

A Thesis Submitted for the Degree of PhD at the University of Warwick

Permanent WRAP URL:

<http://wrap.warwick.ac.uk/170616>

Copyright and reuse:

This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

Please refer to the repository record for this item for information to help you to cite it.

Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk



Motivated Cooperation in Autonomous Agents

by

Nathan Griffiths

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science

University of Warwick, Department of Computer Science
August, 2000



Contents

List of Figures	xi
List of Tables	xii
Acknowledgements	xiv
Declaration	xiv
Abstract	xv
1 Introduction	1
1.1 Introduction	1
1.2 Agents	2
1.2.1 Autonomy	3
1.2.2 Other Characteristics	4
1.3 Cooperation	4
1.3.1 Autonomous Cooperation	5
1.3.2 Agent Architectures for Cooperation	6

1.4	Aims and Principles	7
1.5	Structure of Thesis	11
2	Related Work	12
2.1	Introduction	12
2.2	Agency and Autonomy	13
2.2.1	Autonomy Through Motivations	14
2.2.2	Motivated Goal Creation	15
2.2.3	Autonomy as a Dependence Relationship	16
2.3	Individual Intentionality	17
2.3.1	The Need for Intention	19
2.3.2	Cohen and Levesque’s Model of Intention	19
2.4	BDI-based Architectures	22
2.4.1	IRMA	23
2.4.2	PRS	24
2.4.3	Alternative Architectures	26
2.5	Social Intentionality	27
2.5.1	Joint Intention Theory	30
2.5.2	Social Power Theory	31
2.5.3	Commitments and Conventions	34
2.6	Multi-Agent Systems and Architectures	35
2.6.1	GRATE*	35
2.6.2	Planned Team Activity	37

2.6.3	STEAM	38
2.7	Stages in Cooperative Problem Solving	38
2.8	Summary	40
3	Motivated BDI Agents	42
3.1	Introduction	42
3.2	The Z Specification Language	44
3.3	Overview of SENARA	46
3.4	Primitives	49
3.5	Environment	50
3.6	Perceptions	51
3.7	Beliefs	52
3.8	Goals	53
3.9	Actions	54
3.10	Joint and Concurrent Actions	55
3.11	Plans	57
3.12	Intentions	60
3.13	Motivations	64
3.14	Summary	67
4	A Motivated BDI Agent Architecture	69
4.1	Introduction	69
4.2	Perceiving the Environment	72

4.3	Updating Beliefs	73
4.4	Updating Motivations	75
4.5	Ensuring Goals are Motivated	78
4.6	Goal Generation	80
4.7	Ensuring Intentions are Appropriate	83
4.8	Intention Adoption	85
4.8.1	Selecting and Adopting a Plan	86
4.8.2	Immediate Elaboration	90
4.8.3	Delayed Elaboration	92
4.9	Intention Selection	93
4.10	Action and Deliberation	95
4.11	Summary	97
5	Autonomous Cooperation in Open Environments	99
5.1	Introduction	99
5.2	Cooperative Intention	100
5.2.1	Requirements of a Model of Cooperation	101
5.2.2	Conventions for Cooperative Intention	103
5.2.3	Formalising Cooperative Intention	106
5.3	Stages in Cooperation	107
5.4	Risk in Cooperation	110
5.4.1	Trust	111
5.4.2	Updating Trust of Others	112

5.4.3	Agent Models	113
5.5	Overview of the Warehouse Domain	115
5.6	Summary	117
6	Plan Selection	119
6.1	Introduction	119
6.2	Cooperative Plan Selection	120
6.3	Plan Selection Criteria	122
6.4	A Model of Cooperative Plan Selection	124
6.4.1	Plan Ratings	124
6.4.2	Assessing Contributions	125
6.4.3	Assessing Joint Actions	127
6.4.4	Assessing Concurrent Actions	128
6.4.5	Cooperative Rating of a Plan	129
6.4.6	Plan Quality	130
6.5	Pre-Execution Plan Assessment	131
6.5.1	Best-case and Mean-case Advantage	133
6.5.2	Recursion	134
6.5.3	Selecting Between Partial Plans	135
6.6	Warehouse Example	137
6.7	Summary	140
7	Cooperative Intention Formation	143

7.1	Introduction	143
7.2	Overview	144
7.3	Annotation Strategies	147
7.3.1	Choice of Annotation Strategy	149
7.3.2	Pre-annotated Plans	150
7.4	Plan Annotation	151
7.4.1	Agents to Annotate	152
7.4.2	Individual Action Annotation	153
7.5	Annotating Simultaneous Actions	154
7.5.1	Joint Actions	155
7.5.2	Concurrent Actions	157
7.5.3	Annotated Plans	158
7.6	Soliciting Commitment to Cooperate	160
7.7	Requesting Assistance	163
7.8	Nominal Commitment	165
7.9	Committing to Cooperate	168
7.9.1	Trust in Requesting Agent	171
7.9.2	Commitment to Actions	172
7.9.3	Commitment to Goals	173
7.9.4	Commitment to Plans	174
7.10	Generating Full Commitment to Cooperation	175
7.11	Cooperative Intentions in the Warehouse Domain	177

7.12	Commitment Strategies	179
7.12.1	Minimising Risk and Cost	180
7.12.2	Choice Factors	181
7.12.3	Strategy Choice	183
7.13	Intention Execution	185
7.13.1	Cooperative Plan Elaboration	186
7.13.2	Centralised Elaboration	188
7.14	Summary	190
8	Conclusions	193
8.1	Introduction	193
8.2	Contributions	194
8.3	Relation to Existing Work	196
8.4	Limitations and Future Work	198
8.5	Summary	200
	References	202
A	Specification	214
A.1	Introduction	214
A.2	Primitives	214
A.3	Environment	215
A.4	Perceptions	215
A.5	Beliefs	215

A.6	Goals	216
A.7	Actions	216
A.8	Joint and Concurrent Actions	217
A.9	Plans	217
A.10	Intentions	218
A.11	Motivations	218
A.12	Agent Mental Components	219
A.13	Perceiving the Environment	220
A.14	Updating Beliefs	220
A.15	Updating Motivations	221
A.16	Ensuring Goals are Motivated	222
A.17	Goal Generation	223
A.18	Ensuring Intentions are Appropriate	224
A.19	Intention Adoption	225
A.20	Intention Selection	227
A.21	Action and Deliberation	227
A.22	Necessary and Optional Cooperation	229
A.23	A Model of Cooperative Plan Selection	230
A.23.1	Assessing Joint Actions	232
A.23.2	Assessing Concurrent Actions	233
A.23.3	Cooperative Rating of a Plan	234
A.23.4	Plan Quality	234

A.24 Cooperation in Partial Plans	234
A.24.1 Best-case and Mean-case Advantage	236
A.24.2 Recursion	237
A.24.3 Selecting Between Partial Plans	237
A.25 Cooperative Intention	238
A.26 Cooperative Plans	239
A.27 Plan Annotation	240
A.27.1 Action Annotation	240
A.27.2 Joint Action Annotation	241
A.27.3 Concurrent Action Annotation	244
A.27.4 Annotated Plans	245
A.28 Soliciting Commitment to Cooperate	246
A.29 Nominal Commitment	249
A.30 Committing to Cooperate	250
A.31 Generating Full Commitment to Cooperation	252
A.32 Strategy Choice	255
A.33 Cooperative Plan Elaboration	256
A.34 Updating Trust of Others	257
B Implementation of the SENARA Testbed	258
B.1 Introduction	258
B.2 Overview of the Testbed	259
B.3 Plan Library	263

B.4	Synchronising and Ordering Action Execution	266
B.4.1	Synchronisation	266
B.4.2	Action Ordering	267
B.5	Example Interaction	269
B.5.1	Plan selection	269
B.5.2	Pre-execution Assessment	270
B.5.3	Plan Selection	273
B.5.4	Intention Adoption	275
B.5.5	Intention Execution	277

List of Figures

2.1	Norman and Long's motivated agent architecture (from [77])	16
2.2	The IRMA architecture (from [5])	23
2.3	The PRS architecture (from [38])	26
2.4	The GRATE* architecture for cooperation (based on [50])	36
3.1	Overview of the SENARA architecture	49
3.2	An example partial plan	58
3.3	The use of a <i>stack</i> of plans in an intention	63
4.1	The SENARA architecture	70
4.2	The problem of plan over-commitment	91
5.1	Example agent models	114
5.2	The warehouse environment	116
B.1	The interface to the SENARA testbed	261
B.2	Algorithm for inserting ordering actions based on Kinny <i>et al.</i> 's work . . .	269

List of Tables

2.1	Cohen and Levesque's definitions of <i>intentions</i> and <i>persistent goals</i>	22
2.2	Cohen and Levesque's definitions of <i>joint intentions</i> and <i>joint persistent goals</i>	30
3.1	Summary of the Z notation (taken from [25])	47
4.1	The stages in the SENARA reasoning cycle	71
4.2	Algorithm for agent perception	72
4.3	Algorithm for updating beliefs	74
4.4	Algorithm for updating the intensity of motivations	76
4.5	Algorithm for dropping unmotivated goals	79
4.6	Algorithm for goal generation	81
4.7	Algorithm for dropping inappropriate intentions	84
4.8	Algorithm for intention adoption	88
4.9	Algorithm for intention selection	94
5.1	Conventions for motivated cooperation	104
5.2	Observations about group mental state after Wooldridge and Jennings	105

7.1	Algorithm to determine whether responses are sufficient to enter into cooperation	146
7.2	The initiator's algorithm for cooperative intention formation	148
7.3	Valid and invalid annotations	155
7.4	The initiator's algorithm for requesting assistance	164
7.5	Algorithm for updating nominal commitment	166
7.6	The recipient's algorithm for processing a request	170
B.1	Plans for moving and recharging in the warehouse domain	263
B.2	Plans for storing and checking boxes in the warehouse domains	265

Acknowledgements

Any endeavour as consuming as the production of a thesis, necessitates the support and assistance of others. Many people have indirectly been involved in this work, and deserve thanks. First, I would like to thank Mike Luck for his encouragement, supervision, perception, and for the many discussions that helped clarify my ideas. Second, I would like to thank Mike Joy for his guidance and advice. Third, thanks go to the members of the Agent-Based Systems Group at Warwick for their comments and discussions, in particular thanks to Kevin Bryson and Simon Miles. Fourth, thanks to Mark d’Inverno for looking through extracts of the Z specification contained in this thesis. Fifth, I would like to thank EPSRC for financial support. Finally, my utmost thanks to Jane for moral and practical support, and being there when I needed her.

Declaration

The contents of this thesis are a result of my own work, and it contains nothing that is based on collaborative research. No part of the work contained in this thesis has been submitted for any degree or qualification at any other university. Earlier work on various aspects of this thesis has been published in [40] and [64].

Abstract

Multi-agent systems are underpinned by the notion of cooperation — the process by which independent agents act together to achieve particular goals. Cooperation between autonomous agents requires appropriate motivations on behalf of those agents, since an agent's behaviour is guided by its motivations. Interaction with others involves an inherent risk and, to manage this risk, an agent must consider its trust of others in conjunction with its motivations in entering into, and continuing in, cooperative activity. The aim of this thesis is to develop a framework for motivated cooperation, focusing in particular on the motivational reasons an agent might have for cooperating, and how it can use the information it has about others (such as their capabilities and trustworthiness) to make informed judgements about the risk involved in cooperating.

The main body of this thesis can be decomposed into four parts. First, we introduce the issues associated with motivated cooperation, identify the outstanding problems, and discuss the key related work that gives a context to the thesis. Second, we present the *motivated* agent architecture, SENARA, which forms the foundation of our framework. Third, we introduce the framework itself, drawing out the details related to motivation and risk, and describing how this framework can be instantiated in particular applications. Finally, we conclude the thesis by considering the contributions it has made, and identifying potential areas for future work.

Chapter 1

Introduction

1.1 Introduction

Artificial intelligence (AI) is a relatively young field, having been established for little more than half a century. Early research in AI, around the 1960s and 1970s, was concerned with solving isolated problems such as storing information effectively for future processing (knowledge representation), and finding an appropriate sequence of actions to achieve a given task (planning). The next stage of AI research in the 1980s and 1990s saw expansion into many threads, encompassing neural networks, machine learning, computer vision, and many others. Until this point AI research had concentrated on the individual, but from the early 1990s onward researchers began to consider groups of entities. Indeed, one of the strongest areas of recent growth has been distributed problem solving (DPS), which is concerned with developing mechanisms for a collection of problem solving nodes, working together to perform a particular task. Nodes in a DPS system are typically based upon the earlier products of AI, in particular with respect to their local problem solving abilities. There are many examples of DPS systems, such as Smith's cooperating experts model [95], which is based around the Contract Net protocol [94], and Lesser and Corkill's Distributed Vehicle Monitoring Testbed [57] both of which have led to extensive further work.

Recent work in the area of artificial intelligence has seen significant development of earlier ideas on DPS. Problem solving nodes have become more sophisticated, both in terms of their capabilities and their reasoning power, and are now typically viewed as *agents* — independent entities, capable of acting in their environment according to their perceptions, based on their own decision-making processes and objectives [60]. Perhaps the single most significant development, however, is that agents typically have some degree of *autonomy*, in that they are able to guide their own behaviour and function effectively without outside intervention. Systems containing a group of such problem solving agents are called *multi-agent systems*, and form part of the now established subfield of AI, distributed artificial intelligence (DAI). However, DAI is still young, and comprises much diverse and sometimes contradictory work.

In this thesis we consider multi-agent systems, and in particular we develop a model of cooperation between autonomous agents. Cooperation is the foundation upon which multi-agent systems are built; groups of agents cooperate to achieve goals that they would not be able to achieve alone.

1.2 Agents

Over the last decade or so, the question of what constitutes an agent has been the subject of much discussion [34, 78], with various camps making different claims about what is required from an entity for it to be considered an agent. In this section we introduce the notion of agency, and the characteristics that agents typically exhibit, in particular the quality of autonomy. Two general uses of the term *agent* are distinguished by Wooldridge and Jennings [103], the first of which is a weak notion in which agents are viewed as having the following properties.

Autonomy: agents are able to operate without external intervention, and should have some degree of control over their own behaviour.

Social ability: agents are able to interact with others.

Reactivity: in a dynamic environment, agents perceive relevant changes in the environment and are able to react appropriately in a timely fashion.

Pro-activeness: in addition to responding to their environment, agents exhibit goal-directed behaviour and act under their own volition.

Secondly, Wooldridge and Jennings recognise that for some researchers the term *agent* imports a stronger notion so that, in addition to the properties outlined above, agents are conceived as intentional systems in terms of mentalistic notions, such as belief, knowledge, intention, and motivation [103]. Several proposed sets of mental components exist but the combination of *beliefs*, *desires* and *intentions* is arguably the most widespread [44] in terms of agent characterisation. Beliefs are those propositions about the environment and itself that an agent takes to be true. An agent's desires are the situations that it wishes to bring about, and its intentions are those desires to which it has committed to achieving.

In broad terms, the debate has recently settled, marked by the general acceptance of this weak definition of agency.

1.2.1 Autonomy

Autonomy is an important quality of an agent, and can be viewed along two complementary dimensions that are really two aspects of the same characteristic. Firstly, if an agent is autonomous it is able to act, at least to some extent, without external intervention either from a human user or another agent. Secondly, an autonomous agent is able to guide its own behaviour and act according to its own priorities, rather than being under the direct control of another. Although the importance of autonomy in agents is widely alluded to [1, 11, 45], there are few explicit detailed models, and autonomy does not form an explicit part of the common view of agents as intentional systems. Of those explicit models that do exist, some model autonomy through the provision of additional mental components (e.g. [61, 77]).

The outstanding problem, however, is integrating these additional components seamlessly with the agent's other mental components, such as its beliefs, desires, and intentions.

1.2.2 Other Characteristics

Despite the general acceptance of Wooldridge and Jennings' weak definition of agency¹, there are still a few small areas of debate left open. For example, there is no consensus over whether temporal continuity, learning, and mobility are required characteristics for agents [34]. Rosenschein and Genesereth identify benevolence as another important area of contention by noting that many early DAI systems make the *benevolent agent assumption* [85], by which the agents in a given domain have common or non-conflicting goals. A consequence of this is that agents are often assumed to aid one another through providing information and performing tasks requested of them. Rosenschein and Genesereth were amongst the first to argue that this is unrealistic in many real world situations, and in situations where agents are autonomous, since the behaviour of autonomous agents cannot be guaranteed. DAI is underpinned by the notion of cooperation, through which a collection of agents achieve their goals together; in the following section we introduce cooperation and its relation to agent autonomy.

1.3 Cooperation

A group of agents *cooperate* when they engage in a joint activity for which the actions of each are needed for a successful outcome, and where agents' actions are not under the direct control of another [101]. Individual agents have limited capabilities and resources, and consequently the tasks they are able to achieve are also limited. Cooperation allows agents to transcend these limitations and achieve tasks that they would not be able to achieve

¹Of course, while DAI remains a relatively young and active research area there will not be complete acceptance of such definitions.

alone. By pooling their resources and capabilities, a group of agents may be able to achieve many more tasks than any individual in the group. Consider the oft cited example of two agents wishing to move a heavy piece of furniture, where each agent does not have the strength to move it alone. By acting together and assisting each other they are able to move the object [67]. Similarly, in a situation where individual agents do not possess sufficient information to perform some task, then by cooperating and sharing information they are able to draw on the collective knowledge of the group members and perform the task.

1.3.1 Autonomous Cooperation

When autonomous agents cooperate in a dynamic environment, uncertain behaviour may arise as a result of their autonomy, or from changes in the environment. An agent acts according to its own individual high-level desires, which may change over time. Consequently, even though an agent might initially agree to contribute to achieving some objective through cooperation, there is no guarantee that its attitude will not change during the course of the interaction. If the agent's individual high-level desires change such that the reason for its cooperation is removed, then it is likely to stop cooperating, and not perform any additional actions in pursuit of the group objective. This can have repercussions for the other agents involved, since, if they are not aware that one of their number has ceased to cooperate, they may continue to act assuming that others are acting accordingly. Thus, if an action relies on a previous action that should have been performed by an agent that is no longer cooperating, this latter action will fail. In such a situation the group can either determine another course of action to complete the achievement of their goal, or they can concede failure. Indeed, there may be no choice but to allow their cooperation to fail, depending on the circumstances and the manner in which cooperation broke down. Regardless of whether the group later goes on to achieve its objective or concede failure, it has been negatively affected by the agent that left the group, since the members have invested time and effort in their contributions to the objective, only to have to reconsider their plans, or for the objective to fail.

Cooperation between autonomous agents, therefore, implies an element of risk since agents may rescind their cooperation at any point, as their desires change. In order for agents to make informed decisions about when to cooperate with others, some means of assessing and managing this risk is needed.

1.3.2 Agent Architectures for Cooperation

There is a plethora of architectures ranging from *deliberative* agents, which have some mechanism for constructing plans to achieve their goals (e.g. [5, 38]), to *reactive* agents where the emphasis is on the agent's ability to react in a timely manner to changes in its environment (e.g. [6, 66]). Since agents generally need both the problem solving abilities that arise from deliberation, and the ability to react quickly to their environment, a number of *hybrid* agent architectures have also been suggested (e.g. [30, 32]). Existing architectures are generally designed with a specific task in mind, and so will tend to emphasise some characteristic such as problem solving ability, reactivity, or autonomy, according to the requirements of the task. Where cooperation is investigated without reference to a specific agent architecture, any resulting model will be abstract and can only serve as a high-level specification, since it will not give details of how cooperation arises with respect to an agent's mental components. While the relationships and high-level mechanisms involved in cooperation can be investigated without reference to a particular agent architecture, we cannot look at how cooperation arises from individuals without knowing the details of their individual architecture. Any model of cooperation must, therefore, give consideration to the details of the architecture on which it is based; in particular it must account for how agents are made autonomous. Autonomous agents act according to their high-level desires, which provide the *reason* for their actions, and a model of cooperation must also account for how an autonomous agent can come to have appropriate reasons to cooperate. We discuss these issues within this thesis in Chapters 3, 4, and 6, where the former two chapters present our autonomous agent architecture, and the latter describes the reasons why such an agent might cooperate.

1.4 Aims and Principles

Several factors affect the nature of a cooperative interaction among a group of agents, including the details of the agent architecture, how autonomy is achieved, and whether agents have an appropriate mechanism for managing the risk arising from cooperation. Cooperation involves all of these, and a model of cooperation between autonomous agents must include them all. Existing models, however, typically do not sufficiently address these areas. In this thesis, we aim to provide a theory of cooperation that encapsulates all of the relevant factors, with a particular focus on those identified above, namely the need for autonomy and risk management. Additionally, this theory must not fall foul of the *benevolent agent assumption* where the agents in a system are fundamentally assumed to be helping each other, and will cooperate regardless of the benefit they themselves will receive from the cooperation. Rather, we assume that agents are fundamentally autonomous, and are self-interested. In summary, the aim of this thesis is to develop a principled theory of cooperation grounded in a specific architecture for a dynamic environment where agents are potentially unreliable. This can be instantiated more precisely as follows.

- We aim to construct a general framework within which to consider the general issues surrounding cooperation in multi-agent systems. This framework must identify the key areas of relevance for examination, the structure within which to organise them, and the broad-based mechanisms required. It should not be tied to particular domains or applications, and any mechanisms described must be generic and abstract, to apply across the spectrum of cooperative activity.
- Within this framework, we aim to develop the particular models and mechanisms needed for agents to establish and perform cooperative interactions. This can be regarded as instantiating the broad-based framework described above, tailoring abstract mechanisms to particular well-defined sub-areas. In so doing, we do not intend to restrict the applicability of the work, but to ensure that applicability is instead well-recognised, and that points of variance for different sub-areas may be identified with

potential alternative strategies fitting in.

In particular we aim to explicitly consider the development of mechanisms for the following stages in cooperation.

