_2, however their possible plans are in conflict with c_1. According to the simple post-conflict decision process that we are applying, all the agents must consider the constraints already shared. Thus, a_2 will not be able to complete its activity. This problem becomes worse if the performance of a_2 is critical to the coalition aim. In this case a_1 should replan its activities, eliminating c_1 and enabling the generation of p_2 by a_2. We can conclude that the simple use of time is not adequate for the post-conflict decision process and the priority of the activities is a very important attribute that must be considered during such process.

11.3.2 Human-Agent Interaction Mechanisms

The principal problems associated with HAI were raised when we tried to implement the theoretical solutions. The implementation of the adjustable autonomy module, for example, could have a better solution to the treatment of scenarios. Currently we need to reuse a considerable part of other constraint managers to define the autonomy constraint manager. The module of explanation also requires a better implementational investigation if we intend to use specialised templates provided by different handlers.

However, the principal HAI lack in our approach is related to the preferences module. The concept of preferences is very important and suitable to be implemented in systems that intend to provide some kind of autonomy. The simple definition of *weak-constraints* (constraints that can be eliminated in case of conflict) could be a simple solution for preferences. However we argue that the restriction of possible values to plan variables could increase the number of preference styles defined by the model.

The evaluation and test of preferences' models require a considerable change in the planning process. Note that if there is a preference type of constraint, then, for this to be useful, we must also implement a constraint manager that understands the meaning of such constraints. However a "no" answer from this constraint manager does not invalidate the activity. Furthermore this constraint should also work as a *consultant component* to the variable binder. Thus the relation of such a constraint manager, having these features, with other components of the system is not clear for us yet.

11.4 Future Directions

The definition of possible extensions for this thesis is directly indicated by the problems and limitations summarised in the last section. Such extensions are listed below:

- Development of experiments that measure the usefulness and usability of conditional constraints, considering the process of mutual support. The idea is to investigate, from the set of all constraints received by an agent, which of such constraints are useful for the different processes provided by the mutual support approach (conflict resolution, information sharing and activity generation). Based on the results of this experiment, we could also be able to know which information, other than conditional constraints, is important to agents. If we apply such experiments to all the hierarchical levels, we can produce the basis for supplying the lack in the current planning literature discussed in Section 11.3.1;

- Study and implementation of mechanisms that enable the elimination of knowledge which is no longer valid from the coalition. Rather than agents exchanging messages saying which information must be eliminated, agents should be able to reason about such elimination by themselves. An interesting metaphor, used in the previous section, is to think about this process as a garbage collection used for some object-oriented languages. In Java, for example, each virtual machine uses a specific rule (there are no longer any references to an object) to eliminate unnecessary objects. In the same way, we could implement some rule in each agent so that they eliminate unnecessary knowledge;

- Specification and test of a post-conflict decision process so that it considers the idea of priority. In fact the <I-N-C-A> ontology already provides a representational attribute for priority in the activity definition. Thus we could use this attribute to decide which agent must replan in case of conflict. Note that a typical solution in this case is the use of negotiation. However, as discussed in Chapter 2, we are avoiding this kind of interaction between agents;

- Investigation of better implementational solutions for both modules of adjustable autonomy and explanation generation, looking for a balance between representational and implementational complexities;

- Complete study about the impact of the use of preferences in the I-X architecture and if our solution is really adequate to be implemented in such architecture.

Independently of our future directions, the idea is to still follow the implementational basis of the I-X approach during the investigation and development of new features. As proposed by I-X, all the functionalities must be provided as an Java API so that the coalition support systems' developers can choose which modules and features their applications will use.

Another possible direction of this work could consider its extension to applications whose features are incompatible with our current approach. Note that our approach assumes some restrictions on the coalition domains that intend to use it. The principal restrictions are:

- Coalition members must present fixed roles in the hierarchical structure because the approach does not provide ways to a self-reconfiguration of roles and positions in the hierarchy;
- The domains must consider the process of coordination as a simple delegation of activities from superior to subordinate members. The feedback from subordinates is just a commitment or non-commitment to such activities;
- Coalition members must present a form of reasoning based on an explicit goal-based reasoning. This means that coalition components must plan sequences of activities or make decisions on sets of possible actions to reach pre-specified goals.

The first restriction indicates that our approach is not compatible with domains that require dynamic and autonomous change of roles. An example is the satellite constellation domain, discussed in Section 10.3. In this domain it is important that, in case of failure of a critical satellite, the most adequate satellite of the coalition substitutes the place and functions of such a critical satellite. Note that this issue is very important in hierarchical structures because each superior member is a critical component and its failure can set apart all the sub-coalition from which it is a top component.

The second restriction indicates that our approach is not compatible with domains that require negotiation during the process of activity delegation. For example, in a multinational military coalition, countries could join a runtime process of negotiation to decide which areas they are going to perform in. To provide negotiation at that level, we should extend the commitment function so that it also considers, for example, ideas of a negotiation protocol [Conry et al., 1988]. Such an extension could define several types of commitment constraints (Figure 7.7) so that such commitments can be associated with negotiable conditions.

The last requirement indicates that our proposal does not support a direct implementation of reactive behaviour. According to our three-levels hierarchical structure (Figure 2.3), we can note that the approach considers reaction as one of the principal features of the tactical level. However the closer mechanism to reactive behaviour that we have provided is the definition of SOPs. Probably such pre-planned sequences of activities are not enough to cope with the dynamic of domains such as the *RoboCup Soccer* [RoboCup, 2006].

11.5 Conclusion

During his invited talk entitled “Adventures in Artificial Intelligence” and proffered in the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03), Nils Nilsson (IJCAI-03 award for Research Excellence) discussed the evolution of AI over the past 40 years. According to Nilsson, one big problem of the current AI research is its continuing fragmentation/specialisation into sub-areas, which leads investigations and new proposals to deal with very specific problems. Thus the original aim and effort of AI, which could be summarised as to create intelligent machines or extend the cognitive human abilities, is losing force. Nilsson also used his talk to motivate young researchers so that they try to integrate some of these specific works and create more general intelligent systems.

The work developed during the elaboration of this thesis was in part influenced by this talk. During our initial study in coalition support systems, we have noticed that existent solutions could be individually applied to resolve particular problems. However, such solutions may not be compatible with other solutions already in use by ongoing systems. Based on this fact, this thesis focused on the investigation of a unique basis on which we could unify and implement several conceptual solutions, which are very suitable to the development of coalition support systems.

The study of coalition support systems is itself very motivating because such systems are in fact able to extend the abilities of a team to perform tasks. Furthermore there are several applications, from disaster relief to space domains, which can take advantage of such systems.

As we can notice from the discussion in Section 11.1, proposed systems that can be used to support coalitions focus their proposals on specific problems. In a different way, we try to present a broader approach that considers several important requirements for such systems, however not going so deep in all of them. In this way, the principal contribution of our approach is to offer a unique basis for integrating different requirements. Anchored in this basis, we build up specific solutions for problems such as resource representation, mutual support, explanation generation and so on.

The choice of a constraint-based ontology as the basis for our approach was very appropriate, providing a good support for the implementation of the three main groups of requirements: multiagent planning, collaboration and HAI. Associated with planning, constraints have interesting properties (Section 3.2), some of them complementary to the abilities of HTN planning that we are exploring. Several projects have been using constraints as a basis for their planning approaches, demonstrating that the union of planning and constraints is already a promising and well-known approach.

In the research field of collaboration we did not find a clear study about the modelling of collaborative concepts based on constraints. None of the projects discussed in Section 11.1 ever directly discusses the role of collaboration, in a general way, in their analysis. However we demonstrate that such an approach can be easily done if we consider commitments as a kind of constraint on the planning process, reports as resultant events of the constraint processing, and mutual support as a process based on the knowledge that one agent has about the conditional constraints of other agents.

Finally, constraints are also suitable to HAI. As previously discussed, constraints are declarative so that human users only need to specify what relationships must hold without specifying a computational procedure to enforce those relationships.

The practical idealisation of our project, considering disaster relief domains, is the development of an *International Rescue Support Site* (Figure 11.1) whose basic idea is to provide ways for the stabilisation of search and rescue coalitions in moments just after disasters are identified. In this context, the site could be a repository of I-X agents and domain libraries that, together, could provide the power of avant-garde computer technology in favour of human life.

Figure 11.1: Logo of the I-Rescue web page (http://i-rescue.org).