- Agents must be able to recognise when the potential for cooperation exists, and when it is the most appropriate way of achieving their goals.
 - Once an agent has determined that the best method to achieve its goal is through cooperation, it needs some mechanisms through which it can elicit assistance from others.
 - Conversely, when requested to offer assistance to another, an agent needs some means of determining whether to accede to the request.
 - Finally, assuming sufficient agents agree to assist, the group of agents concerned must be able to execute the required actions in a coordinated and synchronised fashion to achieve their objective.
-
- In line with the previous point, we aim to develop a prototypical implementation of the developed model of cooperation as a demonstration, and in order to perform experiments, to investigate, test and analyse the algorithms and mechanisms developed, to provide empirical understanding, and to provide indicative ways for the work to be used in practice. We view this implementation as of secondary importance, in that while it offers a useful demonstration the model, it not central to our analysis.
 - In considering cooperation of any kind, it is important to pay attention to the reasons why cooperation arises. These can constrain or bias cooperative activity significantly. In the development of the two tiers of analysis above (i.e. the framework and model), we aim to include an explicit recognition of the *reasons* an agent might have for entering into cooperation, through the notion of some internal desire or *motivation* for doing so. Where cooperation occurs as a result of, and is constrained by, agents' motivations, we say that it is *motivated cooperation* — it is motivated cooperation,

which is the specific focus of this thesis. Thus, we aim to provide an account of how motivations provide reasons for

- an agent deciding to request assistance for one of its own objectives;
 - an agent agreeing to cooperate with another in achieving an objective that might not be of *direct* benefit to itself; and
 - an agent continuing to cooperate until the objective is achieved, *or* deciding that cooperation is no longer in its best interest, and so terminating its involvement.
- Similarly, it is important to consider the consequences of agents' autonomy on the cooperative process. In particular, autonomy implies that agents follow their own individual high-level desires, and consequently, whether a given agent is cooperative or not is a direct function of them. If an agent's high-level desires change during cooperation, that agent may drop its involvement in cooperation in favour of some other activity, even if this is detrimental to the remaining agents. Thus, there is a risk of cooperative activity breaking down due to changes in agents' desires, potentially to the cost of those involved. We aim, therefore, to provide a mechanism for agents to assess and manage the risk of such situations occurring.

We intend the work contained in this thesis to be as general as possible, without over-complication, and easily applicable. In pursuit of these aims, there are a number of principles that we adopt and use to guide our work. These principles are based on those first articulated by Luck [59].

- Simplicity contributes to ease of development, evaluation and refinement. The vast amount of research in AI has led to an ever growing variety of tools, and methodologies for using those tools, of ever increasing complexity. Arguments for what has been called 'minimalist AI' suggest that there should be a limited range of tools and methodologies which should only be added to when they can be shown to be inadequate. This is based on the premise that advances are not made by increasing the

number or complexity of tools, but from a small range of simpler tools applied in useful ways. An important consequence of this approach is that it allows the merit of such simple tools and methodologies to be evaluated easily and the tools to be revised as appropriate. Thus, where existing solutions suffice for particular problems, we aim to use them in an effort to prevent duplication, to better relate our framework to existing work in the area, and to frame the work in a context (that already exists) in which it can be understood.

- Similarly, simplicity also counsels against the imposition of unnecessary constraints and for the adoption of more general and more applicable solutions. Despite the inevitable need to focus on details of domains and applications, the more general relevance in this research must be made clear. Our solution is twofold: we provide at least two levels of analysis through the general framework and its instantiation as a model, and we avoid making premature commitments to particular instantiated solutions wherever possible. In this way, we avoid contributing only in the narrowest of areas, and also avoid a level of abstraction that tends toward the meaningless.
- One of the most significant problems faced by researchers in the field of multi-agent systems, and which threatened to limit its development, relates to the vagueness and ambiguity of much early work. We recognise the need to be careful and precise in relation to our work, especially in this rapidly moving area, and aim to achieve that precision through the use of formal specification techniques. Thus, the agent architecture on which the theory is built and the theory itself, are formally specified to aid understanding, analysis, and critical comparison with other architectures and models.

1.5 Structure of Thesis

Before we begin to address the aims described above, in Chapter 2 we set the context for our work by considering related research. Since there is a wide body of work relating to agency, agent architectures, and cooperation, we select the most significant with respect to this thesis.

Chapters 3 and 4 describe the architecture on which we base our framework of motivated cooperation. The former of these introduces the mental components required of a motivated agent situated in a cooperative environment, while the latter details the control processes that act on these components. Then, in Chapter 5, we give an overview of the cooperative process, describing the stages it comprises — we discuss how the need for cooperation arises, and how it can be established. In this chapter we also introduce some of the key concepts that the remainder of the thesis relies on (specifically the notions of commitment and risk with respect to cooperation).

In Chapter 6 we describe in more detail the situations under which the need for cooperation arises and, in particular, we discuss the reasons an agent might have for choosing to achieve its goals cooperatively. Where an agent wishes to cooperate for the achievement of a particular goal, there are a number of steps that it must take to gain the assistance of others, and these are described in Chapter 7. Finally, Chapter 8 identifies the contributions made in this thesis, and areas of possible future extension.

Chapter 2

Related Work

2.1 Introduction

Motivated by the need for the work contained in this thesis to be generally applicable, it must be seen within the wider context of related research. There is, however, a vast array of related literature — too much to include an exhaustive summary within the constraints of this thesis. Furthermore, several other researchers have produced extensive reviews (e.g. [53, 72, 73, 78, 79, 103]) which, taken together, offer a broad coverage of the state of the art in multi-agent systems research. We avoid duplicating such work here, and instead introduce only the most directly relevant material. The objective of this chapter is to set the context for the subsequent chapters, rather than to provide a detailed description of specific theories and models and, as a consequence, we give a precis of the relevant work, and offer pointers to the appropriate literature. Four broad areas stand out for discussion as directly relevant to motivated cooperation: agency and the notions of autonomy and intention, agent architectures, theories of social agency, and existing multi-agent systems. Sections 2.2 and 2.3 introduce the notions of *agents*, *autonomy* and *intention*, and outline the prevailing views in these areas. In Section 2.4 we describe some of the more significant agent architectures arising from these views. The key theories of social agency

are discussed in Section 2.5, and Section 2.6 introduces selected multi-agent systems that these theories give rise to. Finally, Section 2.7 gives an overview of the stages involved in cooperation between autonomous agents.

2.2 Agency and Autonomy

In Chapter 1 we noted that most researchers concur with the view of an agent as an independent entity, capable of acting in its environment according to its perceptions, based upon its own decision-making processes and objectives. This corresponds to Wooldridge and Jennings' *weak notion* of an agent as an entity that possesses the properties of autonomy, social ability, reactivity, and proactiveness. Recall also, from Chapter 1, that Wooldridge and Jennings observe that some researchers use the term agent to import a stronger notion of *intentionality*, conceived in mentalistic terms, with notions such as belief and intention. We adopt this intentional view of agency, and in the remainder of this section we introduce the foundational work on which it is based. It should be noted that there are many other notions of agency, and corresponding architectures arising from them, of which the most significant are briefly introduced in the final part of Section 2.4.

In our view there are a number of fundamental characteristics to autonomy, as identified below.

- Autonomous agents follow their own desires, are not under the control of others, and are able to adjust their behaviour in response to their current situation.
- Autonomy allows agents to function effectively when situated in an environment that is unpredictable due to its complexity and dynamic nature. An agent's environment may change through the effects of others' actions, and it must be able to respond to the changes that occur. In particular, it must be able to recognise when the goals it is pursuing are no longer relevant, or when the course of action it is following needs modification in the light of change.

- Finally, in order to react appropriately to changes in the environment an agent must be able to generate and adopt new goals, according to the current situation, and autonomy provides a means to achieve this.

There are several theories of autonomy in agents, ranging from the philosophical and psychological which aim to further our understanding of autonomy in humans, to those that are firmly part of DAI. In relation to this thesis, and our investigation of motivated cooperation, the most significant models are those that discuss autonomy in relation to the notion of *motivation* and we introduce these in the remainder of this section.

2.2.1 Autonomy Through Motivations

According to Luck and d’Inverno, autonomous agents possess goals that are *generated* internally, rather than being *adopted* from an external source [61, 62]. These goals are generated from a set of *motivations*, which are high-level desires or preferences that characterise an agent’s purpose. In addition to causing the generation and subsequent adoption of goals, motivations also direct an agent’s reasoning and action.

An agent has a given set of motivations, each with a particular intensity, which may be variable dependent on the current situation. Motivations are represented as triples (m, v, b) where m is the kind of motivation, v its intensity (as a real number), and b a boolean which is *true* if the intensity is fixed, and false otherwise [63]. Luck and d’Inverno suggest that, in order to achieve action, a *threshold value* for intensity may be introduced, such that if the intensity of a particular motivation exceeds this threshold, action towards it is necessary.

Motivations are mitigated by agents selecting an action to achieve an existing goal, or by retrieving a goal from a repository of known goals. In order to retrieve goals, an agent must have some means of assessing the alternatives, and should select the set of goals that affords it the highest motivational benefit. If generating a goal would cause a conflict with an existing goal, then agents should only generate that goal (and remove the existing one) if the motivational value of doing so is greater than that of not doing so. Agents are therefore

assumed to have appropriate mechanisms for assessing the motivational value of generating, satisfying, and removing goals. However, Luck and d’Inverno do not give details of how to instantiate these mechanisms, since they are concerned with the development of a framework for motivated agency, rather than the development of a particular agent architecture.

2.2.2 Motivated Goal Creation

Norman and Long have a related view of *motivation* to that described above, which focuses on goal creation in agents [75, 76, 77]. They claim that agents situated in a realistic domain must be able to generate goals on the fly, limit the goals that are considered for action (in order to manage the reasoning overhead), and direct attention to the most appropriate goals for the current situation. Agents are given a set of *motives*, the function of which is to monitor the environment and the internal state of the agent, and ensure that a particular objective is served. Motives ensure that any significant changes to the environment or to the agent’s internal state are detected and acted upon, and are defined as a function that maps a set of beliefs to a set of *motivated goals*. These motivated goals are tuples containing a goal and a *motivation*, where a motivation is a heuristic function that maps beliefs to an intensity used to select between motivated goals. The final component that is needed for goal generation and goal activation is a set of *triggers*. A trigger takes a set of motivated goals and a set of beliefs and causes each goal whose motivation exceeds a certain threshold to be a candidate for consideration by a deliberative process that decides whether to adopt the goal. The resultant set of goals is then passed to a planner that directs action in pursuit of them.

Figure 2.1 illustrates the key parts of this motivated agent architecture. In particular, motivated goals are generated on the basis of an agent’s motives and beliefs, and are added to the set of goals considered for activation. If the intensity of the motivation associated with a goal is greater than some threshold (determined by the planner) then the goal is activated,

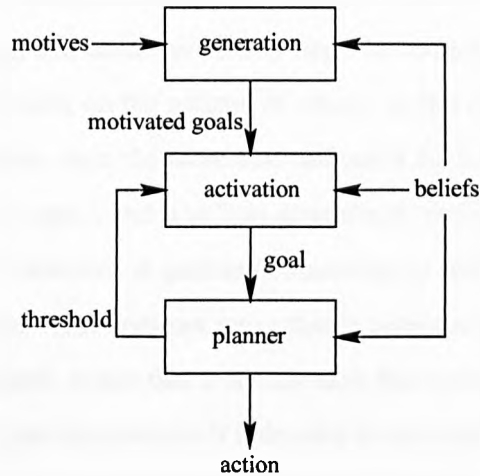


Figure 2.1: Norman and Long’s motivated agent architecture (from [77])

and passed to the planner, which determines how it should be achieved. Despite differences in terminology, Norman and Long’s view of autonomy is broadly analogous to Luck and d’Inverno’s — the goals an agent can generate are associated with some intensity based on its beliefs¹, and should this intensity exceed some threshold then the goal is adopted, or *generated*.

2.2.3 Autonomy as a Dependence Relationship

In collaboration with others, Castelfranchi [11, 14] proposes a view of autonomy as a relational notion, in that an agent is autonomous *for* a given action or goal if it is able to perform the action or achieve the goal without assistance, and is autonomous *from* some agent or resource if it can act without reliance on that agent or resource. This view comprises two distinct categories of autonomy: *executive* and *social*. Executive autonomy refers to an agent’s ability to follow its own initiative and preferences, and guide its own behaviour in

¹This association is indirect in Luck and d’Inverno’s approach, since motivations have an intensity, and for each motivation there is a particular set of goals that affords it motivational benefit.

terms of the goals it adopts and the actions it performs. In a multi-agent environment, autonomy is a social notion, and social autonomy has two complementary aspects. Firstly, the less an agent is *dependent* on the actions of others, or the resources they control, for the achievement of its goals, then the more autonomous it is. Secondly, if an agent is not dependent on others for its goals, and is able to generate its own goals according to its own desires, then it is *goal autonomous*. A goal autonomous agent should have complete control over which goals it adopts. This does not mean that it cannot adopt a goal on the basis of a request from another agent, rather that it should have the option of refusing to do so. In other words, an agent is goal autonomous if it decides to do something for its own reasons, regardless of whether or not this involves complying with others' requests.

Castelfranchi is not concerned with providing a model of autonomy at the level of the data structures and functions required, but with the investigation of the various types of autonomy that can be distinguished, and extending the notions of executive and social autonomy; thus Castelfranchi's work is more abstract than that of Luck and d'Inverno or Norman and Long.

In this thesis we accept the notions introduced in this section (and adopt Luck and d'Inverno's terminology). However, none of this work investigates how autonomy relates to cooperation amongst agents with respect to a particular architecture, in terms of accounting for the *reasons* an agent might have for entering into cooperation, and the processes through which cooperation arises from agents' behaviour, and as stated in Chapter 1 we aim to address this. Other models of autonomy exist, such as (e.g. [1, 45, 70, 92, 93]), but a discussion of these is beyond the scope of this chapter.

2.3 Individual Intentionality

In our everyday activities it is common for us to make use of such notions as belief, hope, desire and intention in reasoning about ourselves and others. As Wooldridge points out,

these mental states, or *intentional attitudes*, are part of a well established *folk psychology* in which human behaviour can be investigated and forecast [105]. *Intentional systems* are those whose behaviour can (at least sometimes) be explained and predicted by the ascription of intentional attitudes [22]. It is clear that agents can be viewed as intentional systems, and it is commonplace for researchers to couch descriptions of agents in terms of intentional attitudes. However, it should be noted that it is generally not claimed that intentional systems *must have* beliefs, desires, intentions, and so on, rather that *ascribing* such attitudes provides a useful abstraction for investigating agents, and for comprehending and managing their complexity. The exact composition of the set of attitudes that are required to explain and predict individual behaviour has been, and will almost certainly continue to be, the subject of debate for philosophers, cognitive scientists, and artificial intelligence researchers alike. However, many researchers agree that the notions of *belief*, *desire*, and *intention* are broadly sufficient (e.g. [2, 3, 15, 22, 51, 87]). Recall from the previous chapter that an agent's beliefs are those propositions about the environment and itself that it takes to be true, and its desires are the situations that it wishes to bring about in the environment. Intentions are more complex, and represent the goals to which an agent has *committed*.

Philosophers have often drawn a distinction between *future-directed* and *present-directed* intentions, where the former guide agents' planning and constrain their adoption of other intentions, while the latter function causally in producing behaviour [15]. For example, a future-directed intention may be to go to London tomorrow, while a present-directed intention may be the action of standing up now. It is the notion of future-directed intentions that we are concerned with in this thesis, since we are concerned with how agents guide their actions, form plans, and commit to the achievement of their goals. For clarity, we hereafter use the term *intention* to refer to future-directed intentions, unless explicitly specified otherwise.

2.3.1 The Need for Intention

An intention is a commitment to the achievement of a particular goal. Bratman [2, 3] offers three reasons why intentions are needed to explain and predict agent behaviour. Firstly, since agents are resource-bounded, and deliberation uses resources, there are limits to the extent of deliberation possible at the time of action. Agents cannot continually consider their competing goals and beliefs in deciding what action to take. Eventually an agent must settle on a particular goal, and establish a commitment towards achieving that goal, thereby balancing deliberation and acting. Secondly, by adopting intentions, an agent is able to coordinate its present and future actions. Once committed to achieving a given state, an agent can consider what to do after that state has been established based on the expectation that its commitment will result in it being achieved. Current intentions, therefore, constrain the future intentions an agent can adopt. Thirdly, intentions are a commitment to achieve a particular goal, without specifying how that goal should be achieved; as a result, intentions require deliberation, since an agent must determine how to achieve them. Moreover, since it is not rational for an agent to be committed to achieving conflicting goals, i.e. hold conflicting intentions, its current intentions establish standards of relevance against which future options can be judged.

From this description of the functional role of intentions, Bratman [3] identifies two important desirable properties. Firstly, to prevent agents becoming committed to conflicting goals, intentions should be consistent in that they should not conflict with each other or with the agent's beliefs. Secondly, intentions should have a degree of stability and resist being reconsidered or abandoned, but should not be completely irrevocable, otherwise an agent would not be able to adapt to changes in its environment.

2.3.2 Cohen and Levesque's Model of Intention

Cohen and Levesque offer one of the most influential models of intention, which forms the base for many theories relating to agents and, in particular, multi-agent cooperation. Build-

ing on the ideas of Bratman they add the following desirable properties of intention [15]. An autonomous agent should act upon its intentions and not regardless of them, adopt intentions it believes are possible, and forego those believed unachievable. It should commit to intentions, but not indefinitely, and discharge those intentions believed to have been satisfied. Finally, intentions should be modified when relevant beliefs change, and an agent should adopt subsidiary intentions in favour of achieving existing ones.

These properties give rise to a set of seven criteria that Cohen and Levesque claim must be satisfied by any reasonable theory of intention. The first three criteria represent the functional roles of intention, while the remaining four represent its desirable characteristics.

1. Intentions require deliberation since agents must determine ways to achieve them.
2. Intentions constrain the adoption of further intentions, since an agent should not adopt an intention that is inconsistent with existing ones.
3. Agents must track the success of their intentions, and be disposed to re-planning if their attempts to achieve their intentions fail.
4. Agents must believe their intentions are possible.
5. Agents must not believe that they will not bring about their intentions. If an agent believed it would not achieve its intention, then it would not be rational to plan past it; thus, without this property, agents would not be able to plan to do certain actions in the future based on achieving their existing intentions.
6. Under certain circumstances, agents must believe that they will bring about their intentions, meaning that they believe a situation will eventually arise in which they can bring about their intentions.
7. Agents need not intend all the expected side-effects of their intentions.

These criteria are accepted by many, but there are certain researchers who reject specific ones. In particular, the final criterion that agents need not intend the expected side effects

of their intentions has been the subject of debate. For example, Sidgwick [91] claims that if an agent has a particular intention and knows that achieving this intention will bring about some other side-effect in the environment, then we must say that the agent intends this side-effect. This debate, however, has largely been of a philosophical nature and is beyond the scope of this thesis.

Now, if, as desired, intentions are to be relatively persistent, but not completely irrevocable, the key question that arises is when it is acceptable for an agent to drop an intention. Cohen and Levesque answer this in their model of intention. Building on the seven criteria above, they propose a formal theory in which intention is modelled as a composite concept specifying the goal an agent has chosen, and how it is committed to achieving that goal [15, 16]. An agent's chosen goals (i.e. its intentions) are assumed to be *consistent*, achieved by requiring that an intention will not be adopted if it would be inconsistent with existing ones. (An agent's desires, or goals, may be inconsistent, but an agent cannot simultaneously *pursue* contradictory desires.) Cohen and Levesque also assume that an agent's beliefs can be incorrect and can be revised, and that the agent may drop its chosen goals before they have been achieved.

The foundation of their model is the notion of a *persistent goal*, where persistence is defined in terms of an internal commitment to a particular course of events or the achievement of a particular goal (see Table 2.1 for this definition). Building on this definition Cohen and Levesque introduce two forms of intention, depending on whether the object of the intention is an action or a goal. The difference between these forms is that when an agent intends to achieve a particular goal, it may not know how it will achieve it, as opposed to when it intends to perform a particular action, which is defined as a primitive level (again see Table 2.1 for the definition). Cohen and Levesque's notion of intention, despite having certain problems (which we address in Chapter 3), is one of the most widely used [44].

Definition 1 *An intention is a persistent goal to have knowingly performed an action, or to have knowingly performed a sequence of events after which a goal is achieved.*

Definition 2 *A persistent goal is a goal that is retained until an agent believes that it is satisfied, can never be satisfied, or is no longer justified, so that in all these cases it is irrelevant.*

Table 2.1: Cohen and Levesque’s definitions of *intentions* and *persistent goals*

2.4 BDI-based Architectures

The Belief-Desire-Intention (BDI) architecture [2, 5] is an abstract model, which forms the basis of numerous agent theories and systems and, since it is one of the earliest and simplest architectures, provides a useful reference for discussing other agent systems. Agents are based around the mental attitudes of beliefs, desires and intentions, which have been introduced above.

A BDI agent operates by reasoning about its current beliefs and desires, to determine one or more desires to make active. Once made active, these desires are committed to and added to the agent’s current intentions, which in turn define its behaviour. The agent then acts upon one of these intentions, and updates its beliefs and desires as appropriate. We describe this process in more detail below, where we consider instantiations of the BDI architecture.

Though this architecture specifies a generic control mechanism it is nevertheless abstract, and needs to be instantiated in detailed architectures for specific applications. There are many BDI-based systems (see [44] for a description of the most significant architectures), along with several logics for reasoning about BDI agents and their behaviour (for example [82, 83]). In the remainder of this section we describe the archetypal BDI-based systems, IRMA and PRS.

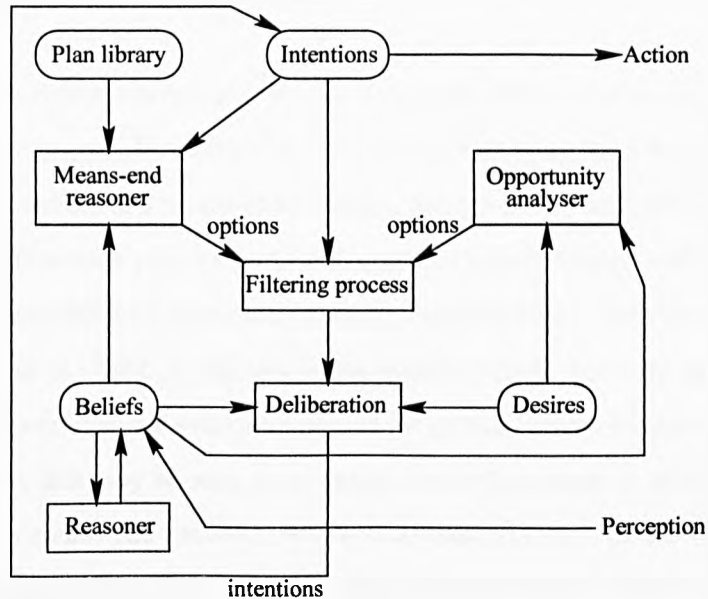


Figure 2.2: The IRMA architecture (from [5])

2.4.1 IRMA

IRMA (Intelligent Resource-bounded Machine Architecture) shown in Figure 2.2, is the earliest BDI implementation, in which information stores are shown as rounded boxes, and processes as rectangles. There are four information repositories in the IRMA architecture for storing beliefs, desires, intentions, and a plan library.