Appendix A

<I-N-C-A> Specification for Plans

```
PLAN ::=
    <plan>
        <plan-variable-declarations>
            <list>PLAN-VARIABLE-DECLARATION</list>
        </plan-variable-declarations>
        <plan-issues>
            <list>PLAN-ISSUE</list>
        </plan-issues>
        <plan-issue-refinements>
            <list>PLAN-ISSUE-REFINEMENT</list>
        </plan-issue-refinements>
        <plan-nodes>
            <list>PLAN-NODE</list>
        </plan-nodes>
        <plan-node-refinements>
            <list>PLAN-NODE-REFINEMENT</list>
        </plan-node-refinements>
        <constraints>
            <list>CONSTRAINER</list>
        </constraints>
        <annotations>
            <map>MAP-ENTRY</map>
        </annotations>
    </plan>

MAP-ENTRY ::=
    <map-entry>
        <key>OBJECT</key>
        <value>OBJECT</value>
    </map-entry>

PLAN-VARIABLE-DECLARATION ::=
    <plan-variable-declaration id="NAME" name="SYMBOL" scope="VARIABLE-SCOPE">
        <annotations><map>MAP-ENTRY</map></annotations>
    </plan-variable-declaration>

VARIABLE-SCOPE ::= local | global

PLAN-ISSUE ::=
    <plan-issue id="NAME" expansion="NAME">
        <issue>ISSUE</issue>
        <annotations><map>MAP-ENTRY</map></annotations>
    </plan-issue>

ISSUE ::=
    <issue status="STATUS" priority="PRIORITY" sender-id="NAME" ref="NAME"
    report-back=YES-NO>
        <pattern><list>PATTERN</list></pattern>
        <annotations><map>MAP-ENTRY</map></annotations>
    </issue>
```

STATUS ::= blank | complete | executing| possible | impossible | n/a

PRIORITY ::= lowest | low | normal | high | highest

YES-NO ::= yes | no

CONSTRAINER ::= CONSTRAINT | ORDERING

```
PLAN-ISSUE-REFINEMENT ::=
    <plan-issue-refinement id="NAME" expands="NAME">
        <plan-variable-declarations>
            <list>PLAN-VARIABLE-DECLARATION</list>
        </plan-variable-declarations>
        <plan-issues>
            <list>PLAN-ISSUE</list>
        </plan-issues>
        <annotations><map>MAP-ENTRY</map></annotations>
    </plan-issue-refinement>
```

```
PLAN-NODE ::=
    <plan-node id="NAME" expansion="NAME">
        <activity>ACTIVITY</activity>
        <annotations><map>MAP-ENTRY</map></annotations>
    </plan-node>
```

```
ACTIVITY ::=
    <activity status="STATUS" priority="PRIORITY" sender-id="NAME" ref="NAME"
    report-back=YES-NO>
        <pattern><list>PATTERN</list></pattern>
        <annotations><map>MAP-ENTRY</map></annotations>
    </activity>
```

```
PLAN-NODE-REFINEMENT ::=
    <plan-node-refinement id="NAME" expands="NAME">
        <plan-variable-declarations>
            <list>PLAN-VARIABLE-DECLARATION</list>
        </plan-variable-declarations>
        <plan-nodes>
            <list>PLAN-NODE</list>
        </plan-nodes>
        <constraints><list>CONSTRAINER</list></constraints>
        <annotations><map>MAP-ENTRY</map></annotations>
    </plan-node-refinement>

PATTERN-ASSIGNMENT ::=
    <pattern-assignment>
        <pattern><list>PATTERN</list></pattern>
        <value>OBJECT</value>
    </pattern-assignment>

CONSTRAINT ::=
    <constraint type="SYMBOL" relation="SYMBOL" sender-id="NAME">
        <parameters><list>PARAMETER</list></parameters>
        <annotations><map>MAP-ENTRY</map></annotations>
    </constraint>

CONSTRAINT ::= KNOWN-CONSTRAINT

KNOWN-CONSTRAINT ::=
    <constraint type="world-state" relation="condition">
        <parameters><list>PATTERN-ASSIGNMENT</list></parameters>
    </constraint>
```

```
KNOWN-CONSTRAINT ::=
    <constraint type="world-state" relation="effect">
        <parameters><list>PATTERN-ASSIGNMENT</list></parameters>
    </constraint>

ORDERING ::=
    <ordering>
        <from>NODE-END-REF</from>
        <to>NODE-END-REF</to>
        <annotations><map>MAP-ENTRY</map></annotations>
    </ordering>

NODE-END-REF ::=
    <node-end-ref end="END" node="NAME">
    </node-end-ref>

END ::= begin | end
```

A.1 Observations

1. OBJECT: objects are instances of I-X (e.g., ACTIVITY, ISSUE, etc.) or atomic (float, integer, string, etc.) classes;

2. NAME and SYMBOL: they essentially represent short and relatively simple strings. Symbols correspond to the symbol type, familiar to Lisp and Prolog programmers;

3. PATTERN: patterns are lists of objects. For example, "size of car" could be a 3-elements list representing a pattern;

4. PARAMETER: parameters are an open kind of element that will be defined according to the constraint to be created.

Appendix B

<I-N-C-A> Proposed Extensions

```
PLAN ::=
    <plan>
        ...
        <constraints>
            <list>CONSTRAINT</list>
        </constraints>
        ...
    </plan>

CONSTRAINT ::= KNOWN-CONSTRAINT

KNOWN-CONSTRAINT ::=
    <constraint type="temporal" relation="TEMP-TYPE" sender-id="NAME">
        <parameters><list>TEMPORAL-PA</list></parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>

KNOWN-CONSTRAINT ::=
    <constraint type="resource" relation="RESOURCE-TYPE" sender-id="NAME">
        <parameters><list>PATTERN-ASSIGNMENT</list></parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>

KNOWN-CONSTRAINT ::=
    <constraint type="commitment" relation="COMMIT-TYPE" sender-id="NAME">
        <parameters> <list>PATTERN-ASSIGNMENT</list> </parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>

KNOWN-CONSTRAINT ::=
    <constraint type="autonomy" relation="DEGREE" sender-id="NAME">
        <parameters><list>PATTERN-ASSIGNMENT</list></parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>

KNOWN-CONSTRAINT ::=
    <constraint type="preference" relation="PREF-TYPE" sender-id="NAME">
        <parameters><list>PREFERENCE-PA</list></parameters>
        <annotations> <map>MAP-ENTRY</map> </annotations>
    </constraint>

TEMP-TYPE ::= interval | TEMPORAL-RELATION

RESOURCE-TYPE ::= reusable | consumable

COMMIT-TYPE ::= nil

PREF-TYPE ::= world-state | temporal | resource | instantiation

DEGREE ::= permission | in-control
```

```
TEMPORAL-RELATION ::= before| equal | meets | during | overlaps | finishes

TEMPORAL-PA ::= TEMPORAL-INTERVAL-PA |
                TEMPORAL-RELATION-PA

TEMPORAL-INTERVAL-PA ::=
    <temporal-interval-pa>
        <pattern><list>
          <name>NODE-ID</name>
        </list></pattern>
        <value>INTERVAL</value>
    </temporal-interval-pa>

TEMPORAL-RELATION-PA ::=
    <temporal-relation-pa>
        <pattern> <list>
          <name>NODE-ID</name>
          <name>NODE-ID</name>
        </list></pattern>
    </temporal-relation-pa>

PREFERENCE-PA ::= PATTERN-ASSIGNMENT | INSTANTIATION-PA

INSTANTIATION-PA ::=
    <preference-pa>
        <pattern><list>PATTERN</list></pattern>
        <variable>VARIABLE-ID</variable>
        <value>OBJECT</value>
    </preference-pa>
```

```
INTERVAL ::=
    <interval>
        <numeric>initial-time</numeric>
        <numeric>final-time</numeric>
    </interval>
```

B.1 Observations

1. Note that this extended proposal has eliminated the CONSTRAINER element (see Appendix A). This element is necessary to represent ordering as a type of constraint. However we have also eliminated the ORDERING element because its role can be played by the "before" temporal relation;

2. The sender-id attribute has an important role of classifying the constraints in internal and external. If such attribute is not specified, the default value is internal. Source of internal constraints are the agent itself and its user;

3. The NODE-ID element must be filled by one id attribute value of the PLAN-NODE element (see Appendix A);

4. The VARIABLE-ID element must be filled by one id attribute value of the PLAN-VARIABLE-DECLARATION element (see Appendix A);

5. The COMMIT-TYPE is not used in this current version, so that its value is nil. However we are considering support to a possible expansion, which could provide types of commitments such as temporal (e.g., I am committed to do the activity *x* during the interval *i*), or conditional (e.g., I will commit to do the activity *y* if I receive the correct payment).