Beliefs An agent's beliefs represent the information it has about itself and its environment.

Desires The set of situations an agent wishes to bring about are its desires.

Intentions The goals to which the agent is committed to achieving correspond to its intentions, but rather than being stored as goals, they are stored as the plans that have been chosen to achieve those goals.

Plan library The plan library stores the set of plans from which an agent can select the

most appropriate to achieve its goals.

In IRMA, therefore, intentions represent a commitment to a particular course of action, rather than just to a goal. In addition to these repositories there are a number of processes, the most significant being a means-end reasoner, an opportunity analyser, a filtering mechanism, and a deliberation process. An IRMA agent's beliefs change with its environment, and different plans become applicable in those environments. The opportunity analyser proposes plans in the light of changes in the agent's beliefs that may lead to previously unexpected opportunities for satisfying desires (or provide means for avoiding unexpected problems). Plans that may be used as subplans in the elaboration of partial plans are also proposed by the means-end reasoner, which is invoked for each of the agent's intentions that contains partial plans. The plans proposed by the opportunity analyser and the means-end reasoner are passed through the filtering mechanism, which checks for compatibility with existing intentions, rejecting plans that are incompatible. There is also a *filter override mechanism* through which plans that are incompatible with existing ones can be deemed to have passed the filtering process and considered by the deliberation process. After filtering, the remaining plans are passed to the deliberation process, where they are considered and a subset committed to and added to the agent's intentions. Finally, the agent performs the plans that are specified by its intentions.

2.4.2 PRS

PRS (Procedural Reasoning System) [38] is another implemented architecture based on the BDI model. It is also based around four main repositories: a belief database, a goal stack, an intention stack and a plan (or Knowledge Area) library, as follows.

Belief database The belief database stores the system's beliefs about its environment, which are generally based on its perceptions, but may also comprise pre-compiled knowledge.

Goal stack The goals of the system correspond to its current desires, i.e. a set of situations the system wishes to be brought about, which are stored on the goal stack.

Intention stack The contents of the intention stack represent the goals to which the system is currently committed and it is these goals, or intentions, which control the actions of the system.

Plan library An agent's plan library is a collection of partial plans from which the agent can select the most appropriate for its current goal.

The system is controlled by a reasoner, known as the *interpreter*, which uses the belief database, goal stack, intention stack and plan library to determine behaviour. At any particular time, the system has a set of beliefs, stored in the belief database, and a set of active goals. Based on these, the interpreter determines a set of plans that are potentially relevant (in that they contribute to the active goals) and applicable based on the current beliefs. One of these plans is activated by placing it on top of the intention stack as a course of action the system is committed to executing. During the execution of a plan, the environment may change, new goals or subgoals can arise, and the interpreter monitors the environment in order to update its beliefs. The new goals are placed on the goal stack, at which point the interpreter checks to see which plans are relevant in the light of the updated beliefs and goals. This amounts to a continual process of interleaving planning and execution. Figure 2.3 illustrates the PRS architecture, and again information stores are represented as rounded boxes, and processes as rectangles.

Since plans (or knowledge areas) in PRS are not necessarily complete and can be partially elaborated, the top of the intention stack may be a partial plan, leading to partial execution. In turn, this can lead to new information being obtained and, as a result, PRS can plan with incomplete information. A consequence of the cycle of partial execution, updating the beliefs and choosing a new plan, is that PRS is reactive. This is because after each stage of execution any changes in the environment will be reflected by changes in the beliefs, which in turn influence the reasoning. A change in beliefs does not directly cause

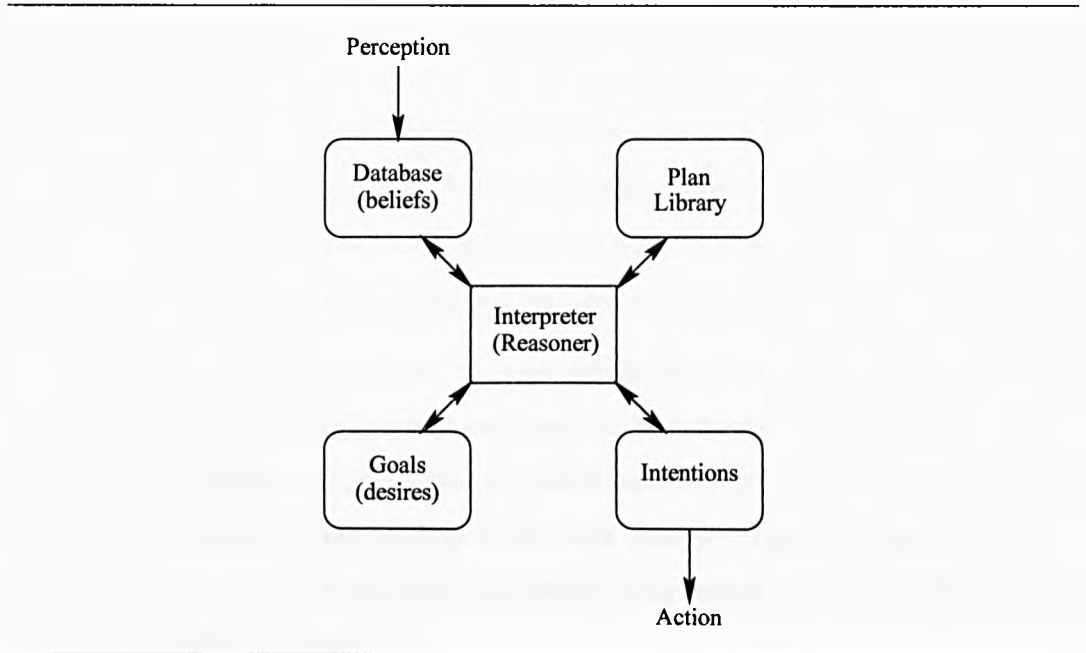


Figure 2.3: The PRS architecture (from [38])

a change in the current goals, and so the system continues to work towards its high level goals i.e. it is goal-oriented.

2.4.3 Alternative Architectures

Many other agent architectures exist, and there are a number of alternatives to the BDI approach, such as SOAR [56], TouringMachines [29, 30] and InterRRap [32, 74], of which the latter two can be categorised as hybrid systems that attempt to combine reactivity with a classical planning approach. TouringMachines has a layered architecture, comprising a reactive layer for responding to events in the environment, a layer for planning that constructs plans and focuses the agent's attention and, finally, a modelling layer for constructing information about others. Each of these layers is passed information from perceptions, and can cause actions through a control framework that mediates between the three layers.

InteRRap also has a layered architecture with behaviour-based, planning, and cooperative planning layers, roughly analogous to the layers in TouringMachines. Each of these layers is divided into sublayers containing knowledge bases and control units specifically for that layer. Thus, each layer contains the information and control structures needed at that level. For example, the behaviour-based layer contains knowledge about the environment and the control mechanisms for acting and perceiving in that environment.

Alternatively, some researchers, most notably Brooks, view agents from a primarily reactive viewpoint. Brooks [7, 9] eschews symbolic AI in favour of *situating*, in a complex environment, autonomous robots without explicit representations of their environment or reasoning ability. In his view, intelligent behaviour *emerges* from a *subsumption architecture* in which a hierarchy of behaviours for specific tasks compete with each other to obtain control of the robot (or agent) [8].

2.5 Social Intentionality

Cooperation underpins multi-agent systems in which individual agents must interact for the overall system to function effectively and perform tasks that otherwise might not be achieved, or at least not achieved as easily. Now, an agent's actions are determined by its intentions, regardless of whether these actions are cooperative or not. Thus, where a group of agents cooperate, their behaviour and therefore their cooperation, is at some level defined by their intentions. In this section we discuss the nature of the commitments required from a group of agents to act cooperatively, and introduce the notion of social intentionality. We begin by describing Cohen and Levesque's theory of group intention, which extends their individual model described in Section 2.3. In a cooperative environment, agents may be unable to achieve their goals without drawing on the knowledge or actions of another, i.e. agents may *depend* on each other. Castelfranchi's Social Power Theory is concerned with the influence of such dependencies, as we discuss later in this section. We end this section by introducing the notion of joint responsibility, which offers a practical approach to social

intentionality.

It is generally accepted that cooperation involves more than just the coordinated simultaneous actions of a group of individuals, and that it involves some form of group intention towards the cooperative activity (e.g. [4, 58, 100, 102]). Such a group intention cannot simply be a version of individual intention where the group is considered to be an agent itself, since group members may diverge in their beliefs, and there is no obvious coherent set of beliefs that correspond to the group's beliefs. This leads to problems in situations where an individual comes to believe that its intended goal is unachievable and so cannot remain committed to it, causing it to drop its intention. If such an individual is a member of a group, then the group should drop its intention. However, not all members of the group necessarily know that the goal is unachievable and that the intention should be dropped. Thus, a group's commitment to cooperate must incorporate some mechanism for an agent to become committed to informing others if it comes to believe that the intended goal is unachievable (or should be dropped for some other reason). Bratman [4] identifies this and a set of other requirements that he claims characterise a group's cooperative activity as follows.

- A degree of *mutual responsiveness* is needed, and each participant should guide its behaviour in response to others' intentions and actions.
- This mutual responsiveness should be driven by agents' commitment and so some form of commitment to joint activity, or *cooperative intention*, is necessary.
- Agent's should be committed to supporting the efforts of others when making their contributions to the cooperative activity, and so some form of *commitment to mutual support* is also required.
- Intentions should not be coerced, and any agents involved in a cooperative intention must have chosen to cooperate without force from others. This is not to say that agents should not try somehow to persuade others, merely that they should not use

force.

- Cooperative intentions should be common knowledge amongst the participants.

Existing work investigating what is required from a notion of cooperative intention can be divided into two categories. Firstly, there is the view that cooperative intention is irreducible, in that it *cannot* be reduced to a set of individual intentions and mutual beliefs [88]. Secondly, the more popular view is that cooperative intention is a combination of individual intentions, mutual beliefs, and a set of mechanisms describing how it should be maintained [84, 98].

Much of the existing work on cooperation in DAI is based on, or at least influenced by, the work of philosophers and psychologists such as Bratman [3] (as described above), Searle [88], and Tuomela and Miller [100].

Tuomela and Miller's model is one of the more influential models of commitment to cooperate, or *joint intention* — the notion of shared plans [42, 43], for example, is based upon it. In their model, for a group of agents to have a joint intention towards some group action there are three requirements. Firstly, each agent must have an intention to do its part of the action. Secondly, agents must believe that eventually suitable conditions will arise for them to successfully perform their group action and, finally, agents mutually believe that each agent will do its part as long as the others do likewise. However, as recognised by Cohen and Levesque, if one agent comes to privately believe the joint intention is no longer appropriate (i.e. is achieved, unachievable, or not justified) then, assuming the agent is rational, it must drop its intention [17, 19]. This, however, leaves the rest of the group abandoned, which is clearly undesirable. There is nothing in Tuomela and Miller's approach that requires an agent to stay committed to the group if it comes to have such a private belief.

Definition 3 *A joint intention is a joint persistent goal by a group to have knowingly performed an action, or to have knowingly performed a sequence of events after which a goal is achieved.*

Definition 4 *A joint persistent goal is a goal that is held, and mutually believed to be held, by two or more agents, such that until the goal is mutually believed to be irrelevant the agents have a corresponding weak goal.*

Definition 5 *An agent has a weak goal if either it has the goal, or believes the goal to be irrelevant and has the goal of making this mutually believed.*

Table 2.2: Cohen and Levesque’s definitions of *joint intentions* and *joint persistent goals*

2.5.1 Joint Intention Theory

According to Cohen and Levesque, a theory of joint intention must take into account that agents’ beliefs may be divergent since, if an individual comes to privately believe that the group’s goal is no longer appropriate and should be dropped, the other members of the group may not hold such a belief, and so do not know they should drop their goal. Therefore, a joint intention must include some mechanism through which an agent that privately comes to hold such a belief makes it known to the whole group, rather than simply abandoning the group action, leaving the others with inappropriate commitments. Cohen and Levesque introduce the notion of a weak goal to embody this commitment to informing others (see Table 2.2 for its definition).

In Section 2.3 we introduced Cohen and Levesque’s model of intention, upon which their model of cooperation is based by generalising their definitions to the case where two or more agents act as a team. A *team* is considered to be a group of agents having a shared objective and a shared mental state. Cohen and Levesque distinguish between a *shared* and *common* goal, in that a shared goal is with respect to a group who collectively have the goal, while a common goal occurs where a group of agents have the same individual goal and any one agent’s success is independent of others achieving their goal. Joint intention

is based on the premise that if a group member comes to privately believe that it should drop its intention it should adopt the (private) goal of making this known to the other group members *before* it can drop its own commitment. This is achieved through the concept of *mutual belief*, which is defined to be an infinite conjunction of beliefs about another's beliefs.

Based on the definition of a *weak goal*, Cohen and Levesque introduce the notion of a *joint persistent goal*, in which agents have appropriate mutual beliefs and weak goals towards some group goal (again see Table 2.2). For a group of agents to be jointly committed to a goal, each member of the group must initially be committed to the goal, and later believe that the other members have a corresponding weak goal. After the initial commitment, others can only be believed to have a weak goal (rather than the main goal itself) because they might have discovered the goal to be inappropriate and so have dropped their goal in favour of a secondary goal to establish mutual belief in the original goal's status. A joint intention is defined in turn, in Table 2.2, by generalising the definition of individual intention.

Under normal circumstances, a group's joint intention will eventually lead to one member of the group adopting the private goal to establish mutual belief (since the goal will eventually be considered inappropriate). This establishment of mutual belief can be viewed as the team overhead that arises from a joint intention and, moreover, Cohen and Levesque claim that joint intention will lead to communication between agents to establish mutual belief. An agent can therefore rely on others in the group to inform it when the goal is no longer appropriate.

2.5.2 Social Power Theory

In a series of collaborations with other researchers [10, 12, 14, 20], Castelfranchi presents a model of social action and cooperation stemming from *Social Power Theory*, which serves as the base for computational work known as *Social Dependence Networks* [89, 90]. Castel-

franchi approaches the subject from the point of view of psychology and sociology, rather than computer science *per se*, and as such many of his comments are concerned with the lack of DAI models to contribute to the understanding of human interaction. While our enterprise is not to develop a theory of human interaction, many of his observations are still relevant.

In Castelfranchi's view, cooperation implies a common goal shared by agents, so that *goal adoption* is necessary as a fundamental aspect of autonomous pro-social behaviour. Agents form a collective entity when they share a common goal, each agent is required to do its share of the common goal, and adopts an intention to do so.

A *common goal* is a goal with respect to which there is mutual dependence between agents [20]. An agent is said to *adopt* another's goal if it forms the goal that eventually the other agent should obtain its goal (where obtaining a goal implies the goal is eventually achieved and is believed to be achieved). Goal adoption may occur through influencing another, such as by offering a reward or issuing a threat, but it is assumed that agents cannot directly modify another's goals, and instead can communicate with them in an attempt to change their mental state. Also, an autonomous agent will only adopt a goal if it is useful with respect to fulfilling its desires. The notion of *social power* is key to the way in which agents get others to adopt their goals.

- An agent is said to have the *power* of a goal if it can ultimately achieve it.
- An agent *depends* on another for a goal if it does not have the power of achieving it and the other agent does; or if it has the power of the goal unless the other agent prevents it.
- An agent is said to have *power over* another agent for a goal if the other agent depends on it for that goal.
- An agent is said to have *power to influence* another for a goal, if it can perform some action that makes the other agent have the goal as a goal of its own.

These notions of social power form the base for Sichman and Demazeau's computational model of Social Dependence Networks, which are concerned with social reasoning of autonomous agents [90]. Each agent is assumed to have an *external description* of all the agents in the group (including itself), which contains details of their goals, actions, resources, and plans. These descriptions correspond to the agent's beliefs about others and, since beliefs are not guaranteed to be accurate, can be incorrect or incomplete.

The reasoning mechanism proposed relies on the following three types of autonomy.

1. An agent is *a-autonomous* with respect to a goal and a set of plans if it has the goal, a plan (in that set of plans) to achieve it, and all actions in that plan are within the agent's capabilities.
2. Similarly, an agent is *r-autonomous* if it controls all the resources required to execute the plan.
3. Finally, an agent is *s-autonomous* if it is both a-autonomous and r-autonomous.

If an agent is not autonomous for a given goal, it may depend on others. Corresponding definitions of *a-dependence*, *r-dependence*, *s-dependence* are given for dependence on an action, resource, or both. Using these notions of dependence and its descriptions of others an agent can construct a *dependence network* to represent all of its action and resource dependencies regarding others. These dependencies can be used in the agent's reasoning process, in particular to identify dependence situations. A key assumption here is that different agents' external descriptions are identical [90]. Based on this assumption an agent is said to *locally believe* a given dependence if it uses its own plans when reasoning about others. If it uses its own plans and those of others, there is said to be a *mutually believed* dependence between them. Using these notions it is possible to describe a number of other possible situations such as locally believed independence, mutual dependence, etc. In more recent work, Sichman and Demazeau extend Social Dependence Networks to include inconsistencies between agents, i.e. difference in agents' external descriptions [89].

2.5.3 Commitments and Conventions

Wooldridge and Jennings (with others) [48, 52, 102] attempt to begin to bridge the gap between theory and practice in DAI through the notion of *joint responsibility*, which is a mental and behavioural state that they claim captures and formalises many of the intuitive underpinnings of cooperative problem-solving. In their view a practically applicable theory of social activity among autonomous agents must address how team activity should be initiated, how to assemble a team, how to plan and distribute work, how to behave once team activity is initiated, and how to complete the team activity. A framework for cooperation is developed based on joint responsibility, which aims to address these aspects. The mechanisms on which cooperative interactions are based can be described in terms of *commitments* and *conventions* [49]. A commitment is a pledge or promise to undertake a specified course of action, and a convention is a means of monitoring commitments in changing circumstances. Conventions specify the conditions under which a commitment might be abandoned, and how an agent should behave in such a circumstance.

A common objective is not sufficient for realizing a collective goal — agents must agree upon a means of achieving it. According to Wooldridge and Jennings much of the other existing work on team activity has concentrated on joint intentionality in terms of goals, and has not considered how such goals will be achieved. The first step to achieving joint action is for a group of agents to have a common objective, or intention, that can only be achieved through collaboration (where intention is taken to be a goal *without* a specified means of achieving it). Agents can then form a *commitment* to this objective by forming a joint persistent goal (in the sense of Cohen and Levesque). However, a joint persistent goal does not specify how to achieve the objective. Wooldridge and Jennings suggest that agents should agree a strategy by which the objective will be achieved, and then develop and agree on a plan to achieve the common goal. Their framework is not concerned with the mechanisms for constructing the common plan, rather that agents agree in principle that such a plan is needed to achieve the objective.

The idea that joint action requires agents to agree to a common plan can be expressed implicitly through the definition of agents' roles, or explicitly in the definition of joint intention. The notion of commitment to a plan defines how an agent should behave once a plan has been developed — under what conditions it should follow the agreed plan, and how it should behave if it is no longer rational to do so, i.e. the *conventions* it should follow. It would be irrational for an agent to remain committed to a plan if it is invalid, unattainable, or violated, or if the objective is already achieved or another team member is no longer committed. If an agent comes to believe that it is irrational to remain committed to the joint solution, it should become committed to informing other team members.

2.6 Multi-Agent Systems and Architectures

In this section we briefly review some of the most significant multi-agent systems with respect to the development of our framework. Firstly, we introduce GRATE*, which is a BDI-based architecture specifically designed for cooperation. We then introduce two models of cooperation, Planned Team Activity and STEAM, both of which are based on the notion of joint intentions, as described above.

2.6.1 GRATE*

GRATE* is a layered BDI-based architecture with the addition of joint intentions (as described in Section 2.5) specifically intended for multi-agent environments where cooperation is important [50]. Agents are divided into two layers: a cooperative layer and a domain-specific layer. The function of the domain-specific layer is to determine how to achieve the agent's tasks, as defined by its desires or objectives. The cooperation layer operates above the domain layer, and is given the role of choosing the tasks that should be achieved locally and those that require cooperation, as well as ensuring that the agent's actions are coordinated with other agents. Three key components provide control in the cooperation layer: a

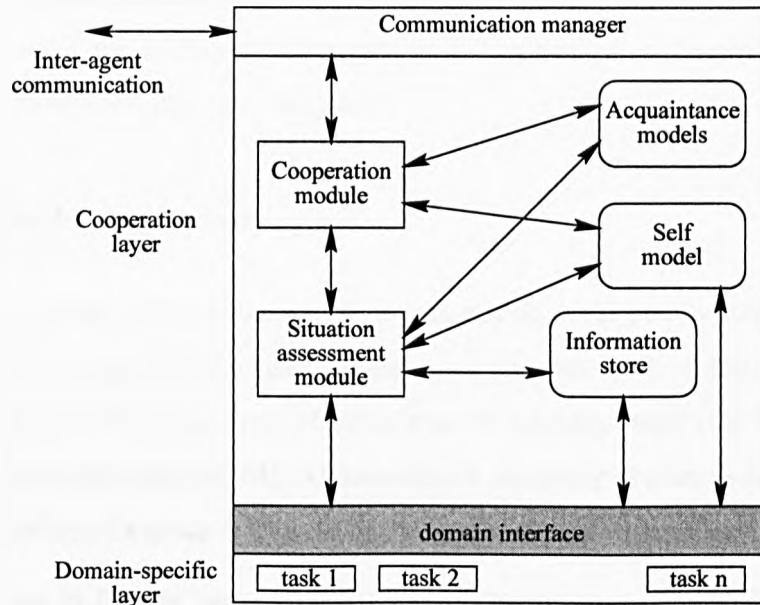


Figure 2.4: The GRATE* architecture for cooperation (based on [50])

domain-interface module that connects the cooperation layer to the domain-specific layer, a situation-assessment module, and a cooperation module.