Appendix C

Search Patterns as Standard Operating Procedures

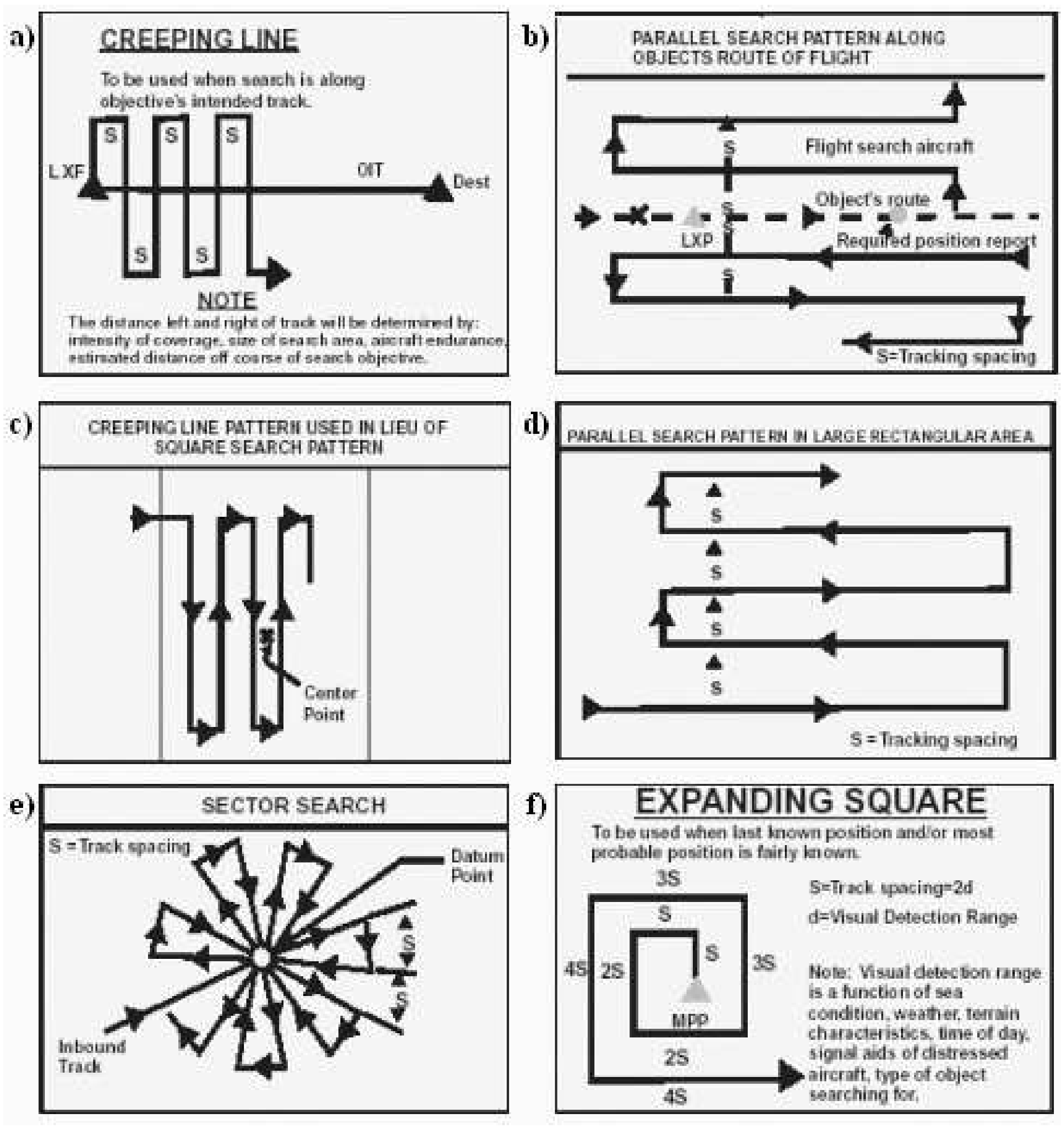
Search Patterns (SPs) are suggestions on routes that can be used during activities of searching. According to the features of environment and current goals, there is a more appropriate SP that can be applied to define a route to perform this search.

An interesting way to incorporate this concept in our approach is to implement each search pattern as a SOP so that it can be applied as an action to specific activities. For example, the general format for such activities could be:

Search *?object-to-be found* in *?region-for-searching*

Then, any SOP that matches this activity format is provided by the system as an action. The specification of SPs as SOPs also involves the definition of variables, whose function is to configure parameters of the route. For example, consider the search pattern illustrated in Figure C.1d. For this SP there is a variable *S* (Tracking spacing) that must be bound before the performance of such SP. Thus, a value for this variable, and for any other more, must be specified in the system.

Another option to incorporate SPs in the system is to provide only one action (activity handler), called "Apply search pattern", that should decide by itself which SP is more appropriate to be carried out. However, note that in this case the action needs to present a more complex reasoning method to be able to take this decision.



To be used when search is along objective's intended track.

The distance left and right of track will be determined by: intensity of coverage, size of search area, aircraft endurance, estimated distance off course of search objective.

To be used when last known position and/or most probable position is fairly known.

Note: Visual detection range is a function of sea condition, weather, terrain characteristics, time of day, signal aids of distressed aircraft, type of object searching for.

Figure C.1: Six different search patterns (from [Joint Chiefs of Staff, 1996]).

Appendix D

Disaster Evolution in RCR Simulator

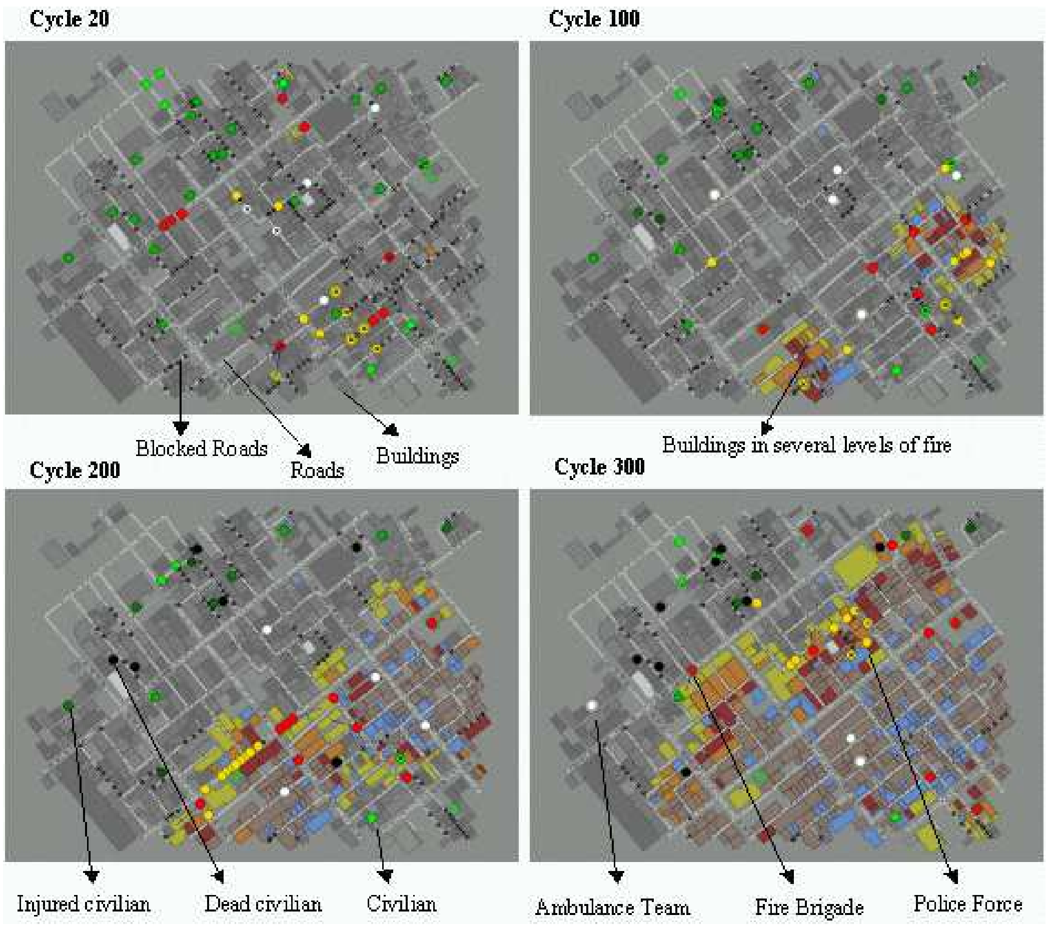


Figure D.1: Four different moments of an experiment in the RoboCup Rescue Simulator.