Changes in the environment are represented by events, which are monitored by the situation-assessment module to determine whether a new objective is needed, to find a means for achieving them, and to check which of them require cooperation. The cooperation module determines potential participants in achieving objectives that require cooperation, attempts to form a joint intention with them, and oversees execution of these joint intentions.

The components of the GRATE* architecture are shown in Figure 2.4, which shows the two layers, along with the domain-interface, situation-assessment, and cooperation modules (in boxes). The figure also shows the information repositories (in rounded boxes) that these modules require, namely, acquaintance and self models that store information about

others and the agent itself, and a general information store for other information. Finally, the figure includes the additional component of a communication manager, to facilitate communication between the agent and others.

2.6.2 Planned Team Activity

Planned team activity (PTA) is the model of cooperation developed by Kinny *et al.* [55] for cooperation among BDI-like agents, building on previous work on BDI agents where agents are supplied with a repository of partial plans in advance, rather than being required to plan from first principles [38, 84]. An individual's repository of plans is its *plan library*, and the plan library of a group is taken to be the intersection of its members' plan libraries.

Cooperation in PTA is based upon the group mental states of mutual beliefs, joint goals, and joint intentions, which are similar to the notions introduced by Cohen and Levesque [18]. Distinct from Cohen and Levesque's notion, however, is that joint intention in PTA is a commitment to a joint plan or action, rather than to a goal. Consequently, a group of agents that have a joint intention also have a commitment to a common course of action, rather than just to a common goal, thereby avoiding the problem of different agents being committed to incompatible plans to achieve a jointly intended goal.

Cooperation requires joint commitment to a common course of action, together with coordination and synchronisation of that action, which is achieved in PTA through communicative actions. When an agent has a goal that it is unable to achieve alone, it takes on the role of team leader and attempts to form a team. The first step is to choose a set of agents with whom it can cooperate and to request their assistance. Once the team agrees to cooperate and a joint goal has been formed, the team must choose a plan to achieve its goal. PTA uses a centralised approach where the team leader chooses a plan and informs the rest of the team of the joint goal, the plan to achieve it, and an assignment of agents to actions (or *role assignment*) in that plan.

2.6.3 STEAM

STEAM (Shell for TEAMwork) [97, 98, 99] is another implemented model of cooperation founded upon Cohen and Levesque's notion of joint intentions. Cooperation in STEAM is based upon agents building a partial hierarchy of joint and individual intentions, and beliefs about others' intentions.

This hierarchy is designed to parallel Grosz and Kraus' notion of partial shared plans [42]. To achieve coherent cooperation, team members must follow a common approach to achieving their joint intentions. Grosz and Kraus' SharedPlan theory requires that agents have a mutual belief in a common plan, and shared plans for the individual steps in that plan. STEAM parallels this, in requiring that agents have a mutual belief in a common plan, and joint intentions for the individual steps in that plan, leading to a recursive hierarchy of joint intentions that ensure team coherence. For each step contained in a jointly intended plan the team must form a joint intention to execute it, and so on recursively. Such joint intention hierarchies can evolve dynamically, as partial plans are elaborated. A result of the commitment implied by joint intentions, is that team members track the subteam's joint intention in order to monitor the state of the team activity.

2.7 Stages in Cooperative Problem Solving

Leading on from the work described in Section 2.5, Wooldridge and Jennings [102, 104] offer one of the few formalised models of cooperation. Their model draws on the notions of *commitments* and *conventions*, and describes cooperation in these terms. Before describing their model of cooperation, it is necessary to introduce some of the concepts and assumptions on which it is based. Firstly, the mutual mental states of joint beliefs and joint goals are used to describe the attitudes of a group. Mutual belief is taken to be the usual infinite recursion of beliefs about others' beliefs, and joint goals are based on the corresponding individual goals, along with a belief about others holding the same goal. Wooldridge and

Jennings recognise that such mutual mental states are idealised and are typically not realisable in real systems. However, they assert that such mental states provide a useful tool for understanding cooperation between agents.

The primary mental attitude that is used in the development of the model is that of joint commitment (or joint intention). Wooldridge and Jennings give a definition that is broader than Cohen and Levesque's and is defined in terms of commitments and conventions. Indeed, they recast Cohen and Levesque's definition in their own terms, by describing the appropriate commitments and conventions that are otherwise implicit. A convention is represented as a tuple comprising a re-evaluation condition and a goal such that, if an agent comes to believe the re-evaluation condition it should adopt the corresponding goal.

A group is defined to have a *joint commitment* to a goal, with respect to some motivation, precondition, and convention if and only if,

- the precondition is initially satisfied, and
- until a termination condition is satisfied (as defined by the convention) every agent in the group either has the appropriate goal or believes the re-evaluation condition of the convention is satisfied, and has the goal defined by that convention.

The process of cooperation is divided into four phases, as follows.

Recognition Cooperation begins with an individual agent recognising the potential for cooperation.

Team formation The agent that recognised the potential for cooperation requests assistance from others and, if successful, obtains a joint commitment from the group of agents that agree.

Plan formation The agents that have the joint commitment attempt to negotiate a mutually acceptable plan to achieve their goal.

Team action The agreed plan is executed by the agents.

2.8 Summary

In this chapter we have introduced the context for the remainder of the thesis; we have introduced both the notions of agency and cooperation. In particular we have discussed motivations and intentions, and presented the abstract BDI architecture, upon which we build. We have discussed social intentionality, and Cohen and Levesque’s extension of individual intention to be a group notion.

Each of the theories and architectures described in this chapter are valuable, and make useful contributions that we incorporate into our framework. However, a number of issues are unaddressed in existing work, in particular the *reasons* why agents perform certain actions are often not accounted for. Furthermore, the relation of these reasons, or motivations, to cooperation in terms of the reasons why agents cooperate and the way in which they manage the cooperative process are not sufficiently explored. Similarly, the *risk* that arises from cooperation with autonomous agents is not considered. It is our aim to address these issues in the following chapters, in our framework for motivated cooperation.

Our view of autonomy, which is described in Chapters 3 and 4, is based upon Luck and d’Inverno’s model. However, we instantiate details that are abstract in their model, such as the mechanisms for assessing the motivational value of a particular situation, goal, or plan. In Section 2.2 we also introduced Norman and Long’s view of motivation, which is broadly compatible with Luck and d’Inverno’s. The terminology differs between the two views, however, and we adopt that given by Luck and d’Inverno’s and base our approach upon theirs. Castelfranchi’s work on autonomy is concerned with a general investigation of autonomy and how the dependencies between agents can be used to account for interactions. Unlike us, Castelfranchi is not concerned with the role an individual’s autonomy plays with respect to cooperation. Therefore, while we concur with Castelfranchi’s observations we do not make further use of them in the development of our framework.

In Section 2.3 we described the notion of intentionality, and introduced Cohen and Levesque’s widely accepted view of intention. The agent architecture that we describe in

the following two chapters is, in part, based upon Cohen and Levesques view of intention. There is, however, one key difference since their model is focused on a commitment to a particular goal, whereas we are concerned with commitment to a particular course of action in pursuit of a goal (as we describe in Section 3.12).

The BDI architecture introduced in Section 2.4 forms the base of our agent architecture, which in turn is the foundation upon which our framework is built. We introduced the archetypical instantiations of the BDI model in the form of IRMA and PRS, and our architecture broadly corresponds to these, with one very significant difference, namely the incorporation of motivations. We provide the mechanisms required for an agent to be fully autonomous and yet cooperate with others, through the additional mental component of motivation.

It is widely accepted that cooperation amongst individuals involves some form of group commitment, and Cohen and Levesque's notion of joint intention described in Section 2.5 is used by many researchers as the base for further work. We are no exception in that the precise nature of group commitment in our framework is based upon Cohen and Levesque's notions, as we discuss in Chapter 5. Our framework also utilises Wooldridge and Jennings' notions of commitment and conventions, and (again in Chapter 5) we extend their formulation of joint intention to include the key additional factor of motivation.

The multi-agent architectures introduced in this chapter do not consider motivations or risk, and in this respect our framework is fundamentally different to them. However, regardless of how cooperation arises, the actions involved must be synchronised and ordered, and we use the mechanisms for this defined by PTA.

Finally, the framework of cooperation contained within this thesis can be seen as instantiating Wooldridge and Jennings' stages as described in Section 2.7, and we view Chapters 5 to 7 as providing this instantiation.

Chapter 3

Motivated BDI Agents

3.1 Introduction

Early work on cooperation in DAI was concerned with non-autonomous agents, and the mechanisms used to achieve cooperation often presupposed benevolence on their behalf. Additionally, it has often been assumed that some, possibly external, entity exists that has knowledge of the complete system, and which is given both the ability and the authority to dictate the actions of agents in terms of their interactions with others. In our work, however, we are concerned with autonomous agents that have complete control of their own actions and cannot have their cooperative activity externally controlled. Instead, cooperation between a group of autonomous agents must arise from the mental attitudes of the individuals concerned. Typically, one member of the group has an objective that it is unable to achieve alone and attempts to gain cooperation from other agents which, in turn, will only enter into a cooperative interaction if it is of benefit to themselves as individuals. A practically applicable theory of cooperation, therefore, must provide details both of how agents can assess when they require assistance in achieving their objectives, and also when it is of benefit to themselves to enter into a particular cooperative interaction in pursuit of another's objective. The details of these mechanisms depend on the specific architecture of the agents

concerned, since behaviour is determined by agents' mental components. Thus, as we argued in Chapter 1, it is important for us to have a full understanding of the specific nature of the agent architecture upon which we are grounding the development of a framework for cooperative interaction.

In this and the following chapter, before focusing on the issues involved in cooperation, we present the agent architecture, which we call SENARA, on which we base our later model. SENARA is centred around the BDI abstract architecture [5] discussed in Chapter 2, and draws upon existing work on achieving agent autonomy through the ascription of motivations (e.g. [61, 62, 76, 77]). An agent architecture can be seen as containing two related parts: the mental components it comprises, and the control mechanisms that act upon those components. Whether or not a particular architecture is intended to be situated in a cooperative environment influences the set of mental components required. If an agent is to be cooperative, then it will typically require some means of representing actions or plans that involve others and have some model of others' capabilities, reliability etc. These requirements are relatively easy to incorporate into SENARA. Cooperation also has a significant influence on the nature of the control mechanisms that guide an agent's behaviour, since cooperation must arise from an agent's control mechanisms and there is no external force imposing cooperation.

In this chapter we introduce the mental components that SENARA contains, which are those defined in the BDI model (as discussed in Section 2.4), with the addition of motivations to achieve autonomy. We also introduce a few small extensions to the BDI model to allow for cooperation; in particular, joint and concurrent actions as described in Section 3.10. In the following chapter, we describe the control mechanisms that act on these mental components.

3.2 The Z Specification Language

In keeping with the principles outlined in Chapter 1, we aim to ensure clarity and precision in our description of the architecture through the use of formal specification. There are many specification languages that we might use, ranging from relatively abstract but expressive logics, to more implementation-oriented approaches. In this thesis we use the Z notation [96] to formalise SENARA’s components, since we consider it sufficiently expressive to represent the concepts we discuss, and close enough to the level of implementation to ensure that an implementation of the specification is essentially a programming exercise of creating the appropriate data types and interactions. Z is also widely used generally, and increasingly so in AI (e.g. [21, 39, 61, 71]). In this section we give a brief introduction to Z, borrowing heavily from that provided by d’Inverno and Luck in [25], and then in remaining sections describe SENARA’s mental components.

The Z specification language is based on set theory and first order predicate logic, along with the additional concept of a *schema type*. Schemas comprise two parts: an upper *declarative* part that defines a set of variables and their types, and a lower *predicate* part that defines the relationships between, and the constraints on, these variables. For example, the following schema contains two variables, x and y , both natural numbers, such that y is defined to be the square of x .

<i>Example</i>
$x, y : \mathbb{N}$
$y = x * x$

The type of a Z schema can be thought of as the Cartesian product of its variables (in no particular order) constrained by the predicates. Any given variable in a schema can be referred to in Z by giving the schema name and variable name in the form *schema_name.variable_name*. Thus, with respect to the above schema, *Example.x* refers to the variable x . To facilitate modularity and decomposition in specifications, a schema can include other schemas and inherit the variables and predicates defined therein. There are two types of

schema inclusion, Delta (Δ) inclusion, in which the variables of the included schema can be changed, allowing the specification of operations, and Xi (Ξ) inclusion where the included schema is unchanged. A schema is included using one of these conventions using the notation $\Delta schema_name$ (or $\Xi schema_name$) in the declarative part.

If we wish to introduce a new type, without specifying its details, we are able to introduce it as a *given set*. For example, we might introduce the set of all agent identifiers using the following given set.

$$[AgentID]$$

We can now define a variable to be of type agent identifier, a set of identifiers, or an ordered pair as $x : AgentID$, $x : \mathbb{P} AgentID$, and $x : AgentID \times AgentID$ respectively.

A *relation* between two types, a *source* and a *target* type, is defined to be a set of ordered pairs $\mathbb{P}(X \times Y)$ for source and target types X and Y . If there is no element in the source type that is related to more than one element in the target type, the relation is a *function*. A function is *partial* if not all elements in the source type are related, and is *total* if all elements of the source type are related. The *domain* of a relation, or function, is the particular set of elements of the source type that are related. Correspondingly, the *range* is the set of elements in the target type that are related. A *sequence* is a particular type of function whose domain is the set of natural numbers (from 1 to the number of elements in the sequence), and range is the set of elements in the sequence. For example, the following relation defines a sequence of three agent identifiers (although we would typically write such a sequence as $\langle agent1, agent2, agent3 \rangle$).

$$agentIDseq = \{(1, agent1), (2, agent2), (3, agent3)\}$$

A set of values can be constructed using set comprehension where, for example, we can specify the set of squares of natural numbers between 5 and 10 as $\{n : \mathbb{N} \mid n \geq 5 \wedge n \leq 10 \bullet n * n\}$. Predicates in Z can be written using the usual universal and existential quantification operators. For example, we can write that the squares of the natural numbers

between 5 and 10 are between 25 and 100 as follows.

$$\forall n : \mathbb{N} \mid (n \geq 5) \wedge (n \leq 10) \bullet (n * n \geq 25) \wedge (n * n \leq 100)$$

If we do not need to constrain the variable being quantified, then we can omit the constraint part of the predicate (i.e. the expressions between “|” and “•”). For example, we can write that all natural numbers are greater than zero, as $\forall n : \mathbb{N} \bullet n > 0$.

A summary of the Z notation is shown in Table 3.1. We do not, however, discuss the details of the language further in this thesis, instead we refer the reader to one of the many texts on the subject, such as [80, 106].

The specification contained in this and the following chapter is based upon the work of Luck and d’Inverno, who propose a formal model of agency and autonomy [61, 62]. In their work, an agent’s autonomy is characterised by the ability to generate goals, and select which goal is of the most benefit to the agent *individually*. We follow this approach, and base our notion of autonomous agency upon it. However, Luck and d’Inverno’s work is concerned with a discussion of autonomy and motivation rather than with the development of a particular agent architecture. We view the architecture presented in this and the following chapter as an instantiation of their work, with the required details filled in. It should be noted that there are generally several, equally valid, ways in which we could specify a particular notion in Z , without changing the *meaning* of the specification. Our criteria for choosing one approach over another is how well it integrates into the specification as a whole, the ease of understanding and the extent to which it lends itself to implementation.

3.3 Overview of SENARA

The key mental components of a SENARA agent are beliefs, goals, intentions, motivations, and a plan library. Recall from Chapter 1 that beliefs are the propositions about the environment and itself that an agent takes to be true. Goals are the situations that an agent wishes to bring about, intentions are the goals to which it has committed to achieving, and moti-

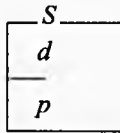

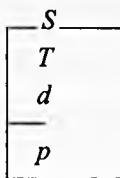
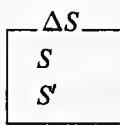
Definitions and declarations		Sequences	
a, b	Identifiers	$\text{seq } A$	Set of finite sequences
p, q	Predicates	$\text{seq}_1 A$	Non-empty set of sequences
s, t	Sequences	$\langle \rangle$	Empty sequence
x, y	Expressions	$\langle x, y, \dots \rangle$	Sequence
A, B	Sets	$s \hat{\ } t$	Sequence concatenation
R, S	Relations	$\text{head } s$	First element of sequence
$d; e$	Declarations	$\text{tail } s$	All but first element
$a == x$	Abbreviated definition	Schema notation	
$[a]$	Given set		Vertical schema
$A ::= b \langle\langle B \rangle\rangle$ $\quad c \langle\langle C \rangle\rangle$	Free type declaration		Axiomatic definition
Logic			Schema inclusion
$\neg p$	Logical negation		Operation schema
$p \wedge q$	Logical conjunction	$z.a$	Component selection
$p \vee q$	Logical disjunction	Conventions	
$p \Rightarrow q$	Logical implication	$a?$	Input to an operation
$p \Leftrightarrow q$	Logical equivalence	a	State component before operation
$\forall X \bullet q$	Universal quantification	a'	State component after operation
$\exists X \bullet q$	Existential quantification	S	State schema before operation
Sets		S'	State schema after operation
$x \in y$	Set membership	ΔS	Change of state ($S \wedge S'$)
\emptyset	Empty set	$\exists S$	No change of state
$A \subseteq B$	Set inclusion		
$\{x, y, \dots\}$	Set of elements		
(x, y, \dots)	Ordered tuple		
$A \times B \times \dots$	Cartesian product		
$\mathbb{P}A$	Power set		
$\mathbb{P}_1 A$	Non-empty power set		
$A \cap B$	Set intersection		
$A \cup B$	Set union		
$A \setminus B$	Set difference		
$\bigcup A$	Generalised union		
$\#A$	Size of a finite set		
$\{d; e \dots \mid p \bullet x\}$	Set Comprehension		
Relations and Functions			
$A \leftrightarrow B$	Relation		
$\text{dom } R$	Domain of a relation		
$\text{ran } R$	Range of a relation		
$A \mapsto B$	Partial function		
$A \rightarrow B$	Total function		

Table 3.1: Summary of the Z notation (taken from [25])

vations are the high-level desires that guide its behaviour. A control cycle brings together these mental components, which we describe in detail in following chapter, but first we give a brief overview in this section.

Firstly, an agent perceives its environment and updates its beliefs, since in order to act appropriately in a given situation it needs information about that situation. In the light of these new beliefs the importance of the current situation to each of its motivations is assessed, and those motivations to which the situation is important are made *active*. According to these active motivations the agent generates a set of goals, and adds them to its existing goals. The agent then selects one of these goals to work on according to its motivations, and commits to performing a particular course of action (or plan) for the achievement of this goal by forming an intention (unless it is already committed to its achievement). Finally, the agent acts towards the intention that is currently considered the most important, based on the agent's motivation. An overview of the architecture is illustrated in Figure 3.1, in which rounded boxes represent the mental components, the central box contains the control mechanisms, and solid and dashed arrows represent the flow of information and control respectively.

The SENARA architecture can be compared with the BDI-based IRMA and PRS architectures described in Sections 2.4.1 and 2.4.2, and illustrated in Figures 2.2 and 2.3 respectively. The overall form of SENARA is broadly similar to these architectures, with beliefs determining the goals to pursue, which in turn determine the intentions to adopt and the actions to perform. The key different between SENARA and IRMA or PRS is the addition of motivations. In the remainder of this chapter we describe in detail the mental components of a SENARA agent, and then in the following chapter the control cycle that acts upon them.

In the specification of SENARA that follows we closely follow the work of Luck and d'Inverno, who have also provided formal Z specifications of AgentSpeak(L) and dMARS, both of which are more recent incarnations of the PRS architecture to which SENARA is

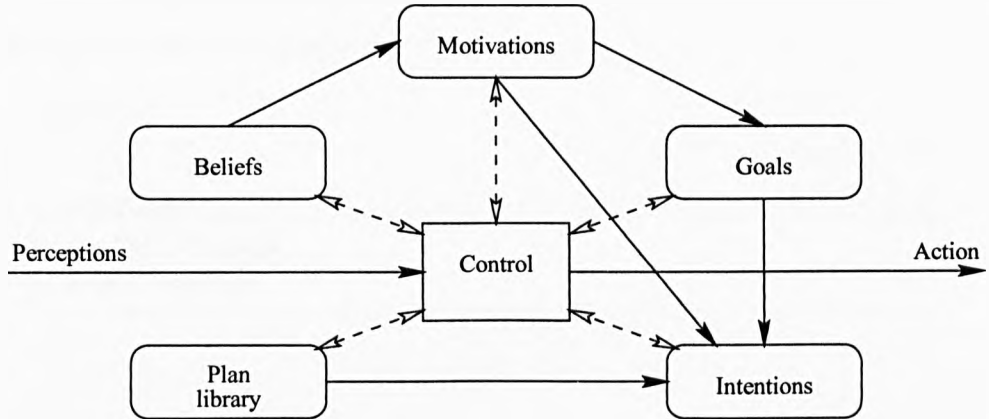


Figure 3.1: Overview of the SENARA architecture

closely related [24, 25]. Where appropriate in the following sections we utilise their specifications in our work. In particular our definitions of beliefs, goals and intentions are based on Luck and d’Inverno’s specifications.

3.4 Primitives

In this section we define the primitive types that are required to build the formal model of the SENARA agent architecture. We begin by introducing given sets to represent all constants and variables, denoted by *Const* and *Var* respectively, and define a *Term* as being either a constant or a variable.

$$[Const, Var]$$

$$Term ::= const\langle Const \rangle \\ | var\langle Var \rangle$$

The set of all possible predicate symbols is also represented by a given set denoted by *PredSym*, while a *Predicate* is a predicate symbol followed by a possibly empty sequence

of terms. For the purposes of specification we are not concerned with the contents of these given sets, since we can use them directly.

[*PredSym*]

<i>Predicate</i> <i>symbol</i> : <i>PredSym</i> <i>terms</i> : seq <i>Term</i>
--

3.5 Environment

Agents can only function within the context of an environment, and before we begin to specify the components of the agent architecture, we must first consider the environment in which it is to be situated. There are several levels of abstraction at which the environment could be specified. For example, from an external viewpoint it can be seen as a collection of objects and agents, while to an agent situated in the environment it can be considered to be a collection of features that can potentially be perceived and acted upon. Since we are concerned with the agent level, we adopt this latter view of an environment as a set of perceivable features, or *attributes*, in the same way as Luck and d’Inverno [61]. Objects in the environment, including agents, are viewed as those clusters of attributes that characterise them. An agent is able to perceive objects, and other agents, by perceiving the appropriate clusters of attributes, which might include the size, colour, and location of objects.