Appendix E

Initialisation File for Experiments in RCR Simulator

The lines in follow represent the content of the gisini.txt file, whose function is to mainly indicate the number and initial position of agents and some other objects (fire points and refuges) to the simulator. Note that if we define a number *x* for some kind of object, we must also define one entry for each of these *x* objects. For example, if we have **ambulanceTeamNum=5**, we must define 5 entries like **AmbulanceTeam0=7...**.

```
[MotionLessObject]
AmbulanceCenterNum=1
FireStationNum=1
PoliceOfficeNum=1
RefugeNum=7
AmbulanceCenter0=2,1,713,230518,37249 FireStation0=3,1,306,229997,36914
PoliceOffice0=4,1,140,228214,36563 Refuge0=5,1,331,231370,35376
Refuge1=5,1,288,229327,35695 Refuge2=5,1,389,228542,36445
Refuge3=5,1,216,230594,35305 Refuge4=5,1,579,230928,35685
Refuge5=5,1,567,229765,38020 Refuge6=5,1,670,231163,37141
[MoveObject]
CivilianNum=72
AmbulanceTeamNum=5
```

```
FireBrigadeNum=10
PoliceForceNum=10
Civilian0=6,1,331,231370,35376,-1,0,0,0 Civilian1=6,1,331,231370,35376,-1,0,0,0
Civilian2=6,1,331,231370,35376,-1,0,0,0 Civilian3=6,1,331,231370,35376,-1,0,0,0
Civilian4=6,1,331,231370,35376,-1,0,0,0 Civilian5=6,1,331,231370,35376,-1,0,0,0
Civilian6=6,1,331,231370,35376,-1,0,0,0 Civilian7=6,1,331,231370,35376,-1,0,0,0
Civilian8=6,1,331,231370,35376,-1,0,0,0 Civilian9=6,1,331,231370,35376,-1,0,0,0
Civilian10=6,1,331,231370,35376,-1,0,0,0 Civilian11=6,1,331,231370,35376,-1,0,0,0
Civilian12=6,1,331,231370,35376,-1,0,0,0 Civilian13=6,1,331,231370,35376,-1,0,0,0
Civilian14=6,1,331,231370,35376,-1,0,0,0 Civilian15=6,1,331,231370,35376,-1,0,0,0
Civilian16=6,1,331,231370,35376,-1,0,0,0 Civilian17=6,0,543,230125,35573,-1,0,0,0
Civilian18=6,0,243,228084,37690,-1,0,0,0 Civilian19=6,0,78,231586,37069,-1,0,0,0
Civilian20=6,1,3,228612,35726,-1,0,0,0 Civilian21=6,0,100,230954,35758,-1,0,0,0
Civilian22=6,1,621,230685,37021,-1,0,0,0 Civilian23=6,1,386,228878,36958,-1,0,0,0
Civilian24=6,1,545,227726,36359,-1,0,0,0 Civilian25=6,1,545,227726,36359,-1,0,0,0
Civilian26=6,1,545,227726,36359,-1,0,0,0 Civilian27=6,1,545,227726,36359,-1,0,0,0
Civilian28=6,1,545,227726,36359,-1,0,0,0 Civilian29=6,1,47,228353,36735,-1,0,0,0
Civilian30=6,1,47,228353,36735,-1,0,0,0 Civilian31=6,1,47,228353,36735,-1,0,0,0
Civilian32=6,1,652,228148,36802,-1,0,0,0 Civilian33=6,1,652,228148,36802,-1,0,0,0
Civilian34=6,1,652,228148,36802,-1,0,0,0 Civilian35=6,1,339,228370,37290,-1,0,0,0
Civilian36=6,1,339,228370,37290,-1,0,0,0 Civilian37=6,1,385,227859,37071,-1,0,0,0
Civilian38=6,1,385,227859,37071,-1,0,0,0 Civilian39=6,1,385,227859,37071,-1,0,0,0
Civilian40=6,1,308,228816,37817,-1,0,0,0 Civilian41=6,1,308,228816,37817,-1,0,0,0
Civilian42=6,1,308,228816,37817,-1,0,0,0 Civilian43=6,1,150,228945,37847,-1,0,0,0
Civilian44=6,1,150,228945,37847,-1,0,0,0 Civilian45=6,1,150,228945,37847,-1,0,0,0
Civilian46=6,1,660,228892,37727,-1,0,0,0 Civilian47=6,1,660,228892,37727,-1,0,0,0
Civilian48=6,1,660,228892,37727,-1,0,0,0 Civilian49=6,1,561,229275,37592,-1,0,0,0
Civilian50=6,1,561,229275,37592,-1,0,0,0 Civilian51=6,1,561,229275,37592,-1,0,0,0
Civilian52=6,1,60,229191,37352,-1,0,0,0 Civilian53=6,1,60,229191,37352,-1,0,0,0
Civilian54=6,0,475,229692,37520,-1,0,0,0 Civilian55=6,1,752,229098,37335,-1,0,0,0
Civilian56=6,1,752,229098,37335,-1,0,0,0 Civilian57=6,1,752,229098,37335,-1,0,0,0
```

```
Civilian58=6,1,752,229098,37335,-1,0,0,0 Civilian59=6,1,658,229874,36907,-1,0,0,0
Civilian60=6,1,202,230450,37861,-1,0,0,0 Civilian61=6,1,202,230450,37861,-1,0,0,0
Civilian62=6,1,202,230450,37861,-1,0,0,0 Civilian63=6,1,202,230450,37861,-1,0,0,0
Civilian64=6,1,756,230695,37973,-1,0,0,0 Civilian65=6,1,756,230695,37973,-1,0,0,0
Civilian66=6,1,756,230695,37973,-1,0,0,0 Civilian67=6,1,405,231306,38043,-1,0,0,0
Civilian68=6,1,405,231306,38043,-1,0,0,0 Civilian69=6,1,405,231306,38043,-1,0,0,0
Civilian70=6,1,405,231306,38043,-1,0,0,0 Civilian71=6,1,405,231306,38043,-1,0,0,0
AmbulanceTeam0=7,0,16,229387,36873,-1,0,0,0
AmbulanceTeam1=7,0,223,229531,36682,-1,0,0,0
AmbulanceTeam2=7,0,21,230369,36665,-1,0,0,0
AmbulanceTeam3=7,0,273,230302,37181,-1,0,0,0
AmbulanceTeam4=7,0,68,229500,36365,-1,0,0,0
FireBrigade0=8,0,156,231260,35626,-1,0,0,0 FireBrigade1=8,0,71,229480,37338,-1,0,0,0
FireBrigade2=8,0,175,230580,36830,-1,0,0,0 FireBrigade3=8,0,528,231149,35535,-1,0,0,0
FireBrigade4=8,0,581,230701,36308,-1,0,0,0 FireBrigade5=8,0,740,231630,35524,-1,0,0,0
FireBrigade6=8,0,471,229803,35980,-1,0,0,0 FireBrigade7=8,0,625,229851,36238,-1,0,0,0
FireBrigade8=8,0,667,229694,36113,-1,0,0,0 FireBrigade9=8,0,189,229145,36814,-1,0,0,0
PoliceForce0=9,2,284,230201,36476,636,230272,36388,7
PoliceForce1=9,0,431,229653,36885,-1,0,0,0
PoliceForce2=9,0,160,230159,36493,-1,0,0,0 PoliceForce3=9,0,389,231441,35762,-1,0,0,0
PoliceForce4=9,0,343,229841,36461,-1,0,0,0 PoliceForce5=9,0,474,229235,36690,-1,0,0,0
PoliceForce6=9,0,314,228899,37109,-1,0,0,0 PoliceForce7=9,0,263,229119,37051,-1,0,0,0
PoliceForce8=9,0,533,229336,37273,-1,0,0,0 PoliceForce9=9,0,81,230447,37004,-1,0,0,0
[FirePoint]
FirePointNum=5
FirePoint0=10,1,668,229833,38095 FirePoint1=10,1,266,229891,37551
FirePoint2=10,1,630,231163,36423 FirePoint3=10,1,470,229701,35243
FirePoint4=10,1,648,228074,35789
```

Appendix F

Pseudocode in Java of the Commitment Constraint Manager

Creating constraints: the first step of the commitment process is to create constraints of commitment. We are considering that all delegated activities must have an associated commitment constraint. Thus, we have extended the *ForwardingHandler* class, which accounts for the activities' delegation process, so that it automatically creates a commitment constraint for each delegated activity. The code[1] below represents the handler method of the *ForwardingHandler* class and the bold face text represents the new code associated with the commitment constraint creation.

```
public void handle(AgendaItem item) {
   item.getAbout().forwardTo(toName, reportBack);
   ...
   LList pattern = Lisp.list(toName,item.getId());
   PatternAssignment pa = new PatternAssignment(pattern,new Variable());
   Constraint c = new Constraint("commitment",null,pa);
   modelManager.addConstraint(item.getParent(),c);
}
```

[1]The pieces of code showed through this appendix give a general idea about the relations between different components of the architecture. Note, however, that they hide several details and cannot be directly implemented.

Adding constraints to model manager: the last line in the previous code adds the commitment constraint to the agent model manager, which recognises the constraint type and sends the constraint to the commitment constraint manager (CCM). The *addConstraint* method is defined below:

```
public void addConstraint(PNode item,Constraint c) {
    item.addExecutionReportListener(this);
    if(constraintsSet.hasKey(item)
        (constraintsSet.get(item)).add(c);
    else {
        Vector constraints = new Vector();
        constraints.add(c);
        constraintsSet.put(item,constraints);
    }
}
```

The second *addConstraint* method has a different input (just a constraint) and it is used to receive constraints related to messages of other agents. Such constraints have a defined value (true or false) that binds the correspondent variable. After that, the manager checks the effects of this new constraint in the model.

```
public void addConstraint(Constraint c) {
    // variable bindings are carried out by other I-X components
    while(constraintsSet.hasMore()) {
        constraintSet = constraintsSet.next();
        if(constraintSet.forAllTrue())
            prepareCommitment(constraintSet.key());
        if(constraintSet.existOneFalse())
            prepareFailure(constraintSet.key(),constraintSet.values());
    }
}
```

Preparing and sending commitments: this method prepares a commitment constraint with a true boolean value and sends such constraint to the superior agent if there is one. This method can be triggered by the *addConstraint* method, or when an activity becomes possible (there is a plan or action to it).

```
public void prepareCommitment(PNode node) {
   if(node.getSenderId()!= mySelf()) {
      LList pattern = Lisp.list(mySelf(),node.getId());
      PatternAssignment pa = new PatternAssignment(pattern,true);
      Constraint c = new Constraint("commitment",null,pa);
      IPC.sendMessage(node.getSenderId(),c);
   }
}
```

Preparing the failure report: this method creates an activity clone so that it can be reconsidered by the system. In addition, all the agents that are dealing with sub-activities associated with the original activity receive a failure report.

```
public void prepareFailure(PNode node, List constraints) {
   agenda.remove(node);
   activity = new Activity(node.getPattern());
   activityItem = new ActivityItem(activity);
   agenda.addItem(activityItem);
   while(constraints.hasNext()) {
      constraint = constraints.next();
      PatternAssignment pv = (PatternAssignment)c.getParameter(0);
      report = new Report(ReportType.FAILURE,null);
      report.setRef((pv.getPattern()).get(1);
      IPC.sendMessage((pv.getPattern()).get(0),report);
   }
}
```

Sending a failure (non-commit): when the constraint model decides that an activity is impossible, it changes the status of such an activity so that listeners on this activity are triggered. The CCM implements the *AgendaItemListener* interface so that it uses this event to create a commitment constraint with a false boolean value, sending such a constraint to its superior.

```
public void statusChanged(AgendaItemEvent e) {
    AgendaItem node = (AgendaItem) e.source();
    if(node.getStatus()==Status.IMPOSSIBLE) {
        LList pattern = Lisp.list(mySelf(),node.getId());
        PatternAssignment pa = new PatternAssignment(pattern,false);
        Constraint c = new Constraint("commitment",null,pa);
        IPC.sendMessage(node.getSenderId(),c);
    }
}
```

Monitoring the activity performance: the monitoring process accounts mainly for sending execution reports to superior agents. The process monitors, via the *ExecutionReportListener*, constraint events such as changes of values and variable binding. Note that completion and failure reports are already provided by other components of the system.

```
public void newBinding(ConstraintProcessEvent e, PNode item) {
    constraint = (Constraint) e.getSource();
    IPC.sendMessage(item.getSenderId(),constraint);
}

public void valueChanged(ConstraintProcessEvent e, PNode item) {
    constraint = (Constraint) e.getSource();
    IPC.sendMessage(item.getSenderId(),constraint);
}
```

Appendix G

Pseudocode in Java of the Mutual Support Constraint Manager

Filtering constraints: the model manager uses the *id-sender* constraint attribute, rather than the type attribute, to filter the constraints that it will send to the mutual support constraint manager (MSCM). The current condition is that if the constraints's *id-sender* is part of the peer contact group of an agent,such constraint is sent to its MSCM. The code[1] below shows how we can implement this feature in the *ModelManager* class.

```
public void addConstraint(Map idToItemMap, Constraint c) {
    ...
    Name idSender = c.getSenderId();
    if((contactManager.getAgentData(AgentRelationship.PEER)).contains(idSender))
        mutualSupportCM.add(c);
    ...
}
```

Dealing with events: constraints that have been added into MSCM are kept in a list (*constraintList*), which is monitored by a listener. This listener generates events, in case of contrasts, that trigger the method specified in follow. The method initially asks

[1]The pieces of code showed through this appendix give a general idea about the relations between different components of the architecture. Note, however, that they hide several details and cannot be directly implemented.

the IPlan component to generate a new node (activity) that considers the constraint parameter as a goal. If this is not possible (for example, because the agent does not have the necessary capability), the method sends its current constraint to the external constraint source. Finally the constraint is removed from the list.

```
public void newContrast(ContrastProcessEvent e) {
    externalC = (Constraint)e.getSource();
    currentC = (Constraint)e.getContrastSource();
    PNode node = IPlan.generateNode(externalC.getParameter(0));
    if(node != null)
        agenda.addItem(node);
    else
        IPC.sendMessage(externalC.getSenderId(),currentC);
    constraintList.remove(externalC);
}
```

Looking for invalid constraints: the process that accounts for eliminating invalid constraints is represented by a low priority thread that verifies time stamps of constraints, removing those with old stamps. This approach enforces two features on the system: (1) the system needs a temporal notion and (2) external constraints should (but do not need to) have an interval as last object of their pattern parameters.

```
public class Eliminator extends Thread {
    ..
    for(;constraintList.hasNext();constraint = (Constraint) constraintList.next()) {
        LList pattern = ((PatternAssignment) constraint.getParameter(0)).getPattern();
        long endTime = pattern.get(pattern.size()-1);
        if(endTime > Util.getWorldTime()) constraintList.remove(constraint);
    }
}
```

Bibliography

[Allen, 1991] Allen, J. (1991). Planning as Temporal Reasoning. *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts, USA, pp.3-14.

[Allen and Ferguson, 2002] Allen, J. and Ferguson, G. (2002). Human-Machine Collaborative Planning. *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*, Houston, Texas, USA.

[Allen et al., 1999] Allen, J., Guinn, C. and Horvitz, E. (1999). Mixed-initiative Interaction. *IEEE Intelligent Systems*, 14(5):14-23.

[Allsopp et al., 2002] Allsopp, D., Beautement, P., Bradshaw, J., Durfee, E., Kirton, M., Knoblock, C., Suri, N., Tate, A. and Thompson, C. (2002). Coalition Agents Experiment: Multi-Agent Co-operation in an International Coalition Setting. *IEEE Intelligent Systems*,17(3):26-35.

[Ankolekar et al., 2002] Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T. and Sycara, K. (2002). DAML-S: Web Service Description for the Semantic Web. *Proceedings of the First International Semantic Web Conference*, Chia, Sardegna, Italy.

[Arthur and Stillman, 1992] Arthur, R. and Stillman, J. (1992). Tachyon: A Model and Environment for Temporal Reasoning. *Technical report, GE Corporate Research and Development Center*, Schenectady, New York.

[Ball et al., 2002] Ball, H., Evans, C., Ballard, J. and Ball, J. (2002). *Safe Passage: Astronaut Care for Exploration Missions.* National Academy Press.

[Bartak, 1999] Bartak, R. (1999). Constraint Programming - What is behind? *Proceedings of the Workshop on Constraint Programming in Decision and Control*, Gliwice, Poland, pp.7-15.

[Bartak, 2001] Bartak, R. (2001). Theory and Practice of Constraint Propagation. *Proceedings of the Third Workshop on Constraint Programming in Decision and Control*, Gliwice, Poland, pp.7-14.

[Beek and Chen, 1999] Beek, P. and Chen, X. (1999). CPlan: A Constraint Programming Approach to Planning. *Proceedings of the Sixteenth National Conference on Artificial intelligence*, Orlando, Florida, USA, pp.585-590.

[Bradshaw et al., 2002] Bradshaw, J., Boy, G., Durfee, E., Gruninger, M., Hexmoor, H., Suri, N., Tambe, M., Uschold, M. and Vitek, J. (2002). Software Agents for the Warfighter. *ITAC Consortium Report*, Cambridge, MA: AAAI Press/The MIT Press.