Formally, an attribute is represented by a predicate, and an environment as a set of attributes. The schema *Env* is introduced to represent the particular environment in which an agent (or group of agents) is situated¹.

Attribute == *Predicate*
Environment == \mathbb{P} *Attribute*

¹Specifying the environment as a schema, rather than a global variable, makes it easier to specify the operations an agent can perform in it, using the Z notation’s Δ convention.

As an example, suppose that the environment contains a white cardboard box and a wooden table such that the box is on the table. Using our definition of an *Environment* we might represent this situation by the following set of attributes.

```
[ (colour, [box, white]), (madeFrom, [box, cardboard]),  
  (colour, [table, brown]), (madeFrom, [table, wood]),  
  (on, [box, table]) ]
```

This set of attributes represents the predicates that are true in the environment, and so are potentially perceivable as attributes by agents, such as the predicate that the cup is red or the predicate that it is made from porcelain. However, if a particular predicate is not contained in the set representing an environment then we may take this to mean either that the predicate must be false, or alternatively that we have no information about its truth or falsity. This is a well known problem, and can be thought of as there existing two kinds of negation [86], and can be addressed by making the assumption, as we do in this thesis, that since it is unrealistic to require a representation of the environment to include *all* true predicates, the absence of a predicate in the set representing the environment does not mean it is false.

3.6 Perceptions

In order to reason about its environment, and act appropriately in it, an agent must be able to *perceive* to determine its current state. In the SENARA architecture, an agent's ability to perceive its environment is determined by its set of perceiving capabilities, or *perception actions*. Perception actions operate on the agent's environment, leaving it unchanged, and return an appropriate set of percepts corresponding to the agent's perceptions. The details of the perception actions that comprise an agent's perception capabilities are related to its

domain. For example, a physically situated robot might have visual and auditory sensors, while a software agent may be able to determine the contents of certain data structures, such as the names of the files in a particular directory. We formally define a *View* to be a set of perceivable features, or attributes, and a perception action, *PerceptionAction*, as a function that takes an environment as its argument and returns a *View*, corresponding to those features that the agent perceives.

$$View == \mathbb{P} \textit{Attribute}$$

$$PerceptionAction == Environment \rightarrow View$$

3.7 Beliefs

An agent's *beliefs* are those propositions about its environment, itself, and others that it takes to be true, and can be thought of as representing the information it has about its environment. They are not necessarily verifiably true facts about the environment, rather they are the propositions that the agent *considers* to be true. Beliefs are typically a combination of *a priori* knowledge (such as information about capabilities, attributes, and the domain) and propositions obtained through perception of the environment and interactions with other agents.

Beliefs persist until an agent obtains new information from perception that contradicts them (as described later in Section 4.3). Since beliefs persist, they allow an agent to keep track of information over time, without which an agent would have available only the information from its immediate perceptions. It would have no access to information about the previous states of its environment, and would be unable to reason about previous events, making it difficult to achieve consistent behaviour over time.

Before we define beliefs we must define a literal as a predicate or its negation. A belief can then be defined as a single literal. Thus, we can define an agent's *beliefs* as a set of beliefs, which we interpret as the conjunction of all its elements, meaning that if the set contains the beliefs b_1 and b_2 the agent believes $b_1 \wedge b_2$.

$$\begin{aligned} \textit{Literal} ::= & \textit{pos}\langle\langle\textit{Predicate}\rangle\rangle \\ & | \textit{not}\langle\langle\textit{Predicate}\rangle\rangle \end{aligned}$$
$$\textit{Belief} ::= \textit{Literal}$$
$$\textit{Beliefs} ::= \mathbb{P} \textit{Belief}$$

As an example, consider an agent situated in the environment given in Section 3.5, such an agent represent its beliefs about the environment as follows, where `pos` signifies a predicate and `not` its negation.

```
[ (pos colour, [box, white]), (pos madeFrom, [box, cardboard]),  
  (pos colour, [table, brown]), (pos madeFrom, [table, wood]),  
  (pos on, [box, table]), (not empty, [table]) ]
```

3.8 Goals

An agent must have some means of guiding its behaviour and choosing between actions; it must have *direction*. *Goals* give an agent a purpose and allow it to select between the courses of action open to it. We take the common view of goals as being those situations that an agent wishes to bring about. Through perceiving its environment the agent is able to determine which of its goals are currently relevant, and then choose a course of action that contributes to the achievement of these goals. Consequently, an agent's goals influence how it will react to any given situation. Furthermore, where an agent is capable of performing problem-solving, its goals pose problems for it in that it must choose or construct plans about how best to achieve them. A goal is just a description of a situation, and can be formally specified as a set of literals.

$$\textit{Goal} ::= \mathbb{P} \textit{Literal}$$

For example, if in the situation given above an agent has the goal of the tabletop being cleared and the box being moved to the floor, it might represent this as follows.

```
[ (pos empty, [table]), (pos on, [box, floor]) ]
```

3.9 Actions

To be useful and to achieve its goals, an agent must be able to interact with its environment, i.e. it must be able to perform *actions*, which correspond to its *capabilities*. Now, since we are concerned with cooperation, we must also consider an agent acting as part of some more complex group action, which might be constructed out of multiple individual actions. For reasons of terminological clarity, therefore we introduce the term *contribution* to refer to the action of an individual, and the term *action* to refer to the wider set of both individual and group actions. In order to formally specify a contribution we first introduce a given set to represent the set of all possible action symbols.

[*ActSym*]

Since we are concerned with cooperation and group actions, we require that a contribution includes information about the agent who performs it. A *contribution* is therefore specified as an action symbol, a possibly empty sequence of terms (representing the parameters of the action), and a unique agent identifier that refers to the agent who performs it. We introduce a given set to represent the possible agent identifiers, each of which refers to precisely one agent.

[*AgentID*]

<p><i>Contribution</i> <i>symbol</i> : <i>ActSym</i> <i>terms</i> : seq <i>Term</i> <i>agentID</i> : <i>AgentID</i></p>

By way of example, consider the action of picking up a box; we can represent this as follows, where *agent1* is the agent that performs the action and *box1* is a box.

(pickup, [agent1, box1], agent1)

In order for an agent to reason about the contributions it can perform, it must have some mechanism for determining their effects. The function *contributionEffects*, takes a contribution and an environment, and returns that environment having had the action performed upon it.

$$| \textit{contributionEffects} : \textit{Contribution} \rightarrow \textit{Environment} \rightarrow \textit{Environment}$$

This could be implemented through the use of *add* and *delete* lists, in a similar manner to Fikes and Nilsson’s STRIPS system [31]. The add and delete lists contain those predicates that should be added and deleted respectively from the state representation on execution of the action.

An agent also needs to know when a contribution can be performed. For example it may only be possible to pick up a box if there is nothing on top of it. We therefore require that agents have an appropriate instantiation of the function *contributionPreconditions* below, which takes a contribution as its argument, and returns the preconditions that must be satisfied to perform it.

$$\textit{Precondition} == \textit{Literal}$$

$$| \textit{contributionPreconditions} : \textit{Contribution} \rightarrow \mathbb{P} \textit{Precondition}$$

3.10 Joint and Concurrent Actions

Cooperation may take the form of an agent performing an action on behalf of another, a group of agents performing an action together, or different agents performing a set of actions at the same time in pursuit of a common aim. Thus, along with the notion of an individual contribution described above, we can identify two additional action types: joint actions and concurrent actions.

A *joint action* is a composite action, made up of individual contributions that must be performed together by a group of agents. Each agent involved in executing a joint action

makes a *simultaneous* contribution to the joint action, corresponding to the component action that it performs. For example, if agents α_1 and α_2 perform the joint action of lifting a table together, then α_1 must make the contribution of lifting one end of the table simultaneously with α_2 lifting the other. A joint action is formally specified as a set of actions, to be performed simultaneously.

<i>JointAction</i>
<i>contributions</i> : \mathbb{P} <i>Contribution</i>
<i>#contributions</i> ≥ 2

As an example, suppose there is a box that is too large to be lifted by an individual agent, but can be lifted if two agents each lift one end of it. This can be represented as a joint action as follows, where *agent2* and *agent3* are variables corresponding to the agents that will perform the action.

```
[ (liftend, [agent2, box2], agent2),
  (liftend, [agent3, box2], agent3) ]
```

Concurrent actions are those that can be performed in parallel by different agents, without the need for synchronisation (except at the beginning and end of a set of concurrent actions). As with joint actions, agents perform *contributions* as part of a set of concurrent actions. For example, if agents α_1 and α_2 each write a chapter for a book, and they perform their actions in parallel, then α_1 and α_2 perform the concurrent contributions of writing their respective chapters. Concurrent actions can comprise both individual contributions and joint actions that are to be performed simultaneously.

CAcomponent ::= *Contrib*⟨⟨*Contribution*⟩⟩
 | *JA*⟨⟨ \mathbb{P} *Contribution*⟩⟩

<i>ConcurrentAction</i>
<i>contributions</i> : \mathbb{P} <i>CAcomponent</i>
<i>#contributions</i> ≥ 2

As an example, consider the concurrent action comprising the individual contribution of lifting a box, `box1`, and the joint action of two agents lifting a second box, `box2`, which can be represented as follows.

```
[ (pickup, [agent1, box1], agent1),  
  (liftend, [agent2, box2], agent2),  
  (liftend, [agent3, box2], agent3) ]
```

Our definitions of joint and concurrent actions are extensions to the BDI model described in Section 2.4, and are respectively related to the notions of “black-box” and weak parallelism described by Kinny *et al.* [55], where “black-box” parallelism refers to an action that must be executed by more than one agent, and weak parallelism refers to a set of actions that may be performed simultaneously without constraint on their ordering. The key difference, however, is that while we consider the component actions, or contributions, that make up a joint or concurrent action, Kinny represents joint actions as primitive, without consideration or representation of the individual contributions that comprise it. By representing the components of joint and concurrent actions, agents are given more scope for reasoning about how to establish cooperation for them, as we discuss later in Chapter 7. In particular, by representing the contributions in a joint action, we are able to develop mechanisms for requesting assistance for a contribution, rather than for a joint action as a whole (see Section 7.6).

3.11 Plans

In addition to the abilities of acting an agent must be able to perform some degree of problem-solving, or *planning*, which is concerned with determining a sequence of actions, or *plan*, to transform the environment into a desired state. Planning is an established subfield of artificial intelligence in its own right, and is not the focus of our work; rather than focusing on sophisticated techniques for planning, therefore, we adopt the simplified

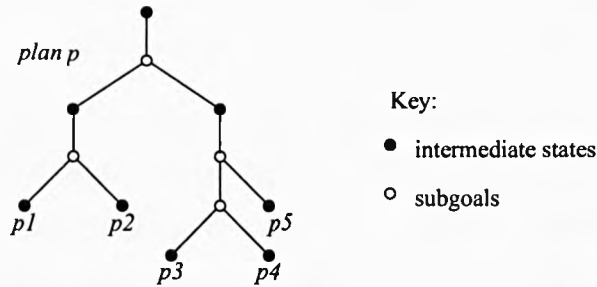


Figure 3.2: An example partial plan

approach taken in many existing BDI-based architectures, where agents are provided with a library of predefined plans from which they can select (e.g. [38, 83]). If a plan is defined solely in terms of a sequence of actions then it can be executed by an agent without the need for further reasoning, and is said to be *fully elaborated*. However, if all of an agent's plans are fully elaborated then its plan library would have to contain a plan to cover every eventuality, making it too large to be manageable. A common solution to this problem is to allow plans to be *partially elaborated*, meaning that they can be further refined to specific situations, and contain subgoals in addition to actions (e.g. [5, 38]). Before a partial plan can be executed, therefore, subplans must be found for the subgoals it contains; this process is called *elaboration* and is discussed later in Section 4.10. In addition to making the plan library more manageable, partial plans provide a simple means of interleaving planning and acting — an agent is able to execute the actions in a plan until it reaches a subgoal, at which point it must select a subplan before performing further actions.

Figure 3.2 shows a graphical representation of a partial plan that includes all possible elaborations, where the edges represent actions, solid bullets correspond to intermediate states between actions, and outline bullets correspond to subgoals. For each subgoal in the plan, there is a set of applicable plans, each of which forms a branch of possible elaboration from that subgoal. The set of plan elaborations is the set of paths from the root of the graph to the leaves. Thus, for plan p , possible elaborations are paths from the root to the nodes

labelled $p1$, $p2$, $p3$, $p4$, and $p5$. Note that this graphical representation is limited and does not allow certain types of plan to be illustrated; it is, however, sufficient for our purposes.

Now, since there may be several plans in a library that achieve a given goal, some way of choosing between them is needed. Different plans may be applicable in different situations, and we introduce the notion of a plan's *preconditions* to represent the situations in which it is applicable, to enable an agent to select an appropriate plan for a given situation. This is different to the approach taken in dMARS and AgentSpeak(L), where plans are associated with a trigger event, which corresponds to a particular change in the environment (or the agent's goals). Such trigger events cause the plan to be adopted and an appropriate intention formed. In our approach to formalising plans we are concerned with specifying the conditions under which a plan *can* be selected, rather than when it *will* be selected since this will be determined by the goals generated by the agent's motivations. The preconditions of a plan define what must be believed by the agent for the plan to be applicable, and are represented by a set of beliefs.

For agents to be able to interact effectively with others, their plans must be able to represent a group of agents performing actions together; they must have the facility to include joint and concurrent actions. Thus, a step in a plan is defined to be either an individual contribution, a joint action, a concurrent action, or a subgoal, and we arrive at the following specification for a plan, defined as a sequence of steps to achieve a particular goal in a given context.

$$\begin{aligned}
 \textit{PlanStep} ::= & \textit{Individual}\langle\langle \textit{Contribution} \rangle\rangle \\
 & | \textit{Joint}\langle\langle \mathbb{P} \textit{Contribution} \rangle\rangle \\
 & | \textit{Concurrent}\langle\langle \mathbb{P} \textit{CAcomponent} \rangle\rangle \\
 & | \textit{Subgoal}\langle\langle \textit{Goal} \rangle\rangle
 \end{aligned}$$

Plan

$ \begin{aligned} \textit{achieves} & : \textit{Goal} \\ \textit{preconditions} & : \mathbb{P} \textit{Belief} \\ \textit{body} & : \textit{seq} \textit{PlanStep} \end{aligned} $
--

To illustrate the form a plan takes, consider an agent with the goal of moving a box b

onto another a , where initially both boxes are on the floor and neither have anything on top of them. This can be represented as $on(a, floor) \wedge on(b, floor) \wedge clear(a) \wedge clear(b)$, and the desired situation can be represented as $on(a, floor) \wedge on(b, a)$. Since there is a simple action that transforms the initial state into the desired state, no subgoals are required, and the plan can be represented as a list of action steps, as follows. As described in Section 3.7 a literal is a predicate or its negation, which we represent as `pos` and `not` respectively.

```

Plan:
achieves:      [(pos on, [a, floor]), (pos on, [b, a]),
               (pos clear, [b])],
preconditions: [(pos on, [a, floor]), (pos on, [b, floor]),
               (pos clear, [a]), (pos Clear, [b])],
body:         [action <(move, [b, floor, a], agentID)>]

```

3.12 Intentions

Intentions represent the plans that an agent is currently *committed* to executing in order to achieve particular goals. As we discussed in Section 2.3, there are two justifications for introducing the mental component of intention. Firstly, since an agent has finite resources, it cannot continually consider its competing goals in deciding its actions, and must eventually settle on a particular goal and establish some form of *commitment* to that objective. Secondly, an agent must coordinate its present and future actions; once a future action has been decided on, and the intention to do it has been formed, an agent should be able to determine further actions on the assumption that the intended action will be performed.

For an agent's intentions to be useful they must be *internally consistent* and not conflict with other intentions, or with the agent's beliefs. They should also have a degree of *stability* in that they resist being reconsidered or abandoned. We take consistency and stability to be fundamental properties of intentions.

An intention is a plan to achieve a goal, together with commitment to its achievement. Thus, to represent intentions we must first find a way to represent commitment, which can

be thought of as defining the period over which an agent must pursue a particular intention, and the time at which it is appropriate to drop that intention. As described in Section 2.3 Cohen and Levesque offer an approach to representing commitment by requiring intentions to include a *relevance condition* that describes the circumstances under which the agent must keep its intention, in addition to the intuitive conditions that the intended goal has been achieved or can never be achieved.

In addition to these conditions, *motivations* (which we describe in the following section) are a key factor in determining when to drop an intention, since they determine how valuable a given goal (and corresponding intention) is to an agent at a particular time. In SENARA an agent's *motivations* provide (at least in part) the *reason* for its intentions, and so an agent's motivations play a role in determining when to drop an intention. If an intention ceases to be motivationally valuable to an agent then it should discharge its commitment to it. Therefore, we extend Cohen and Levesque's notion of commitment by requiring that an intended goal be of motivational value to the agent holding that commitment.

When executing an intention, if the first step of the plan at the top of stack is an action, the agent can execute it as soon as the action's preconditions are met, and once an action is performed it can be removed from the head of the plan. However, if the first step of the plan is a goal, the agent must choose a subplan to achieve it. Now, since plans may be partial, subplans may be required for intention execution, and we need to extend our notion of intention to be a *stack* of plans, along with some associated commitment. As in dMARS and AgentSpeak(L), each plan on the stack is a subplan of the one below it. The chosen subplan is pushed onto the stack of plans, since the agent must execute it and achieve the subgoal before continuing with the remainder of the plan. Both of these plans, however, are part of the same intention; the same course of action to achieve the same goal. Once all the steps of a subplan are executed it is removed from the stack. An *intention*, therefore, is defined as a stack of plans, a relevance condition, and the goal it satisfies. We can represent a stack as a sequence of plans, with the head of the sequence at the base of the stack. The following schema, *Intention*, uses two auxiliary functions *isSubPlanOf* and

extractPlan defined in Appendix A, which check whether a plan is a subplan of another, and extract a particular plan from a sequence of plans respectively.

<p><i>Intention</i></p> <p><i>plans</i> : seq <i>Plan</i> <i>relevance</i> : \mathbb{P} <i>Belief</i> <i>satisfies</i> : <i>Goal</i></p> <p>$\forall i : \mathbb{N}_1 \mid i \leq \#plans - 1 \bullet$ <i>isSubPlanOf</i> (<i>extractPlan</i> (<i>i</i> + 1, <i>plans</i>)) (<i>extractPlan</i> (<i>i</i>, <i>plans</i>)) = true</p>
--

This schema describes how all plans in the sequence are subplans of the plan at the head of the sequence or, more strictly, that the plan at position *i* + 1 in the sequence is a direct subplan of the plan at position *i*. Consider the example of an agent having the intention of moving a box from one room to another in a warehouse, with the body of the plan from which this intention was formed containing the three steps of: lifting the box, achieving the goal of being in the desired location, and putting the box down. If *b* represents the box, and *loc*₁, *loc*₂ the initial and target locations of the box respectively, this plan body might be written as follows.

```
body:          [action <(lift, [b], agentID)>,
                subgoal <[ (pos location, [agentID, loc1] ) ]>,
                action <(putdown, [b], agentID)>]
```

The corresponding intention is represented graphically as (a) in Figure 3.3, which shows the components of the intention: the goal it achieves, its preconditions, and the stack of plans which achieve the goal. After performing the first action, the plan body is left containing the second two steps, i.e. (b) in Figure 3.3. Since the first step is now a subgoal, a subplan must be chosen for this and pushed onto the intention stack. If the chosen subplan contains the single action of moving right, the resultant intention is as shown in (c).

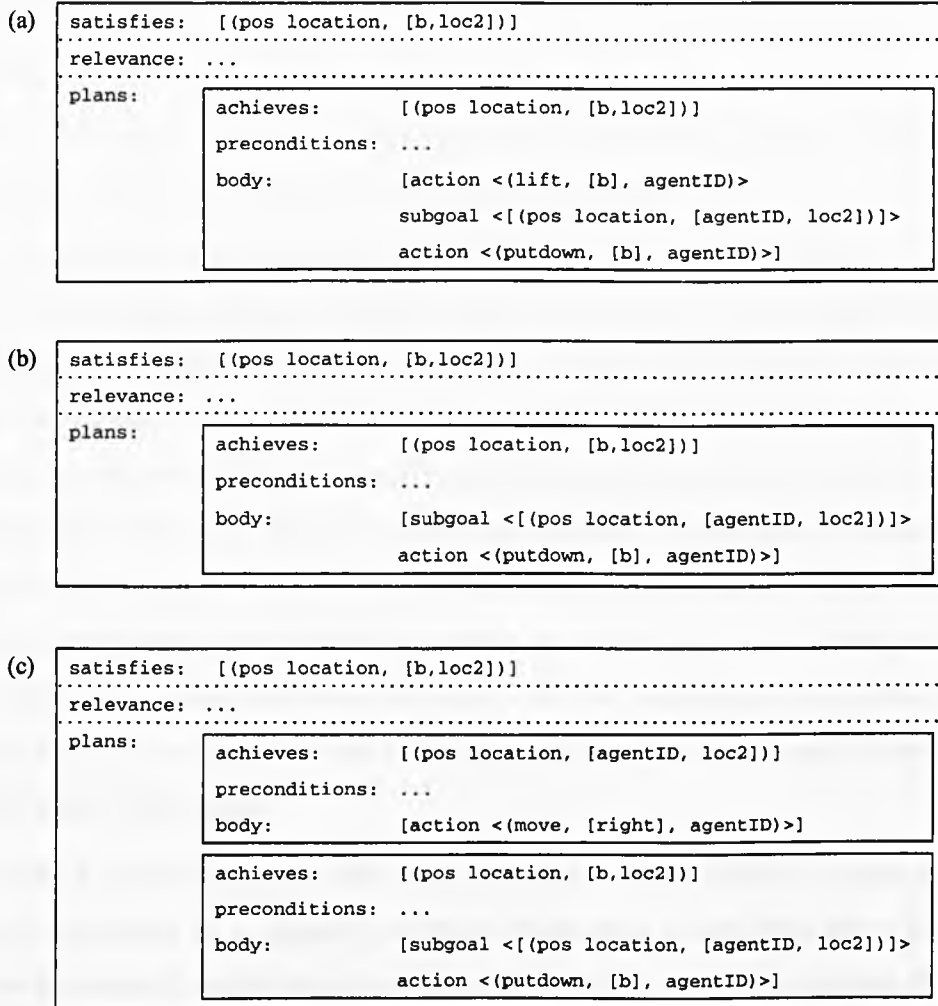


Figure 3.3: The use of a *stack* of plans in an intention

3.13 Motivations

Thus far in this section we have introduced the standard components of a BDI agent: beliefs, desires (or goals in our terminology), and intentions, as well as the common notion of a plan library. However, in order to include the notion of motivation, which as described in Chapter 2 can be used to achieve agent autonomy, we depart from this standard BDI approach. Motivations provide a mechanism for goal generation and adoption, thus allowing an agent to have control over its own behaviour, i.e. be *autonomous*.