[Brenner, 2003] Brenner, M. (2003). A Multiagent Planning Language. *Proceedings of the ICAPS Workshop on PDDL*, Trento, Italy.

[Chandrasekaran et al., 1999] Chandrasekaran, B., Josephson, R. and Benjamins, V. (1999). What are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems*, 14(1):20-26.

[Chapman, 1987] Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333-377.

[Chaulpsky et al., 2001] Chaulpsky, H., Gil, Y., Knoblock, C., Oh, J., Lerman, K., Pynadath, D., Russ, T. and Tambe, M. (2001). Electric Elves: Applying Agent Technology to Support Human Organizations. *Proceedings of the International Conference on Innovative Applications of Artificial Intelligence*, Seattle, Washington, USA, pp.51-58.

[Chien et al., 2000] Chien, S., Knight, R., Stechert, A., Sherwood, R. and Rabideau, G. (2000). Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. *Proceedings of the Fifth International Conference on AI Planning and Scheduling*, Breckenridge, Colorado, USA, pp.300-307.

[Chien et al., 2003] Chien et al. (2003). Autonomous Science on the EO-1 Mission. *Proceedings of the International Symposium on AI, Robotics and Automation in Space*, Nara, Japan.

[CoAX, 2004] CoAX Research Group. (2004). CoAX - Coalition Agents eXperiment. `http://www.aiai.ed.ac.uk/project/coax/`.

[Cohen and Levesque, 1991] Cohen, P. and Levesque, H. (1991). Teamwork. *Nous, Special Issue on Cognitive Science and Artificial Intelligence*, 25(4):487-512.

[Conklin, 2003] Conklin, J. (2003). Dialog mapping: reflections on an industrial strength case study. *Visualizing argumentation: software tools for collaborative and educational sense-making*, Springer-Verlag: London, pp. 117–136.

[Conry et al., 1988] Conry, S., Meyer, A. and Lesser, V. (1988). Multistage Negotiation in Distributed Planning. Bond, A. and Gasser, L. (Ed.), *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, California, pp. 367-384.

[Corkill, 1979] Corkill, D. (1979). Hierarchical Planning in a Distributed Environment. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan, pp. 168-175.

[Decker and Lesser, 1995] Decker, K. and Lesser, V. (1995). Designing a Family of Coordination Algorithms. *Proceedings of the First International Conference on Multiagent Systems*, San Francisco, California, USA, pp.73-80.

[DesJardins and Wolverton, 1999] DesJardins, M. and Wolverton, M. (1999). Coordinating a Distributed Planning System. *AI Magazine*, 20(4):45-53.

[Dix et al., 2002] Dix, J., Munoz-Avila, H., Nau, D. and Zhang, L. (2002). IMPACTing SHOP: Putting an AI Planner into a Multi-Agent Environment. *Annals of Mathematics and Artificial Intelligence*, 37(4):381 - 407.

[Do and Kambhampati, 2000] Do, B. and Kambhampati, S. (2000). Solving Planning Graph by Compiling it into a CSP. *Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO, USA.

[Dorais et al., 1998] Dorais, G., Bonasso, R., Kortenkamp, R., Pell, B. and Schreckenghost, D. (1998). Adjustable Autonomy for Human-centered Autonomous Systems on Mars. *Proceedings of the First International conference of the Mars Society*, pp. 397-420.

[Durfee et al., 1987] Durfee, E., Lesser, V. and Corkill, D. (1987). Coherent Cooperation among Communication Problem Solvers. *IEEE Transactions on Computers*, 36(11):1275 - 1291.

[Eiter et al., 1999] Eiter, T., Subrahmanian, V. and Pick, G. (1999). Heterogeneous Active Agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179-255.

[Erol et al., 1994] Erol, K., Nau, D. and Hendler, J. (1994). HTN Planning: Complexity and Expressivity. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, USA, pp.1123-1128.

[Estlin, 2004] Estlin, T. (2004). MISUS: Multi-Rover Integrated Science Understanding system. `http://www-aig.jpl.nasa.gov/public/planning/dist-rovers/`

[Estlin et al., 2003] Estlin, T., Castano, R., Anderson, B., Gaines, D., Fisher, F. and Judd, M. (2003). Learning and Planning for Mars Rover Science. *Proceedings of the IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modelling, planning, learning, and communicating*, Acapulco, Mexico.

[Ferguson, 2004] Ferguson, B. (2004). *Foundations of New Zealand Military Doctrine.* Headquarters New Zealand Defence Force.

[Ferguson and Allen, 1998] Ferguson, G. and Allen, J. (1998). TRIPS: An Intelligent Integrated Problem-Solving Assistant. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, USA, pp. 567-573.

[Ferguson et al., 1996] Ferguson, G., Allen, J. and Miller, B. (1996). TRAINS-95: Towards a Mixed-Initiative Planning Assistant. *Proceedings of the Third Conference in AI Planning Systems*, AAAI Press, Menlo Park, California, pp.70-77.

[Fikes and Nilsson, 71] Fikes, R. and Nilsson, N. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 5(2):189-208.

[Fox and Long, 2001] Fox, M. and Long, D. (2001). An Extension to PDDL for Expressing Temporal Planning Domains. *Technical Report, Department of Computer Science, University of Durham*, Durham, UK.

[Frank et al., 2000] Frank, J., Jonsson, A. and Morris, P. (2000). On Reformulating Planning as Dynamic Constraint Satisfaction. *Proceedings of the Fourth International Symposium on Abstraction, Reformulation, and Approximation*, Horseshoe Bay, Texas, USA, pp.271-280

[Freksa, 1992] Freksa, C. (1992). Temporal Reasoning Based on Semi-intervals. *Artificial Intelligence*, 54(1-2):199-228.

[Freuder, 1978] Freuder, E. (1978). Synthesizing Constraint Expressions. *Communications ACM*, 21(11):958-966.

[Georgeff, 1983] Georgeff, M. (1983). Communication and Interaction in Multi-Agent Planning. *Proceedings of the Third National Conference in Artificial Intelligence*, pp.125-129.

[Ghallab et al., 2004] Ghallab, M., Nau, D. and Traverso, P. (2004). *Automated Planning: theory and practice.* Morgan Kaufmann, San Francisco, California, USA.

[Goldman et al., 2000] Goldman, R., Haigh, K., Musliner, D. and Pelican, M. (2000). MACBeth: A Multi-Agent Constraint-Based Planner. *AAAI Workshop on Constraints and AI Planning*, Austin, Texas, USA, pp.11-17.

[Grosz, 1996] Grosz, B. (1996). Collaborative Systems. *AI Magazine*, 17(2):67–85.

[Grosz et al., 1999] Grosz, B., Hunsberger, L. and Kraus, S. (1999). Planning and Acting Together. *AI Magazine*, 5(2):23-34.

[Gruber, 1995] Gruber, T. (1995). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *Journal of Human-Computer Studies*, 43(5/6):907-928.

[Hadad, 1997] Hadad, M. (1997). *Using SharedPlan Model in Electronic Commerce Environment.* Master's Thesis, Bar Ilan University, Ramat-Gan, Israel.

[Hayes-Roth, 1985] Hayes-Roth, B. (1985). A Blackboard Architecture for Control. *Artificial Intelligence*, 26(3):251-321.

[Hill et al., 2000] Hill, J., Surdu, J., Ragsdale, D. and Schafer, J. (2000). Anticipatory Planning in Information Operations. *IEEE International Conference on Systems, Man and Cybernetics*, Nashville, Tennessee, USA, pp.2350-2355.

[Horvitz et al., 1999] Horvitz, E., Jacobs, A. and Hovel, D. (1999). Attention-sensitive Alerting. *Proceedings of the Third Conference on Uncertainty and Artificial Intelligence*, Stockholm, Sweden, pp.305-313.

[Jennings, 1990] Jennings, N. (1990). Coordination Techniques for Distributed Artificial Intelligence. *Foundations of Distributed Artificial Intelligence*, London, Wiley, pp. 187-210.

[Jennings, 1992] Jennings, N. (1992). Towards a Cooperation Knowledge Level for Collaborative Problem Solving. *Proceedings of the Tenth European Conference on Artificial Intelligence*, Vienna, Austria, 224-228.

[Jennings, 1995] Jennings, N. (1995). Controlling Cooperative Problem Solving in Industrial Multiagent Systems Using Joint Intentions. *Artificial Intelligence*, 75(2):195-240.

[Jennings et al., 1998] Jennings, N., Sycara, K. and Wooldridge, M. (1998). A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7-38.

[Joint Chiefs of Staff, 1996] Joint Chiefs of Staff. (1996). US Joint Publication for Doctrine for Combat Search and Rescue. *Joint Publication 3-50.2*

[Jonsson et al., 2000] Jonsson, A., Morris, P., Muscettola, N., Rajan, K. and Smith, B. (2000). Planning in interplanetary space: theory and practice. *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, Colorado, USA, pp.177-186.