We adopt Luck and d’Inverno’s view that a motivation has an intensity, a threshold value, a set of goals it can generate, and a mitigation function (as reviewed in Section 2.2). The intensity of a motivation changes according to the state of the agent and its environment and, if it exceeds the associated threshold then a response is triggered. This response is to choose the most applicable goals from the associated set of goals and add them to the agent’s existing goals; we say that these goals have been *generated*. For example, the intensity of a *hunger* motivation may rise above the associated threshold if an agent’s energy level drops below a certain value, and cause the generation of a goal to eat a snack. Note that while the intensity of a given motivation fluctuates with time, motivations themselves are *not* transient, and the set of motivations a particular agent has does not change, but rather it is their intensity that changes.

When a generated goal is satisfied, the intensity of the motivation is reduced by the amount determined by a *mitigation function*, which takes a motivation and a goal, and returns the amount by which the achievement of the goal mitigates the motivation. Different agents may have different mitigation functions and place different motivational values on a given goal. The mitigation function may place the same value on achieving a particular goal regardless of the intensity of the motivation, or the motivational effect may be determined by the current intensity. In our model we view the motivational effect of achieving a goal as dependent on the motivation’s intensity.

[*MotiveSym*]

Motivation

name : *MotiveSym*

intensity : \mathbb{R}

threshold : \mathbb{R}

goals : \mathbb{P} *Goal*

Motivations also aid an agent in choosing which of its current intentions to pursue, when there are more than one. By choosing the intention whose completion will mitigate the highest motivation by the largest amount, the agent ensures that it always acts appropriately. Thus, motivations provide a mechanism by which attention and conduct can be directed.

An agent determines the intensity of its motivations according to the believed state of the environment. Associated with each motivation is a function that defines the intensity of the motivation in terms of the environment, and we describe this function in Section 4.4.

When adopting a goal, an agent may be faced with a number of applicable plans, and its motivations can help in choosing between them. As with choosing between intentions, an agent should choose, from its set of applicable plans for a given goal, the one that offers the highest motivational value.

In addition to a mitigation function determining the value of a *goal*, each of an agent's motivations also has a complementary mitigation function that determines the motivational value of an *action* to the motivation. The value returned by a mitigation function is dependent on the current intensity of the motivation with which it is associated. Thus, if a motivation is of low intensity (relative to other motivations), and of little relevance to the agent, then its associated mitigation function will return low values, since even actions that have a *large* motivational effect on that motivation are of relatively *low* importance to the agent overall. For example, suppose an agent has a *hunger* motivation and the action of eating mitigates it. If the *hunger* motivation is high, then the action of eating is of high motivational importance and the mitigation function should reflect this. Similarly, if the *hunger* motivation is low, then eating is less important and will be of less motivational value to the motivation.

To determine the overall motivational value that would arise from performing an action, the agent must consider the value of the action to each of its motivations, since the action may be of value to more than one motivation. These values can then be combined by summing them into a single value representing the motivational value of the action concerned.

The motivational value of a plan is influenced firstly by the value that would arise from the achievement of the goal that the plan is intended to achieve and, secondly, by the motivational value that is associated with the actions and subgoals contained in the plan. However, determining the motivational value of a plan is complicated by the partial nature of plans, and we postpone our discussion of the issues involved until Chapter 6, in which we describe the process of plan selection.

As stated already, an agent's actions are governed by the intensities of its motivations, since they determine the goal that is of the highest importance to the agent. Recall from earlier in this section the notion of an intention as a commitment to a particular goal, along with a stack of (partial) plans to achieve it. Since action arises from the execution of the plan component of an intention, and agents should act towards the goal that is of the highest motivational value, some mechanism is required to determine which intention is of the highest motivational value. In other words, an agent must be able to determine which intention is of the most importance, and consequently, which plan to execute. In ascertaining the motivational value of a particular intention, an agent should consider both the value arising from the goal the intention is towards, and the plans and action contributions that it contains.

Thus, an agent requires an instantiation of the following functions, where *assessSituation*, *generateGoals*, and *mitigation* correspond to the mechanisms for assessing the current believed situation with respect to the motivation, generating an appropriate response in terms of a set of goals, and determining the motivational value of a goal respectively. The latter three functions in the schema below take a contribution, plan, and intention as arguments and return the motivational value associated with the contribution, plan or intention

respectively.

$$\begin{array}{l} \textit{assessSituation} : \textit{Motivation} \rightarrow \mathbb{P} \textit{Belief} \rightarrow \mathbb{R} \\ \textit{mitigation} : \textit{Motivation} \rightarrow \textit{Goal} \rightarrow \mathbb{R} \\ \textit{mvContribution} : \textit{Motivation} \rightarrow \textit{Contribution} \rightarrow \mathbb{R} \\ \textit{mvPlan} : \textit{Motivation} \rightarrow \textit{Plan} \rightarrow \mathbb{R} \\ \textit{mvIntention} : \textit{Motivation} \rightarrow \textit{Intention} \rightarrow \mathbb{R} \end{array}$$

3.14 Summary

We have now introduced the set of mental components that comprise the agent architecture. These mental components are a standard part of a BDI-based architecture, with the exception of motivations which are an extension of the BDI approach. An agent has a certain set of capabilities, or action contributions, that it can perform, along with a certain set of perception capabilities. In addition to these the agent has sets of beliefs, goals, intentions, and motivations, along with a particular identifier that is unique to each agent. The agent also has a library of plans which it is able to use in determining how to achieve its goals. When an agent cooperates with others it forms a commitment to inform them if it should later cease its cooperative action, as we describe in Chapter 7; we call this a *nominal commitment*, and a set of such commitments is the final mental component of a SENARA agent.

[*NominalCommitment*]

Agent

agentID : *AgentID*

capabilities : \mathbb{P} *Contribution*

perceivingCapabilities : \mathbb{P} *PerceptionAction*

beliefs : \mathbb{P} *Belief*

goals : \mathbb{P} *Goal*

intentions : \mathbb{P} *Intention*

motivations : \mathbb{P} *Motivation*

planLibrary : \mathbb{P} *Plan*

nominalCommitments : \mathbb{P} *NominalCommitment*

motivations $\neq \emptyset$

$\forall c : \text{Contribution} \bullet c \in \text{capabilities} \Leftrightarrow c.\text{agentID} = \text{agentID}$

Chapter 4

A Motivated BDI Agent Architecture

4.1 Introduction

The interplay between an agent's mental components, are defined by a set of control mechanisms, which in turn determine the behaviour an agent exhibits. In order to act autonomously, an agent must react to the changes in its environment, and adopt goals in response to them; the manner in which a motivated agent responds to such changes is determined by its motivations. The significance of a given motivation is dependent on the current state of the environment and, at any one time, the most significant of an agent's motivations determine the goals that it should pursue. In this chapter we present the control mechanisms, in the form of a *reasoning cycle*, that act on the mental components of a SENARA agent, completing the architecture. The reasoning cycle can be broken down into the nine stages shown in Table 4.1, each of which we describe in detail in the remainder of this chapter.

The SENARA architecture is illustrated in Figure 4.1, where the agent's mental components are represented by rounded boxes, which in turn are within the central dashed box. Control processes that operate on these components are represented by rectangular boxes, and arrows correspond to the flow of control.

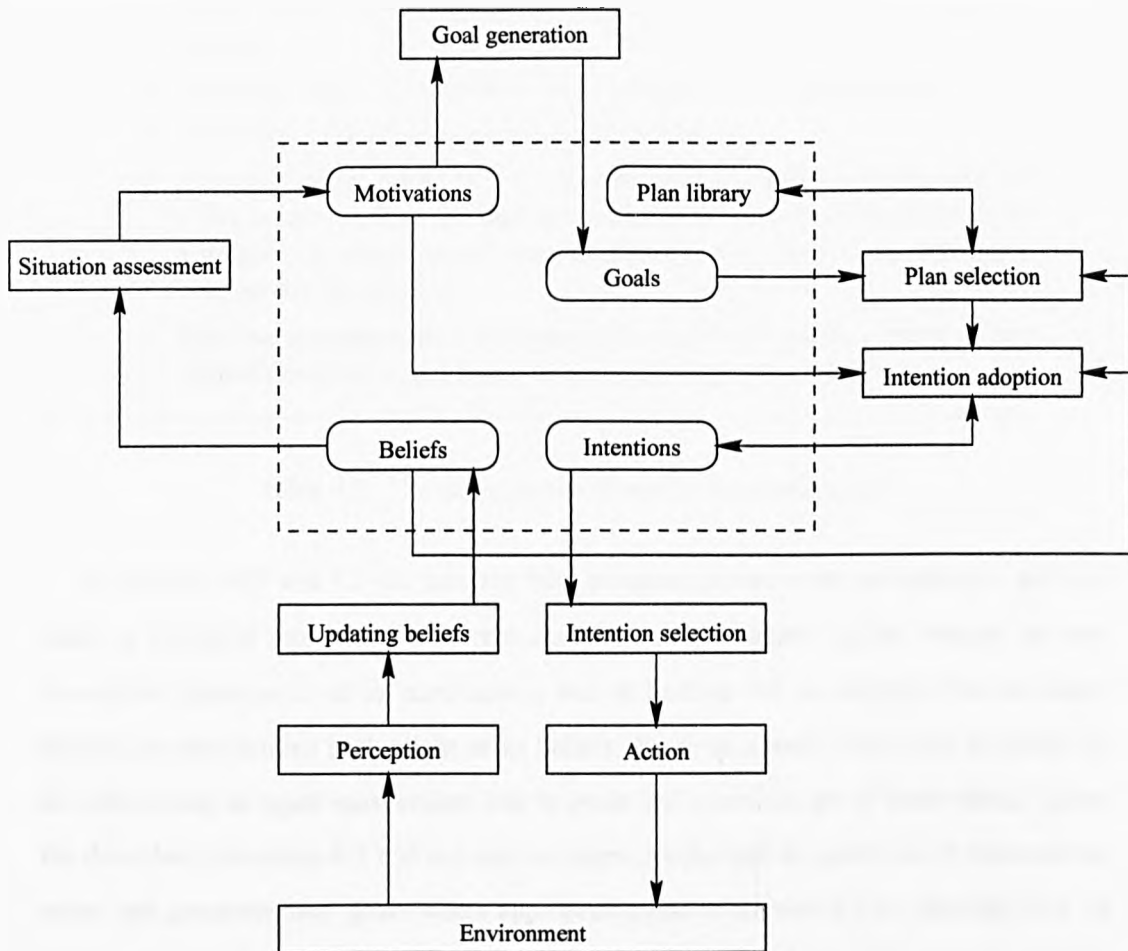


Figure 4.1: The SENARA architecture

-
1. Perceive the environment.
 2. Update beliefs in accordance with current perceptions.
 3. Update motivations in the light of these new beliefs.
 4. Generate a set of new goals from the updated motivations.
 5. Select an appropriate plan for the most motivated of these newly generated goals.
 6. Adopt that plan as an intention to achieve the corresponding goal.
 7. Select one of the current intentions to pursue.
 8. If the first step in the body of the chosen intention is an action then execute it and remove it from the intention body, otherwise, the first step must be a subgoal, in which case attempt to elaborate the intention by selecting a subplan for the subgoal.
 9. Perceive the appropriate changes in the environment, as a result of any actions performed, and return to the beginning of the cycle.
-

Table 4.1: The stages in the SENARA reasoning cycle

In Sections 4.2 and 4.3 we describe how an agent perceives its environment, and updates its beliefs to represent the current situation. As an agent's beliefs change, so does the relative importance of its motivations, and in Section 4.4 we describe how an agent assesses its motivations in the light of its beliefs. Since an agent's behaviour is guided by its motivations, an agent must ensure that its goals and intentions are of motivational value. We describe in Sections 4.5 and 4.6 how an agent checks that its goals are of motivational value, and generates new goals where appropriate, and in Section 4.7 we describe how an agent ensures that its intentions are of value. Once an agent has dropped inappropriate goals and intentions, it must adopt new intentions for the goals it has generated, and we describe this process in Section 4.8. In order to act, an agent must select an intention to focus upon based on its motivations and then act towards it. Finally, Sections 4.9 and 4.10 discuss how to select an intention and how to act towards it respectively.

Inputs:

env — the agent’s environment
ps — the agent’s perception capabilities

Outputs:

view — the agent’s current perceptions

Algorithm:

```
view = empty
for p in ps do
    view = union(view, (p(env)))
return view
```

Table 4.2: Algorithm for agent perception

4.2 Perceiving the Environment

In what follows we give Z specification and algorithms for the various mechanisms that comprise the SENARA reasoning cycle. The first step in reasoning is for an agent to perceive its environment by executing its perception actions, as outlined in Table 4.2. When the outputs of these perception functions are taken together, they result in a set of *percepts* of the current environment represented by a set of attributes. An agent must decide whether or not to incorporate these percepts into its beliefs, since they may conflict with existing beliefs, and we call these percepts its *candidate beliefs*. Note that the perceptions of a socially situated agent may also include information about others and their actions.

We formalise the perception process in the following schema, in which we include the schemas representing an agent and its environment (we use the Ξ convention to signify that there is no change of state to these). We then specify an agent’s view to be the generalised union of the sets of percepts resulting from applying the agent’s perception capabilities in the current environment.

AgentPerception

$\exists Env$

$\exists Agent$

view : *View*

$view = \bigcup \{v : View \mid (\exists pAct : PerceptionAction \mid pAct \in perceivingCapabilities \bullet v = pAct\ environment)\}$

4.3 Updating Beliefs

Beliefs persist until an agent obtains new information contradicting them. Typically, such information will be the result of the agent perceiving its environment, although it may also arise from communication with other agents or a human user. An agent's beliefs are required to be consistent with each other, that is, if an agent believes p it should not also believe $\neg p$. If an agent were to have inconsistent beliefs it might exhibit irrational behaviour, since this might lead to incompatible goals. For example, if I were to simultaneously believe that "it is raining" and "it is not raining" then I might come to have the incompatible goals (and attempt to adopt incompatible intentions) of "staying indoors" and "going outside". An agent cannot, therefore, simply add its candidate beliefs to its existing beliefs because inconsistencies may result. Instead, agents must introduce new beliefs into their belief set using some consistency maintaining mechanism.

Although belief revision (e.g. [27, 35]), which is concerned with determining how an agent should revise its beliefs in the light of contradictory information, is not directly relevant to this thesis, an agent must have some mechanism for updating its beliefs. If an agent's candidate beliefs include information that contradicts its existing beliefs then it must determine which information to drop. It can either keep its existing beliefs and discard the new information, or it can drop the existing beliefs and include the new information. Several strategies for belief revision have been suggested, such as Doyle's Truth Maintenance System [26], and Galliers' model of autonomous belief revision [36], but in this work we simply assume that agents have a mechanism for ensuring their beliefs are kept consistent,

Inputs:

bel — the agent's beliefs
view — the current perceptions

Outputs:

bel' — a revised set of beliefs

Algorithm:

candidateBeliefs = *view*
bel' = *reviseBeliefs*(*bel*, *candidateBeliefs*)
return *bel'*

Table 4.3: Algorithm for updating beliefs

which we call a *consistency maintainer*, without elaborating further. Using this consistency maintainer, an agent is able to incorporate its set of candidate beliefs into its existing beliefs using the algorithm given in Table 4.3, and formally specified below, by which the conflicting information is dropped.

In this table, and the following specification, the function *reviseBeliefs* takes an agent's beliefs and a set of candidate beliefs, and returns the agent's updated beliefs, and corresponds to the consistency maintenance mechanism. The specification includes the schemas *Agent* and *AgentPerception*, the former using Z's Δ convention meaning its state (in particular the agent's beliefs) can change, while the state of the latter is unchanged. The function *interpretView* forms a set of candidate beliefs from the set of attributes that comprise a view.

UpdateBeliefs Δ *Agent* \exists *AgentPerception**interpretView* : *View* \rightarrow \mathbb{P} *Belief**reviseBeliefs* : \mathbb{P} *Belief* \rightarrow \mathbb{P} *Belief* \rightarrow \mathbb{P} *Belief**candidateBeliefs* : \mathbb{P} *Belief*

candidateBeliefs = *interpretView view**beliefs'* = *reviseBeliefs beliefs candidateBeliefs**goals'* = *goals**intentions'* = *intentions**motivations'* = *motivations*

4.4 Updating Motivations

Once an agent has perceived its environment and modified its beliefs accordingly, it must update the intensity levels of its motivations in the light of its changed beliefs. The intensity of a motivation is determined (at least in part) by the agent's beliefs about its current environment. For example, if I am crossing a road and believe that a car is heading towards me, the intensity of my motivation for survival is likely to increase, and similarly, once the car has passed by, it is likely to decrease.

Agents are guided by their motivations, and therefore, do not hold goals that are not motivated. Thus, after updating the intensities of its motivations an agent checks that its goals are still appropriate. In other words, if the intensity of the agent's motivations change in such a way as to make a particular goal *unmotivated*, then the agent drops it. For example, if I see a car heading towards me while crossing a road, my survival motivation may cause me to have the goal of running the across the rest of the road. However, if I then notice that the car has stopped and parked, the intensity of my survival motivation will decrease, and it may no longer be appropriate to keep the goal of running the rest of the way. In the remains of this section we discuss in more detail how an agent should alter the intensity of its motivations, and when it should drop goals due to lack of motivation.

An agent must have some means of assessing the current (believed) situation and adjusting the intensity of its motivations accordingly. We require agents in the SENARA architecture to have an instantiation of the *situation assessment* mechanism, introduced in Section 3.13.

Direct-association

The simplest approach, which we call *direct-association*, is for an agent to have for each of its motivations an association between a particular set of beliefs, and a predetermined proportional change of intensity (positive or negative). Whenever the set of beliefs associated

Inputs:

bel — the agent’s beliefs

motivations — the agent’s motivations

Outputs:

the intensities of the agent’s motivations are updated

Algorithm:

for *m* **in** *motivations* **do**

m.intensity = *m.intensity* × *assessSituation(m, bel)*

Table 4.4: Algorithm for updating the intensity of motivations

with a motivation are believed to be true by the agent, it should change the motivation’s intensity by the specified level, as illustrated by the algorithm in Table 4.4.

In SENARA, the direct-association approach is represented as a set of tuples of the form (b, m, i) , where b is a set of beliefs, m a motivation, and i the proportional change that motivation’s intensity should take, should the beliefs b be held by the agent. We assume that there is at most one tuple for given sets of beliefs and motivations, otherwise it would be ambiguous and unclear which intensity change the agent should implement. As an example, suppose that if the agent’s energy level drops below some threshold, its *tiredness* motivation should be triggered and change intensity by n . Then we would represent this using a tuple of the form $(lessThan(energy, threshold), tiredness, n)$. To specify this we use the supplementary functions, *First*, *Second* and *Third* that return the first, second and third components of a Cartesian product respectively, as defined in Appendix A. We define a direct association tuple to contain a set of beliefs, a motivation, and a real number. The schema *IntensityAssociation* contains a set of direct association tuples, and specifies that an agent should not have ambiguous tuples for a given motivation and set of beliefs, i.e. for each motivation and set of beliefs any direct association with matching beliefs and motivation yields the same change of intensity.

Finally, we specify (in the schema *UpdateMotivations*) that an agent should update the intensity of its motivations by the proportion defined by the direct association tuple that

corresponds to the agents current beliefs and the motivation concerned. We specify this through the use of an auxiliary function *getIntensity*, which takes a set of association tuples, a set of beliefs, and a motivation and returns the degree of intensity change obtained from the matching tuple. The predicate part of the schema first specifies this auxiliary function, and then specifies that it should be applied to each of the agent's motivations (using its current beliefs).

$$iAssociation == \mathbb{P} Belief \times Motivation \times \mathbb{R}$$

IntensityAssociation

iAssociations : $\mathbb{P} iAssociation$

$$\begin{aligned} \forall B : \mathbb{P} Belief; m : Motivation; i : \mathbb{R}; a : iAssociation \bullet \\ a = (B, m, i) \wedge a \in iAssociations \\ \Rightarrow (\forall j : \mathbb{R}; a' : iAssociation \mid a' = (B, m, j) \wedge \\ a' \in iAssociations \bullet i = j) \end{aligned}$$

UpdateMotivations

$\Delta Agent$

$\exists IntensityAssociation$

getIntensity : $\mathbb{P} iAssociation \rightarrow \mathbb{P} Belief \rightarrow Motivation \rightarrow \mathbb{R}$

$$\begin{aligned} \forall IA : \mathbb{P} iAssociation; B : \mathbb{P} Belief; m : Motivation; i : \mathbb{R} \bullet \\ getIntensity IA B m = i \\ \Leftrightarrow (\exists iA : iAssociation \bullet iA \in IA \wedge B = First(iA) \wedge \\ m = Second(iA) \wedge i = Third(iA)) \\ \forall m : Motivation \mid m \in motivations \bullet (\exists_1 m' : Motivation \mid \\ m' \in motivations' \bullet m'.name = m.name \wedge \\ m'.intensity = m.intensity* \\ (getIntensity iAssociations beliefs m) \wedge \\ m'.threshold = m.threshold \wedge m'.goals = m.goals) \\ beliefs' = beliefs \\ goals' = goals \\ intentions' = intentions \end{aligned}$$

It is worth noting that there are several other strategies for changing the levels of motivations that may be explored in further work. For example, agents might learn to associate new sets of beliefs with appropriate motivation intensities, and modify the levels associated with the sets of beliefs in the light of experience.