[Joslin and Pollack, 1995] Joslin, D. and Pollack, M. (1995). Passive and Active Decision Postponement in Plan Generation. *Proceedings of the European Workshop on Planning*, Assisi, Italy.

[Killion, 2000] Killion, T. (2000). Decision Making and the Levels of War. *Military Review*, 80(6):66-70.

[Kinny et al., 1992] Kinny, D., Ljungberg, M., Rao, A., Tidhar, G. and Werner, E. (1992). Planned Team Activity. *Proceedings of the Fourth European Workshop on Modelling Autonomous Agents in a Multiagent World*, Rome, Italy, pp.227-256.

[Kitano and Tadokoro, 2001] Kitano, H. and Tadokoro, S. (2001). RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *AI Magazine*, 22(1):39-52.

[Knoblock and Minton, 1998] Knoblock, C. and Minton, S. (1998). The Ariadne Approach to Web-based Information Integration. *IEEE Intelligent Systems*, 13(5):17-20.

[Kunz and Rittel, 1970] Kunz, W. and Rittel, H. (1970). Issues as Elements of Information Systems. *Working Paper 131*, Institute of Urban and Regional Development, University of California, Berkeley, California, USA.

[Lever and Richards, 1994] Lever, J. and Richards, B. (1994). parcPLAN: a Planning Architecture with Parallel Actions, Resources and Constraints. *Proceedings of the Ninth International Symposium on Methodologies for Intelligent Systems*, Zakopane, Poland, pp.213-222.

[Levesque et al., 1990] Levesque, J., Cohen, P. and Nunes, J. (1990). On Acting Together. *Proceedings of the Eighth National Conference on Artificial Intelligence*, Los Altos, California, USA, pp.94-99.

[Lino et al., 2003] Lino, N., Tate, A., Siebra, C. and Chen-Burger, Y. (2003). Delivering Intelligent Planning Information to Mobile Devices Users in Collaborative Environments. *Proceedings of the IJCAI Workshop on Artificial Intelligence, Information Access and Mobile Computing*, Acapulco, Mexico.

[MacLean et al., 1991] MacLean, A., Young, R., Berllotti, V. and Moran, T. (1991). Questions, Options, and Criteria: Elements of Design Space Analysis. *Human-Computer Interaction*, 6(3-4):201- 250.

[Maheswaran et al., 2004] Maheswaran, R., Tambe, M., Varakantham, P. and Myers, K. (2004). Adjustable Autonomy: Challenges in Personal Assistant Agents: A Position Paper. In Klusch, M., Weiss, G. and Rovatsos, M. (Ed.), *Agents and Computational Autonomy: Potential, Risks and Solutions*, Lecture Notes in Computer Science, Vol. 2969, Springer Verlag, pp.187-194.

[McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso M., Weld, D. and Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. *Technical Report CVC TR-98-003/DCS TR-1165*, Yale Center for Computational Vision and Control.

[Mishkin et al., 1998] Mishkin, A., Morrison, J., Nguyen, T., Stone, H., Cooper, B. and Wilcox, B. (1998). Experiences with Operations and Autonomy of the Mars Pathfinder Microrover. *Proceedings of the IEEE Aerospace Conference*, Aspen, Colorado, USA, 2:337-351.

[Mohammed, 2001] Mohammed, J. (2001). Mission Planning for Formation-Flying Satellite Cluster. *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, Key West, Florida, USA, pp.58-62.

[Moran et al., 1997] Moran, D., Cheyer, A., Julia, L., Martin, D. and Park, S. (1997). Multimodal User Interfaces in the Open Agent Architecture. *Proceedings of the International Conference on Intelligent User Interfaces*, Orlando, Florida, USA, pp.61-68.

[MPAT, 2004a] MPAT. (2004). Multinational Planning Augmentation Team. `http://www2.apan-info.net/mpat/`

[MPAT, 2004b] MPAT. (2004). Multinational Forces Standard Operating Procedures `http://www2.apan-info.net/mnfsop/`

[Muscettola et al., 1998] Muscettola, N., Nayak, P., Pell, B. and William, B. (1998). Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5-48.

[Myers, 1996] Myers, K. (1996). Advisable Planning Systems. *Advanced Planning Technology*, AAAI Press, pp.206-209.

[Myers and Morley, 2003] Myers. K. and Morley, D. (2003). Policy-based agent directability. Hexmoor, H., Falcone, R. and Castelfranchi, C. (Ed.), *Agent Autonomy*, pp. 187-210, Kluwer Academic Publishers.

[Myers et al., 2001] Myers, K., Jarvis, P. and Lee, T. (2001). CODA: Coordinating Human Planners. *Proceedings of the Sixth European Conference on Planning*, Toledo, Spain.

[Nareyek, 2000] Nareyek, A. (2000). Open World Planning as SCSP. *Proceedings of the AAAI-2000 Workshop on Constraints and AI Planning*, Austin, Texas, USA, pp.35–46.

[Narayek, 2001] Nareyek, A. (2001). *Constraint-Based Agents - An Architecture for Constraint-Based Modeling and Local-Search-Based Reasoning for Planning and Scheduling in Open and Dynamic Worlds.* Reading, Springer LNAI 2062.

[Nareyek et al., 2005] Nareyek, A., Freuder, E., Fourer, R., Giunchiglia, E., Goldman, R., Kautz, H., Rintanen, J. and Tate, A. (2005). Constraints and AI Planning. *IEEE Intelligent Systems*, 20(2):62-72

[Nwana et al., 1996] Nwana, H., Lee, L. and Jennings, N. (1996). Coordination in Software Agent Systems. *BT Technology Journal*, 14(4):79-88.

[Pechoucek et al., 2002] Pechoucek, M., Marik, V. and Barta, J (2002). A Knowledge-Based Approach to Coalition Formation. *IEEE Intelligent Systems*, 17(3):17-25.

[Pegram et al., 1999] Pegram, D., Amant, R. and Riedl, M. (1999). An approach to visual interaction in mixed-initiative planning. *Proceedings of the AAAI Workshop on Mixed-Initiative Intelligence*, Orlando, Florida, USA, pp. 15-23.

[Pollack, 1996] Pollack, M. (1996). Planning in Dynamic Environments: The DIPART System. *Advanced Planning Technology*, AAAI Press, pp.218-225.

[Polyak and Tate, 1998] Rationale in Planning: Causality, Dependencies and Decisions. *Knowledge Engineering Review*, 13(3):247-262.

[Puterman, 1994] Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, New York, NY, USA.

[Rabideau et al., 1999] Rabideau, G., Knight, R., Chien, S., Fukunaga, A. and Govindjee, A. (1999). Iterative Repair Planning for Spacecraft Operations in the ASPEN System. *Proceedings of the International Symposium on Artificial Intelligence Robotics and Automation in Space*, Noordwijk, The Netherlands.

[Rao and Georgeff, 1991] Rao, A. and Georgeff, M. (1991). Toward a Formal Theory of Deliberation and its Role in the Formation of Intentions. *Technical report, Australian Artificial Intelligence Institute*, Victoria, Australia.

[Rao and Georgeff, 1995] Rao, A. and Georgeff, M. (1995). BDI Agents: From Theory to Practice. *Proceedings of the First International Conference on Multiagent Systems*, San Francisco, California, USA, pp.312-319.

[Rao et al., 1993] Rao, A., Lucas, A., Morley, D. and Selvestrel, M. (1993). Agent-Oriented Architecture for Air Combat Simulation. *Technical Note 42, Australian Artificial Intelligence Institute*, Melbourne, Australia.

[Rathmell, 1999] Rathmell, R. (1999). A Coalition Force Scenario: Binni - Gateway to the Golden Bowl of Africa. *Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces*, Edinburgh, UK, pp.115-125.

[Rich and Sidner, 1997] Rich, C. and Sidner, C. (1997). Collagen: When Agents collaborate with People. *Proceedings of the First International Conference on Autonomous Agents*, Marina del Ray, California, USA, pp.284-291.

[Richards et al., 2001] Richards, R., Houlette, R. and Mohammed, J. (2001). Distributed Satellite Constellation Planning and Scheduling. *Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference*, Key West, Florida, USA, pp.68-72.

[RoboCup, 2006] The RoboCup Federation. (2006). The RoboCup Soccer Web Page. `http://www.robocup.org`

[Sacerdoti, 1977] Sacerdoti, E. (1977). *A Structure for Plans and Behavior.* New York: American Elsevier.