4.5 Ensuring Goals are Motivated

A motivated agent must always act according to its motivations, and therefore each of its goals and intentions must be of motivational value. Before considering how to ensure that goals are motivated, we first describe what we mean by a goal being *motivated* or *unmotivated*. As an initial attempt we might propose that a goal is motivated while the intensity of the motivation that caused its generation is greater than its associated threshold, and if this intensity falls below the threshold the goal becomes unmotivated. However, this would be too simplistic since a single goal may be of *motivational value* to more than one motivation, and the achievement of the goal may mitigate more than one motivation. For example, if I have the goal of eating a meal its achievement may mitigate both my *hunger* and my *greed* motivations, even if it was my *hunger* motivation that caused its generation in the first instance. A consequence of this is that if my *hunger* motivation drops below its associated threshold it is *not* true to say that my goal is unmotivated, since it is still of motivational value to my *greed* motivation.

This issue can be addressed by saying that a goal is motivated if it mitigates one of the agents motivations, i.e. if the achievement of the goal would reduce the intensity of the motivation. Thus in the above example, my goal of eating a meal would be considered to have motivational value because its achievement would mitigate my *greed* motivation. However, suppose that my *greed* motivation's intensity was zero, and therefore I am currently placing no importance on it. In this case the goal cannot be said to be motivationally beneficial, since a motivation's intensity cannot be reduced below zero — a motivation with zero intensity cannot be mitigated. Therefore, we arrive at a comprise approach, and say that a goal is motivated if and only if its achievement would mitigate a motivation whose intensity is greater than zero.

As we describe in the following section, an agent's goals are generated as a result of a motivation's intensity rising above a certain threshold. Thus, at the time of generation it is clear that the goal is *motivated*, i.e. is the result of a particular motivation having a positive

Inputs:

goals — the agent's goals

motivations — the agent's motivations

Outputs:

goals' — an updated set of goals, such that unmotivated goals are removed

Algorithm:

goals' = *goals*

for *g* **in** *goals* **do**

keepgoal = *false*

for *m* **in** *motivations* **do**

if $\text{mitigationalValue}(g, m) > 0$ **and** $\text{intensity}(m) > 0$ **then**

keepgoal = *true*

if not *keepgoal* **then**

 drop(*g*, *goals'*)

return *goals'*

Table 4.5: Algorithm for dropping unmotivated goals

intensity. Since we require an agent to act in a motivated fashion, it should drop any goals that are no longer motivated. After updating the intensities of its motivation an agent should check which of its goals are appropriate and it should drop any that are not motivated¹.

The algorithm for dropping unmotivated goals is given in Table 4.5, which shows how an agent should check each of its goals against its motivations, and discard those that are not of motivational value to a single motivation with intensity greater than zero². Formally, we can write this as follows in the schema *DropUnmotivatedGoals*, which describes the change in state of an agent's goals after dropping unmotivated goals. A goal remains after dropping those that are unmotivated, if and only if there exists a motivation with positive intensity that is mitigated by that goal. Conversely, a goal is discarded if for all motivations (of positive intensity) it is of no motivational value.

¹We assume here that any unmotivated goals have not been adopted as intentions, since any unmotivated adopted goals, i.e. intentions, will also be dropped because they are not of motivational value, as described in Section 4.7.

²We refer to motivational value as mitigation in our formal specification and algorithms.

DropUnmotivatedGoals

Δ_{Agent}

$$\begin{aligned} \forall g : Goal \mid g \in goals \bullet g \in goals' &\Leftrightarrow (\exists m : Motivation \mid \\ & m \in motivations \bullet (mitigation\ m\ g) > 0 \wedge m.intensity > 0) \\ \forall g : Goal \mid g \in goals \bullet g \notin goals' &\Leftrightarrow (\forall m : Motivation \mid \\ & m \in motivations \bullet mitigation\ m\ g = 0 \vee m.intensity = 0) \\ beliefs' &= beliefs \\ intentions' &= intentions \\ motivations' &= motivations \end{aligned}$$

4.6 Goal Generation

SENARA is an architecture for *autonomous* agents and we consider the ability of an agent to generate its own goals to be a fundamental consequence of its autonomy; the process of goal generation, however, is typically omitted in existing BDI-based architectures. An agent's motivations provide a mechanism for the generation of goals, and an agent should perform *goal generation* as soon as it has updated the intensities of its motivations. In order to generate its goals, an agent must determine which of its motivations are *active*, i.e. which have intensities greater than their associated thresholds. Thus, the first step in goal generation is for an agent to check which of its motivations are active. Each active motivation will cause the generation of one or more goals, and the agent must have some mechanism for determining what these will be. Recall from our definition of a motivation, in Section 3.13, that each motivation has an associated set of goals it can generate, from which the most applicable is selected in a given situation. Formally, we introduce a function, *generateGoals*, that determines the goal a particular motivation generates in a given situation. The inputs to this function are the motivation concerned, the agent's beliefs and its other motivations, and the output is the set of goals that are generated.

$$\mid generateGoals : Motivation \rightarrow \mathbb{P} Belief \rightarrow \mathbb{P} Motivation \rightarrow \mathbb{P} Goal$$

For example, if my motivation of *hunger* is active and I believe that it is lunch time, then I might generate the goal of eating a meal. However, if I believe that it is evening I

Inputs:

gAssociations — the agent's goal generation associations

motivations — the agent's motivations

goals — the agent's goals

beliefs — the agent's beliefs

Outputs:

goals' — an updated set of goals, including newly generated goals

Algorithm:

activeM = empty

goals' = *goals*

for *m* **in** *motivations* **do**

if $\text{intensity}(m) \geq \text{threshold}(m)$ **then**

activeM = union(*activeM*, {*m*})

for *m* **in** *activeM* **do**

for *ga* **in** *gAssociations* **do**

if first(*ga*) = *beliefs* **and** second(*ga*) = *m* **then**

goals' = union(*goals'*, third(*ga*))

return *goals'*

Table 4.6: Algorithm for goal generation

may generate the goal of eating a light snack. In addition to beliefs, other motivations such as that of being *healthy*, may affect the goal that I generate, such as generating the goal of eating a low fat meal rather than a high fat meal.

The choice of which particular goal to generate should consider how *motivationally valuable* each goal is to a particular motivation, namely, how much the achievement of the goal would mitigate the motivation. Since we do not require that an agent's goals are consistent, because it is not committed to achieving *all* its goals, any goals that the agent generates can simply be added to its existing goals.

As with the mechanism an agent uses to update its motivations, different agents may utilise different strategies. However, one possible implementation of this goal selection mechanism is for the agent to associate a set of beliefs with a particular set of goals and a motivation. This can be represented as a tuple, as per the direct association mechanism

described in Section 4.4. Such a tuple would be of the form (b, m, g) , where b is a set of beliefs, m a motivation, and g the set of goals that are generated by an agent with the motivation and the set of beliefs. Although this method is very simple, and does not consider the agent's other motivations, it is adequate for our purposes of investigating cooperation — provided the agent's designer has chosen appropriate associations. More complex agents may use heuristics that consider such issues as the intensity of other motivations, and any past experience of the agent.

The algorithm for goal generation is given in Table 4.6, which shows that for each of its active motivations the agent should generate the goals defined by the corresponding goal generation association tuple. Formally, we define a goal association tuple to contain a set of beliefs, a motivation and a set of goals. The schema *GoalGenerationAssociation* specifies that there should be no ambiguity in an agent's set of goal associations — for a given motivation and set of beliefs there should be exactly one set of generated goals, according to the set of goal associations. An agent's active motivations are specified, in the *GoalGeneration* schema, to be those whose intensity is greater than or equal to the associated threshold. The set of generated goals is obtained by applying the *generateGoal* function for each active motivation, and the resultant set is added to the agent's existing goals.

$$gAssociation == \mathbb{P} Belief \times Motivation \times \mathbb{P} Goal$$

GoalGenerationAssociation

gAssociations : $\mathbb{P} gAssociation$

$\forall B : \mathbb{P} Belief; m : Motivation; G : \mathbb{P} Goal; a : gAssociation \bullet$

$a = (B, m, G) \wedge a \in gAssociations$

$\Rightarrow (\forall H : \mathbb{P} Goal; a' : gAssociation \mid a' = (B, m, H) \wedge$
 $a' \in gAssociations \bullet G = H)$

<i>GoalGeneration</i>
$\Delta Agent$
$activeMotivations : \mathbb{P} Motivation$
$generatedGoals : \mathbb{P} Goal$
$activeMotivations =$
$\{m : Motivation \mid m \in motivations \wedge m.intensity \geq m.threshold\}$
$generatedGoals =$
$\cup \{G : \mathbb{P} Goal \mid (\exists m : Motivation \mid m \in activeMotivations \bullet$
$G = generateGoals\ m\ believes\ motivations)\}$
$goals' = goals \cup generatedGoals$
$beliefs' = beliefs$
$intentions' = intentions$
$motivations' = motivations$

4.7 Ensuring Intentions are Appropriate

Just as an agent drops unmotivated goals, it must also drop any commitment to achieving unmotivated intentions. Recall that according to our definition of intention, an agent should drop its intentions should they become achieved, unachievable, irrelevant, or of no motivational value. Now, since the information the agent has about its environment, i.e. its beliefs, may have changed, the agent must check to see which, if any, of these *drop conditions* apply. It is a straightforward task for an agent to check whether its intentions, or rather the goals they achieve, have been achieved. The agent must simply check whether the situation it intends to bring about is achieved, either by its own action, or through some other agent. If the intention is achieved the agent should drop it³. Checking to see whether an intention is unachievable is more difficult, since a greater amount of reasoning is required. If the agent explicitly believes that its goal can never be achieved, then it clearly must drop its intention. An example of how this might occur is if an agent acquires information provided by another about the futility of one of its goals. However, if an agent has no explicit beliefs about the unachievability of its intentions, then it must perform further reasoning about its beliefs and

³Note that if an agent drops an intention because it has been achieved though the actions of another, we would *not* say that the agent necessarily achieved its goal intentionally.

Inputs:

intentions — the agent's intentions

motivations — the agent's motivations

Outputs:

intentions' — an updated set of intentions

Algorithm:

intentions' = *intentions*

for *i* **in** *intentions* **do**

keepintention = *false*

if not *achieved(i)* **and not** *unachievable(i)* **and** *relevant(i)* **then**

for *m* **in** *motivations* **do**

if *mitigationalValue(i, m)* > 0 **and** *intensity(m)* > 0 **then**

keepgoal = *true*

if not *keepgoal* **then**

drop(i, intentions')

return *intentions'*

Table 4.7: Algorithm for dropping inappropriate intentions

the inferences it can make from them, to determine whether its goal is unachievable. As we are not concerned with how an agent should reason about inferences from its beliefs in our system, we take the simplistic approach of checking explicit beliefs. However, should an agent be given such reasoning abilities, it would be a trivial matter to integrate them into the agent architecture, since it does not affect the form of the agents reasoning cycle.

If an intention is not achieved or believed unachievable, then its relevance condition must be checked, and to do this the agent must simply check that the relevance condition is still believed. If it is not believed then the agent must drop its intention as being irrelevant. Finally, an agent must ensure that its intention is still motivated, i.e. is of value to at least one active motivation, since motivations determine how valuable a given goal is to the agent at a particular time. If an intention is not motivationally valuable then it should be dropped, along with the goal that it achieves — the algorithm for ensuring that intentions are appropriate is given in Table 4.7.

In the following schema, *DropInappropriateIntentions*, we specify the conditions under which an agent should drop an intention. The predicate part specifies, firstly, that an intention is relevant only while the relevance condition is believed and, secondly, that an intention is motivated if and only if it is of value to a motivation of positive intensity. An agent's intentions are updated such that any that are achieved, unachievable, irrelevant, or unmotivated are dropped.

<i>DropInappropriateIntentions</i>
Δ_{Agent}
$isAchieved : Intention \rightarrow \mathbb{P} Belief \rightarrow bool$
$isAchievable : Intention \rightarrow \mathbb{P} Belief \rightarrow bool$
$isRelevant : Intention \rightarrow bool$
$isMotivated : Intention \rightarrow bool$
$\forall i : Intention \mid i \in intentions \bullet isRelevant\ i = true$ $\Leftrightarrow (\forall b : Belief \mid b \in i.relevance \bullet b \in beliefs) \wedge$ $isRelevant\ i = false \Leftrightarrow (\exists b : Belief \mid b \in i.relevance \bullet b \notin beliefs)$
$\forall i : Intention \mid i \in intentions \bullet isMotivated\ i = true$ $\Leftrightarrow (\exists m : Motivation \mid m \in motivations \bullet$ $m.intensity > 0 \wedge (mitigation\ m\ i.satisfies) > 0) \wedge$ $isMotivated\ i = false \Leftrightarrow (\forall m : Motivation \mid m \in motivations \bullet$ $mitigation\ m\ (i.satisfies) = 0 \vee m.intensity = 0)$
$\forall i : Intention \mid i \in intentions \bullet i \in intentions'$ $\Leftrightarrow isAchieved\ i\ beliefs = false \wedge isAchievable\ i\ beliefs = true \wedge$ $isRelevant\ i = true \wedge isMotivated\ i = true \wedge$ $i \notin intentions'$ $\Leftrightarrow isAchieved\ i\ beliefs = true \vee isAchievable\ i\ beliefs = false \vee$ $isRelevant\ i = false \vee isMotivated\ i = false$
$beliefs' = beliefs$
$goals' = goals$
$motivations' = motivations$

4.8 Intention Adoption

Once the agent has dropped inappropriate intentions, it must determine whether to adopt new intentions for the goals to which it is not committed, and if so, it must create suitable intentions for them. Now, one of the requirements of intentions described by Bratman [2]

and Cohen and Levesque [15] is that intentions should be consistent, or rather, they should not knowingly be inconsistent. For example, I should not intend to eat out tonight and not eat out tonight, while I might intend to eat out tonight and write a paper, even if these later turned out to be incompatible. Thus, an agent cannot simply adopt intentions for its newly generated goals, since they may contain inconsistencies. As we require our agents to be driven by their motivations, these motivations play a key role in determining which intentions it adopts. For each of its active motivations an agent attempts to adopt an intention for the goal generated by that motivation, or for the most motivated goal if more than one goal is generated. When incompatibilities are found they must be resolved in such a way as to afford the highest motivational value to the agent. This can be determined simply by considering which motivation has the highest intensity (although an alternative approach would be to consider a combination of motivational effect and motivational intensity).

4.8.1 Selecting and Adopting a Plan

To adopt an intention an agent must select a plan to use to achieve its goal. From its plan library an agent selects the set of applicable plans to achieve each of its chosen goals⁴. The set of applicable plans for a goal is defined to be those plans that achieve the goal whose preconditions are met, as specified below in function *planSetForGoal*. We define *planSetForGoal* through the use of a subsidiary function, *preconMet*, which takes a set of preconditions and a set of beliefs, and returns true if and only if the preconditions are believed to be true (as described in Appendix A).

$$\frac{\begin{array}{l} \textit{planSetForGoal} : \textit{Goal} \rightarrow \mathbb{P} \textit{Belief} \rightarrow \mathbb{P} \textit{Plan} \rightarrow \mathbb{P} \textit{Plan} \\ \forall g : \textit{Goal}; \textit{bel} : \mathbb{P} \textit{Belief}; \textit{plib} : \mathbb{P} \textit{Plan} \bullet \textit{planSetForGoal} \ g \ \textit{bel} \ \textit{plib} \\ = \{p : \textit{Plan} \mid p \in \textit{plib} \wedge p.\textit{achieves} = g \wedge \\ \textit{preconMet} \ p.\textit{preconditions} \ \textit{bel} = \textit{true}\} \end{array}}{}{}$$

⁴Note that we assume there is at least one plan in the plan library for each of the goals that an agent might generate. This is the responsibility of the agent designer, and we do not consider how an agent should behave if it cannot find at least one applicable plan for a given goal.

Once an agent has obtained the set of applicable plans for a given goal and context, it must choose one, and there are many possible criteria for selecting a plan from a set of applicable plans. For example, the agent may choose the plan containing the minimum number of subgoals or the one with the minimum number of actions, i.e. the plan that seems to require the least further reasoning or action respectively. Alternatively, it may select from a set of plans by considering the joint and concurrent actions they contain and the agents with whom cooperation may occur. Without fully elaborating the applicable plans, however, an agent cannot be certain about its choice, since it does not know which subplans will be used in the plan’s elaboration. Moreover it is also unable to predict the way the environment may change, so cannot elaborate the applicable plans to select between them⁵ and must, therefore, use some heuristic to choose between plans. We discuss in detail how agents can choose between applicable plans in the Chapter 6. However, in order to continue with our specification of the agent architecture, we introduce the function *planForGoal* that takes a set of applicable plans and chooses the most appropriate depending on the agent’s current beliefs and intentions. Beliefs and intentions constrain the plans that are applicable, since a chosen plan must be consistent both with the agent’s beliefs and its existing intentions. Since we describe in Chapter 6 the mechanisms an agent might use to chose a particular plan, we simply specify a function signature here, and describe its instantiation later.

| *planForGoal* : $\mathbb{P} \textit{Belief} \rightarrow \mathbb{P} \textit{Intention} \rightarrow \mathbb{P} \textit{Plan} \rightarrow \textit{Goal} \leftrightarrow \textit{Plan}$

Thus, intention adoption comprises the two stages of first determining which goals to adopt intentions for, and then selecting plans for those goals. This is done by resolving incompatibilities between goals to get a set of compatible motivated goals to which an agent can commit. Then plans are selected for each of these intentions (and checked for consistency with each other), before forming intentions. The algorithm for intention adoption is shown in Table 4.8. We do not include a specification of the relevance conditions since they are dependent on the domain and situation, and so cannot be specified.

⁵Such elaboration would also place a significant burden on the agent’s resources, and may prevent it from making a timely decision.

Inputs:

motivations — the agent's motivations

intentions — the agent's intentions

goals — the agent's goals

beliefs — the agent's beliefs

Outputs:

intendedgoals' — an updated set of intended goals

intentions' — an updated set of intentions

Algorithm:

activeM = empty

activegoals = empty

intendedgoals = empty

for *m* **in** *motivations* **do**

if $\text{intensity}(m) \geq \text{threshold}(m)$ **then**

activeM = union(*activeM*, {*m*})

for *m* **in** *activeM* **do**

for *g* **in** *goals* **do**

if $\text{mitigationalValue}(g, m) > 0$ **then**

isactive = true

for *g'* **in** *goals* **do**

if $\text{mitigationalValue}(g, m) <$

$\text{mitigationalValue}(g', m)$ **then**

isactive = false

if *isactive* **then**

activegoals = union(*activegoals*, { *g* })

for *i* **in** *intentions* **do**

intendedgoals = union(*intendedgoals*, { achieves(*i*) })

intendedgoals' = resolveIncompatibilities(*intendedgoals*, *activegoals*)

intentions' = *intentions*

for *g* **in** *intendedgoals'* **do**

if not *g* **in** *intendedgoals* **do**

p = planForGoal(*beliefs*, *intentions*, planlib, *g*)

newintention = adopt(*p*)

intentions' = union(*intentions'*, { *newintention* })

return *intendedgoals'*, *intentions'*

Table 4.8: Algorithm for intention adoption

Formally, we specify the stages of intention adoption in the schema *IntentionAdoption*. The set of active motivations is used to determine the set of active goals, which is constructed such that for each motivation the goal that is of the most motivational value is a member of the set of active goals. Before adopting intentions for active goals (that are not already committed to), any inconsistencies with the agent's current intentions must be resolved, and the set *newIntendedGoals* represents the resulting set of goals, for which the agent must have corresponding intentions. Intentions are then formed for any goals in this set that are not already committed to, using the plan returned by the *planForGoal* function.

<i>IntentionAdoption</i>
Δ_{Agent} <i>activeMotivations</i> : \mathbb{P} Motivation <i>activeGoals</i> : \mathbb{P} Goal <i>currentIntendedGoals</i> : \mathbb{P} Goal <i>newIntendedGoals</i> : \mathbb{P} Goal <i>resolveIncompatibilities</i> : \mathbb{P} Goal \rightarrow \mathbb{P} Goal \rightarrow \mathbb{P} Goal
<hr/> <i>activeMotivations</i> = $\{m : \text{Motivation} \mid m \in \text{motivations} \wedge m.\text{intensity} \geq m.\text{threshold}\}$ <i>activeGoals</i> = $\{g : \text{Goal} \mid g \in \text{goals} \wedge$ $(\exists m : \text{Motivation} \mid m \in \text{activeMotivations} \wedge \text{mitigation } m \ g > 0 \bullet$ $(\forall g' : \text{Goal} \mid g' \in \text{goals} \wedge g' \neq g \bullet$ $(\text{mitigation } m \ g) \geq (\text{mitigation } m \ g')))\}$ <i>currentIntendedGoals</i> = $\{g : \text{Goal} \mid (\exists i : \text{Intention} \mid i \in \text{intentions} \bullet i.\text{satisfies} = g)\}$ <i>newIntendedGoals</i> = <i>resolveIncompatibilities</i> <i>currentIntendedGoals</i> <i>activeGoals</i> <i>intentions'</i> = $\{i : \text{Intention} \mid i.\text{satisfies} \in \text{newIntendedGoals} \wedge$ $i \in \text{intentions}\} \cup \{i : \text{Intention} \mid i.\text{satisfies} \in \text{newIntendedGoals} \wedge$ $i \notin \text{intentions} \wedge \text{head}(i.\text{plans}) =$ $\text{planForGoal } \text{beliefs } \text{intentions } \text{planLibrary } i.\text{satisfies}\}$

There are two possible strategies that the agent could adopt to forming an intention, depending on the point at which it commits itself to a particular course of action, and chooses a particular plan. The simplest strategy is for the agent to select the most appropriate plan from its plan library given its current situation, and use that plan to form its intention. We call this an *immediate elaboration strategy*, since the agent commits itself to a particular

plan at the point of intention formation. Alternatively, it can form an intention using an *abstract* plan, which is one that contains the single step of a subgoal. Thus, the agent is able to form an intention without elaborating its plan, and since the agent postpones committing to a particular course of action (namely, a particular elaboration of its plan) we call this a *delayed elaboration strategy*.

4.8.2 Immediate Elaboration

Based on its motivations, an agent selects a goal (and therefore an intention) to act upon, which is to be achieved by executing the plan described in the corresponding intention. If the selected intention was formed using an immediate elaboration strategy then the agent is able to begin execution of the plan immediately. Where the first step of the plan is an action, the agent can perform it (if its preconditions are met) and where it is a subgoal the agent can elaborate its plan by selecting a subplan for the subgoal.