[Saint-Andre, 2001] Saint-Andre, P. (2001) Jabber Technology Overview. *Jabber Software Foundation.*

[Scerri et al., 2001] Scerri, P., Pynadath, D. and Tambe, M. (2001). Adjustable Autonomy in Real-world Multiagent Environments. *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, pp.300-307.

[Schmorrow, 2002] Schmorrow, D. (2002). The DARPA Control of Agent Based Systems (CoABS) Program and Challenges for Collaborative Coalitions. *Proceedings of the Second International Conference on Knowledge Systems for Coalition Operations*, Toulouse, France, pp.182-183.

[Schreiber et al., 1999] Schreiber, G., Akkermas, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W. and Wielinga, B. *Knowledge engineering and management. The CommonKADS methodology.* Bradford book, MIT Press, London.

[Schwalb and Vila, 1998] Schwalb, E. and Vila, L. (1998). Temporal Constraints: A Survey. *Constraints*, Kluwer Academic Publishers, Boston, 3:129-149.

[Siebra, 2005] Siebra, C. (2005). Planning Requirements for Hierarchical Coalitions in Disaster Relief Domains. *Expert Update*, 8(1):20-24, Summer 2005, The Specialist Group on Artificial Intelligence, British Computer Society (BCS-SGAI).

[Siebra and Ramalho, 1999] Siebra, C. and Ramalho, G. (1999). Uma Arquitetura para Suporte de Atores Sinteticos em Ambientes Virtuais. *Second Brazilian Workshop on Artificial Intelligence*, Rio de Janeiro, Brazil.

[Siebra and Tate, 2003] Siebra, C. and Tate, A. (2003). I-Rescue: A Coalition Based System to Support Disaster Relief Operations. *Proceedings of the Third International Conference on Artificial Intelligence and Applications*, Benalmadena, Spain, pp.289-294.

[Siebra and Tate, 2004] Siebra, C. and Tate, A. (2004). Implementing Hierarchical Agent-Human Teamworks via Constraint-Based Models. *Proceedings of the International Conference on Artificial Intelligence and Applications*, as part of the Twenty-Second International Multi-Conference on Applied Informatics, Innsbruck, Austria.

[Siebra and Tate, 2005] Siebra, C. and Tate, A. (2005). Integrating Collaboration and Activity-Oriented Planning for Coalition Operations Support. *Springer Lecture Notes on Artificial Intelligence*, (Volume to be published).

[Siebra et al., 2004] Siebra, C., Tate, A. and Lino, N. (2004). Planning and Representation of Joint Human-Agent Space Missions via Constraint-Based Models. *Fourth International Workshop on Planning and Scheduling for Space*, Darmstadt, Germany, pp.180-188.

[Sierhuis et al., 2003] Sierhuis, M., Bradshaw, J., Acquisti, A., Hoof, R., Jeffers, R. and Uszok, A. (2003). Human-Agent Teamwork and Adjustable Autonomy in Practice. *Proceedings of the Seventh International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Nara, Japan.

[Silva et al., 2001] Silva, D., Siebra, C., Valadares, J., Frery, A., Falcao, J. and Ramalho, G. (2001). Synthetic Actor Model for Long-Term Computer Games. *Virtual Reality*, 5:1-10.

[Smith, 1988] Smith, R. (1988). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *Distributed Artificial Intelligence*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 357–366.

[Smith, 1994] Smith, S. (1994). OPIS: A Methodology and Architecture for Reactive Scheduling *Intelligent Scheduling*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 29-66.

[Smith et al., 1998] Smith, B., Sherwood, R., Govindjee, A., Yan, D., Rabideau, G., Chien, S. and Fukunaga, A. (1998). Representing Spacecraft Mission Planning Knowledge in ASPEN. *Proceedings of the AI Planning Systems Workshop on Knowledge Acquisition*, Pittsburgh, Pennsylvania, USA.

[Sqalli and Freuder, 1996] Sqalli. M. and Freuder, E. (1996). Inference-Based Constraint Satisfaction Supports Explanation. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, USA, pp.318-325.

[Stefik, 1981] Stefik, M. (1981). Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16(2):111-140.

[Sycara, 1989] Sycara, K. (1989). Multi-Agent Compromise via Negotiation. *Distributed Artificial Intelligence,* Volume 2, Morgan Kaufmann, Los Altos, CA.

[Sycara, 1998] Sycara, K. (1998). Multiagent Systems. *AI Magazine*, 19(2):79-92.

[Tambe, 1997a] Tambe, M. (1997). Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, 7:83-124.

[Tambe, 1997b] Tambe, M. (1997). Agent Architectures for Flexible, Practical Teamwork. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, Rhode Island, USA, pp. 22-28.

[Tambe, 2003] Tambe, M. (2003). Multiagent and Agent-Human Teamwork: Theory and Practice. *IJCAI-03 Tutorial*, Acapulco, Mexico.

[Tate, 1977] Tate, A. (1977). Generating Project Networks. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, USA, pp.888-893.

[Tate, 1995] Tate, A. (1995). Integrating Constraint Management into an AI Planner. *Journal of Artificial Intelligence in Engineering*, 9(3):221-228.

[Tate, 1997] Tate, A. (1997). Mixed Initiative Interaction in O-Plan. *Proceedings of the AAAI Spring Symposium on Computational Models for Mixed Initiative Interaction*, Stanford, California, USA.

[Tate, 2003] Tate, A. (2003). <I-N-C-A>: an Ontology for Mixed-Initiative Synthesis Tasks. *Proceedings of the IJCAI Workshop on Mixed-Initiative Intelligent Systems*, Acapulco, Mexico.

[Tate, 2004] Tate, A. (2004). I-X: Technology for Intelligent Systems. `http://www.aiai.ed.ac.uk/project/ix/`

[Tate et al., 1994] Tate, A., Drabble, B. and Kirby, R. (1994). O-Plan2: An Open Architecture for Command, Planning and Control. In Fox, M. and Zweben, M. (Ed.), *Intelligent Scheduling*, Morgan Kaufmann, pp.213-239.

[Tate et al.,2002] Tate, A., Dalton, J. and Stader, J. (2002). I-P^2 - Intelligent Process Panels to Support Coalition Operations. *Proceedings of the Second International Conference on Knowledge Systems for Coalition Operations*, Toulouse, France, pp.184-190.

[Tate et al., 2004] Tate, A., Dalton, J., Siebra, C., Aitken, S., Bradshaw, J. and Uszok, A. (2004). Intelligent Agents for Coalition Search and Rescue Task Support. *Proceedings of the Nineteenth AAAI National Conference on Artificial Intelligence*, San Jose, California, USA.

[Tidhar et al., 1996] Tidhar, G., Rao, A. and Sonenberg, E. (1996). Guided Team Selection. *Proceedings of the Second International Conference on Multiagent Systems*, Kyoto, Japan, pp.369-376.

[U.S Marine, 1994] U.S. Marine Corps. (1994). *Intelligence Preparation of the Battlefield.* Doctrine Division, Department of Army, Washington, DC, USA.

[Valente et al., 1999] Valente, A., Russ, T., MacGregor, R. and Swartout, W. (1999). Building and (Re)Using an Ontology of Air Campaign Planning. *IEEE Intelligent Systems*, 14(1):27-36.

[Verfaillie and Schiex, 1994] Verfaillie, G. and Schiex, T. (1994). Solution Reuse in Dynamic Constraint Satisfaction Problems. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, Washington, USA, pp.307-312.

[Verfaillie et al., 2003] Verfaillie, G., Garcia, F. and Peret. L. (2003). Deployment and Maintenance of a Constellation of Satellites. *Proceedings of the ICAPS Workshop on Planning under Uncertainty and Incomplete Information*, Trento, Italy.

[Walker, 1999] Walker, E. (1999). Coalition Planning for Operations Other Than War. *Proceedings of the International Workshop on Knowledge-Based Planning for Coalition Forces*, Edinburgh, UK, pp.23-27.

[Wilkins, 1988] Wilkins, D. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan-Kaufmann, San Francisco, California, USA.

[Wilkins and Myers, 1998] Wilkins, D. and Myers, K. (1998). A Multiagent Planning Architecture. *Proceedings of the Forth International Conference on AI Planning Systems*, Pittsburgh, USA, pp.154-162.

[Wilkins et al., 1995] Wilkins, D., Myers, K., Lowrance, J. and Wesley, L. (1995). Planning and Reacting in Uncertain and Dynamic Environments. *Journal of Experimental and Theoretical AI*, 7(1):197-227.

[Zetocha et al., 2000] Zetocha, P., Self, L., Wainwright, R., Burns, R., Brito, M. and Surka, D. (2000). Commanding and Controlling Satellite Clusters. *IEEE Intelligent Systems*, 15(6):8-13.