If the agent is the only entity able to act in its environment, and is constrained so that it is prevented from having more than a single goal (or intention) at any one time, then the immediate elaboration strategy is adequate. However, typically an agent has more than one goal (and corresponding intentions), and shares its environment with other acting entities, so that plans elaborated at the point of intention formation may not be executable by the time the agent chooses to act upon them, since the environment may have changed. Suppose an agent has two goals, g_1 and g_2 , and uses an immediate elaboration strategy to form respective intentions, i_1 and i_2 , which contain (at least partially) elaborated plans. Based upon its motivations the agent chooses an intention to act upon, say i_1 , and begins to execute the plan contained in that intention. Now, suppose that the agent's motivations are such that i_1 remains the chosen intention until it is completed (and g_1 is achieved), at which point i_1 and g_1 are dropped (assuming it was successfully achieved). Thus, on achieving g_1 the agent is left with a single goal and intention, g_2 and i_2 respectively. However, since the plan in i_2 was chosen before g_1 was achieved, it may not be relevant in the current environment,

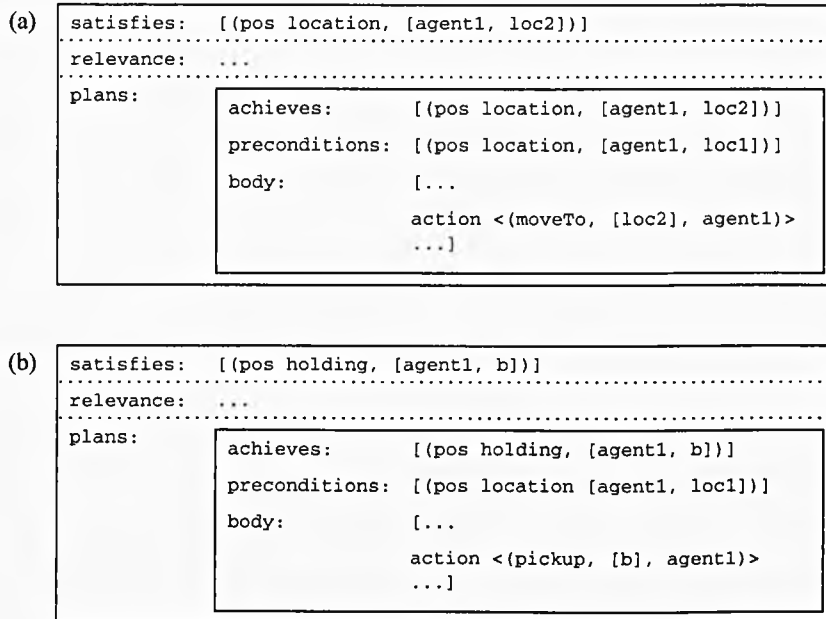


Figure 4.2: The problem of plan over-commitment

and therefore i_2 may be unachievable (unless the environment changes). We say that an agent in this situation suffers from the problem of *plan over-commitment*⁶.

This problem can be seen more clearly with a simple example, where an agent (with agent identifier $agent1$) is situated in some location loc_1 , in which there is a box b . Suppose that g_1 corresponds to being in another location loc_2 written $location(agent, loc_2)$, and g_2 to holding the box written $holding(agent, b)$. Suppose that the agent forms intentions, i_1 and i_2 , for these goals, and uses an immediate elaboration strategy to select plans that require the environment to be in this situation for their execution, as shown in Figure 4.2 (a) and (b) respectively. Once the agent has executed the plan in i_1 , the environment will have changed such that the agent is in location loc_2 . Since i_2 is now the agent's sole intention it will try to act to achieve it. However, since the plan component of i_2 was selected, the environment has

⁶This is analogous to the problem of *goal clobbering* in planning [86].

changed and the plan is no longer applicable (since it requires the agent to be in the same location as the box, namely loc_1). Thus, the agent is forced to either drop this intention, and form a new one with a new plan, or wait until it is in loc_1 again. As there is no guarantee that the environment will return to its previous state, the most viable approach is to drop i_2 , and form a new intention given the current situation.

4.8.3 Delayed Elaboration

If the agent were to use a delayed elaboration strategy instead, then the risk of plan over-commitment is reduced, since the elaboration of the plan component of an intention is delayed until the agent comes to execute the plan. In the example above, *delayed elaboration* would entail using abstract plans rather than elaborated plans, and on completion of i_1 and g_1 the agent is again left with a single intention, i_2 , but with a corresponding *abstract* plan that must be elaborated before the agent can act. The agent can now select a plan for i_2 (based on the current state of the environment) and execute it, without the need to drop the existing intention. Thus, the problem of plan over-commitment is avoided in this simple scenario.

The example above is simplified in that we assume that there is only one agent acting in the environment, and that once it begins to act on an intention it will continue until the intention is complete. Where the environment is more dynamic, or there is more than one agent active in it, the problem of plan over-commitment is more likely to occur (regardless of whether the agent uses an immediate or a delayed elaboration strategy), since the environment will change of its own accord, or as a result of others' actions. During the execution of a plan, such changes may render the remainder of the plan not executable. For example, if an agent adopts a plan to pick up a box, another agent might pick it up before it executes its plan, thereby preventing execution of the plan. Similarly, if the agent's motivations are such that its attention changes between different partially completed intentions, the agent itself may cause a plan in one of its intentions no longer to be executable, due

to changes it has made in the environment. In these more complex situations there are too many factors involved to estimate the risk of plan over-commitment for either strategies.

In terms of this thesis, we consider cooperation resulting from agents soliciting assistances with respect to a particular plan, as we describe in Chapter 7. It is simpler for an agent to solicit assistance with respect to a *particular* plan, and therefore where cooperation is needed, the immediate elaboration approach is used. Moreover, since we are concerned with cooperation rather than the construction of an optimal mechanism for intention adoption, we assume that agents always use the immediate elaboration approach. In the worst case scenario, this choice means that the agent will have to re-adopt its intention with a different plan, and we consider this cost to be relatively low given the added simplicity of always using the same approach to form an intention.

4.9 Intention Selection

In order to act, an agent must select an intention to focus upon. As before, motivations play a fundamental role here in that the intention whose achievement is of the most value in motivational terms is selected. As we described in Section 3.13, motivational value is dependent on the intensity of the motivation concerned, and an intention that is of high relevance for a motivation with minimal intensity might be said to have *less* motivational value than an intention with relevance for a motivation of large intensity. Therefore, the intensity of the agent's motivations is crucial in selecting an intention to focus on — the agent should select an intention that favours its most significant motivation. The most significant motivation is defined as being the active motivation with the highest intensity if any active motivations exist, otherwise it is simply the motivation with the highest intensity (recall that a motivation is said to be active if its intensity is greater than or equal to its threshold). Once the agent has determined its most significant motivation, it must then select which of its intentions contribute the most to it. This is determined by the motivational effect the achievement of an intention would have on the motivation, and the agent should select

Inputs:

motivations — the agent’s motivations

intentions — the agent’s intentions

Outputs:

chosenintention — the selected intention

Algorithm:

activeM = empty

for *m* **in** *motivations* **do**

if $\text{intensity}(m) \geq \text{threshold}(m)$ **then**

activeM = union(*activeM*, {*m*})

if *activeM* = empty **then**

M = *motivations*

else

M = *activeM*

m = selectRandom(*M*)

for *m'* **in** *M* **do**

if $\text{intensity}(m') > \text{intensity}(m)$ **then**

m = *m'*

highestM = *m*

chosenintention = selectRandom(*intentions*)

for *i* **in** *intentions* **do**

if $\text{mitigationalValue}(i, \text{highestM}) >$

$\text{mitigationalValue}(\text{chosenintention}, \text{highestM})$ **then**

chosenintention = *i*

return *chosenintention*

Table 4.9: Algorithm for intention selection

the intention that has the most motivational effect on its most significant motivation. We call this intention the *chosen* intention. This algorithm for intention selection is shown in Table 4.9.

We specify this below in the schema *IntentionSelection*, where the first three predicates in the schema determine the most significant motivation. The final predicate specifies that the agent should select the intention that is of the most motivational value to the most significant motivation.

IntentionSelection

$\exists Agent$

activeMotivations : $\mathbb{P} Motivation$

chosenMotivation : *Motivation*

chosenIntention : *Intention*

activeMotivations =

$\{m : Motivation \mid m \in motivations \wedge m.intensity \geq m.threshold\}$

activeMotivations $\neq \emptyset \Rightarrow (\exists m : Motivation \mid m \in activeMotivations \bullet$

$(\forall m' : Motivation \mid m' \in activeMotivations \wedge m' \neq m \bullet$

$m.intensity - m.threshold \geq m'.intensity - m'.threshold \wedge$

$chosenMotivation = m))$

activeMotivations = $\emptyset \Rightarrow (\exists m : Motivation \mid m \in motivations \bullet$

$(\forall m' : Motivation \mid m' \in motivations \wedge m' \neq m \bullet$

$m.intensity - m.threshold \geq m'.intensity - m'.threshold \wedge$

$chosenMotivation = m))$

$(\exists_1 i : Intention \mid i \in intentions \bullet (\forall i' : Intention \mid i' \in intentions \wedge$

$i \neq i' \bullet mitigation chosenMotivation i.satisfies \geq$

$mitigation chosenMotivation i'.satisfies \wedge$

$chosenIntention = i))$

4.10 Action and Deliberation

After determining its chosen intention, an agent acts towards it, but the way in which the agent does this depends on the contents of the intention or, more specifically, on the plan component of the intention. If the first step in the plan is an individual action contribution then the agent can execute that action. If the step is a subgoal then the agent must elaborate the plan, and choose a subplan for that subgoal — we call this process *deliberation*. However, if the first step is a joint or concurrent action then the agent must seek assistance from others, and this is covered in detail in Chapter 7.

If the first step in the body of the plan at the top of the intention stack is an action, then the agent executes this action, and the action is removed from the plan. On executing an action the environment is changed in the manner defined by the function *contributionEffects*, as introduced in Section 3.9. The agent's beliefs are also updated, since it can infer that

the environment has changed in the manner defined by *contributionEffects*. For the purpose of the specification, we assume the existence of the function *believedChanges* which takes a contribution and an environment, and returns a set of beliefs representing the changes to the environment that performing the contribution produces. We also introduce in the *AgentHistory* schema a sequence of contributions that records the contributions performed by a particular agent. An agent acts by performing the first contribution step in chosen intention, changing the environment and appending this contribution to its history. The agent's beliefs are updated to include information about the effects of performing this contribution, as determined by *believedChanges*.

| $believedChanges : Contribution \rightarrow Environment \rightarrow \mathbb{P} Belief$

<p><i>AgentHistory</i> <i>history</i> : seq <i>Contribution</i></p>
--

<p><i>AgentAction</i> $\Delta Agent$ ΔEnv $\Delta AgentHistory$ $\exists IntentionSelection$ $\exists UpdateBeliefs$ <i>nextStep</i> : <i>PlanStep</i></p> <p><i>nextStep</i> = <i>head</i>(<i>last chosenIntention.plans</i>).<i>body</i> $\exists a : Contribution \bullet Individual(a) = nextStep$ $\Leftrightarrow history' = history \hat{\ } \langle a \rangle \wedge environment' =$ $contributionEffects\ a\ environment \wedge$ $beliefs' = reviseBeliefs\ beliefs\ (believedChanges$ $a\ environment)$</p>
--

If the first step in the body of the plan at the top of the intention stack is a goal, then the agent selects a subplan to achieve it from its plan library. This process is the same as that for forming an intention, except that once the agent has selected an appropriate plan it is then pushed onto the intention stack, rather than a new intention being formed. In the following schema we specify that where the next step of an intention is a subgoal, the best

plan is selected for that goal using *planForGoal*, and pushed onto the intention stack. The relevance condition and the goal the intention satisfies are unchanged.

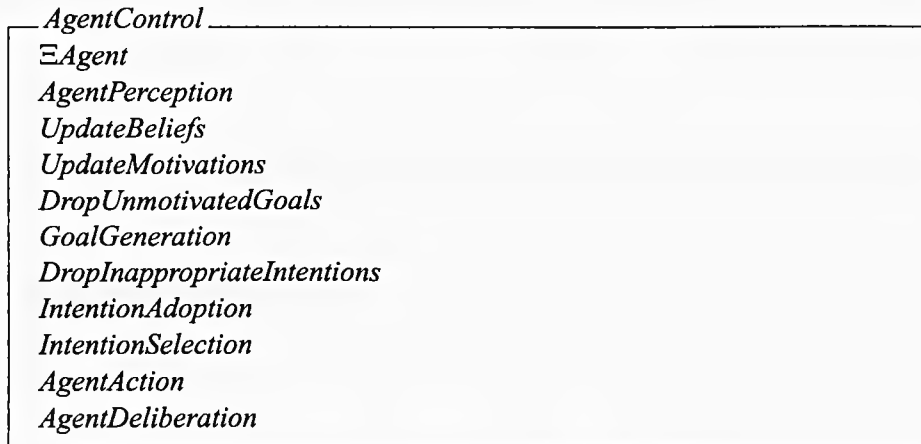
<p><i>AgentDeliberation</i></p> <p>$\Delta Agent$</p> <p>$\exists IntentionSelection$</p> <p><i>nextStep</i> : <i>PlanStep</i></p> <hr/> <p><i>nextStep</i> = <i>head</i>(<i>last chosenIntention.plans</i>).<i>body</i></p> <p>$\exists g : Goal \bullet Subgoal(g) = nextStep$</p> <p>$\Leftrightarrow chosenIntention'.plans = chosenIntention.plans$</p> <p>$\wedge ((planForGoal\ beliefs\ intentions\ planLibrary\ g))$</p> <p>$\wedge chosenIntention'.relevance = chosenIntention.relevance$</p> <p>$\wedge chosenIntention'.satisfies = chosenIntention.satisfies$</p>
--

4.11 Summary

In this chapter we have completed our description of SENARA, our BDI-based architecture for cooperation between autonomous agents. The BDI model, as exemplified by Bratman *et al.* [5], does not directly support the mechanisms required for agents to be truly autonomous and have full control over the direction of their own behaviour. Additionally, it does not provide a mechanism for an agent's plans to include cooperative actions that are to be performed in conjunction with other agents. Therefore, we have described extensions to this BDI model in two key areas: the addition of *motivations* to agents, and the addition of *joint* and *concurrent* actions.

The first significant extension is the addition of the mental component of *motivations* to the existing ones of beliefs, desires (i.e. goals), and intentions. Motivations serve to allow an agent to guide its behaviour, both through the generation and adoption of appropriate goals, and through influencing the decisions an agent makes in the course of its operation. For example, an agent must consider its motivations in choosing a plan for a goal, and in choosing which agents to cooperate with, thereby ensuring that it always acts to afford itself the greatest motivational value. A result of introducing motivations into the architecture is

that the agent reasoning cycle must be extended. In this chapter we have described the extensions necessary to allow an agent to update the intensities of its motivations according to its beliefs, generate goals from these motivations, choose a goal to pursue, adopt an appropriate intention from its goals, and select an intention to act towards. These mechanisms are brought together in the following schema, which specifies an agent's control mechanisms.



Joint and concurrent actions provide a mechanism for representing composite actions made up of contributions that are to be performed by individual agents. These composite actions can be included as steps in an agent's plans, and so allow agents to be given predetermined plans of how to achieve goals through cooperation. In Bratman *et al.*'s BDI abstract architecture an agent's plans are only able to represent the actions it should perform individually, rather than those it should perform as part of a cooperative activity.

Chapter 5

Autonomous Cooperation in Open Environments

5.1 Introduction

In the preceding chapters we introduced the mental components and control mechanisms that comprise the agent architecture upon which we base our framework of cooperation. However, we have not yet considered the process of cooperation amongst such agents. Agents act according to their motivations and therefore will only cooperate if they receive motivational value from doing so. Cooperation between autonomous agents involves certain choices about when to cooperate, with whom, and for how long, and the outcome of these choices is determined by an agent's motivations. Consequently cooperation implies an inherent degree of risk, since where a group of agents cooperate towards achieving a particular goal, any one of them may cease to cooperate if it is no longer of motivational value, regardless of the effect this has on the achievement of the goal. In order to cooperate effectively, an agent must be able to make appropriate choices about cooperation and manage the associated risk. This chapter seeks to provide the context for the instantiation of the architecture presented in the preceding chapters to address these issues.

Specifically, we start this chapter by describing the nature of the commitments required by a group of agents to cooperatively execute a plan, and we introduce the stages involved in such cooperation activity. We then introduce the notion of *trust* as a means of modelling the perceived risk associated with other agents. Finally, for ease of understanding, it is useful to have a particular domain to provide a context for discussing our framework, and in which we can give an example instantiation; we end this chapter by describing such a domain.

5.2 Cooperative Intention

Cooperative action by a group of agents consists of the individual actions of its members and, because the actions of individuals are determined by their intentions (according to the SENARA control cycle), those agents must adopt appropriate intentions before acting cooperatively. However, it is not sufficient for each agent simply to adopt an intention to achieve the cooperative goal, since each agent's intention is independent of the others, and the success of one agent is unrelated to the success of another. Cohen and Levesque illustrate this through the example of a group of people running for shelter under a tree in a rainstorm [19] — they all have the same goal and intention (to run under the tree), but there is no cooperation amongst them. Indeed, if there is limited shelter under the tree, their intentions may actually lead to competitive behaviour. Therefore, as is widely recognised (e.g. [4, 58, 100, 102]), cooperation requires some form of group commitment towards the cooperative goal, that is more than just a collection of individual intentions and embodies the notion of agents acting *together*. We call our notion of this group commitment a *cooperative intention* (thereby distinguishing it from other related work on group commitments, such as that of Cohen and Levesque [58], and Bratman [4]), and discuss its requirements and definition in this section.

5.2.1 Requirements of a Model of Cooperation

Cooperation involves more than just the simultaneous actions of a group of individuals, as illustrated above, and some form of cooperative intention is required. However, since a group of agents may diverge in their beliefs, a group's intention to cooperate cannot simply be a version of individual intention where the group is taken to be the agent. It is suggested by Bratman [4] that agents are rational entities, and must not be committed to achieving a goal that they believe unachievable (or already achieved, or irrelevant). Therefore, if a cooperating agent comes to believe that one of these conditions is the case with respect to the goal for which it is cooperating, then it will cease to cooperate, and should inform others. We concur with this view, and adopt it as one of the key requirements for a group's cooperative intention. Moreover, in our view, agents act according to their motivations, and if cooperation is of motivational value they will cooperate, otherwise they will not. Thus, if the motivations of a member of a cooperating group change such that cooperation is no longer of motivational value, then that agent will cease to cooperate since it is not motivationally beneficial to continue, and it must inform others.

Even where agents take diverse roles in the achievement of their goal, the commitments they hold should still be towards a common course of action. Commitment alone is insufficient for cooperation; it must be towards a specific (possibly partially determined) *course of action*, otherwise agents could be committed to achieving a goal through conflicting means. Cooperation, therefore, requires agents to act together through a common approach towards a common goal. For example, if two agents are committed to achieving the goal of moving a table together from one room to another, they must be committed to using a particular approach — if one agent is committed to performing the action of lifting one end of the table and carrying it, while the other is committed to dragging the table then their actions will not result in cooperation, and they will not achieve their goal.

Agents should not be forced into commitment; in our view, agents are autonomous entities and have control over their own behaviour, and while an agent may attempt to gain

the assistance of another (for example by offering to reciprocate) it cannot force another to cooperate. Consequently, agents should only enter into cooperative activity, and commit to a cooperative intention if it is in their own interests to do so — if it is of motivational benefit. Furthermore, the intention must be common knowledge amongst the group, thereby allowing agents to reason about their commitments in the light of others' corresponding commitments. In particular, before performing its part of a cooperative interaction an agent needs to know that others will do likewise.

We can summarise these requirements for cooperation as follows, based on those given by Bratman [4]. The key difference between our view and Bratman's is that we require a cooperative intention to be *motivated* on behalf of the agents concerned.

Commitment to openness There must be some mechanism for an agent to inform others if it comes to believe that cooperation is no longer appropriate — when the goal is believed to be achieved, unachievable, irrelevant, or unmotivated.

Commitment to a common means A group's commitment must be towards a specific course of action, otherwise agents might become committed to achieving a goal through conflicting means.

Common Knowledge A group's commitment must be common knowledge amongst its members.

Motivated commitment Commitments must be motivated on the behalf of the cooperating agents.

In addition to these requirements it is also *desirable* that cooperating agents guide their behaviour in response to others' intentions and actions, and are committed to supporting others; cooperative intention should lead to *mutual responsiveness* and *mutual support* between agents. For example, if a group member cannot perform its contribution to a joint action, and a second member is in a position to perform some action after which the contribution could be performed, then the second agent's commitment to the joint intention

should lead it to act so as to enable the cooperative action to proceed (assuming it is of motivational value to do so).

5.2.2 Conventions for Cooperative Intention

In Chapter 2 we identified two main views on the nature of cooperative intention, where it is either viewed as being irreducible, or as comprising an appropriate combination of individual intentions, mutual beliefs, and a set of mechanisms describing how it should be maintained [4, 58, 100]. In both approaches, action is seen as governed by intention, and so some relationship between cooperative and individual intention must exist, otherwise no action would be performed. However, the former approach views the required individual intentions as *arising* out of the cooperative intention, while the latter approach views these individual intentions, in part, as *defining* the cooperative intention. It is our view that since cooperative intention can arise only from the interaction of individuals, the components that comprise it must be formed from the mental components and attitudes of those individuals. In this thesis, therefore, we adopt the view that a cooperative intention comprises a set of individual intentions, mutual beliefs, and mechanisms describing its operation. A useful consequence of taking this approach is that it is consistent with the majority of existing computational work, with the alternative generally confined to philosophical investigations. Our model of cooperation can, therefore, be more easily compared, and integrated, with other existing computationally-oriented work.

Recall from Section 2.5.3 that a *convention* specifies the conditions under which a commitment can be abandoned, and how an agent should behave in such a circumstance. We follow Wooldridge and Jennings' notion of cooperative intention as a commitment to a course of action with associated conventions (as introduced in Chapter 2). However, Wooldridge and Jennings' associated model of cooperation is somewhat abstract, and they do not, therefore, give the details of such issues as how agents should form and represent plans [102, 104]. Our concern in this thesis is cooperation between motivated BDI-like

re-evaluation condition	goal
believe that goal is achieved	establish mutual belief that goal is achieved
believe that goal is unachievable	establish mutual belief that goal is unachievable
believe that goal is no longer relevant	establish mutual belief that goal is no longer relevant
believe that goal is not of motivational value	establish mutual belief that goal is not of motivational value

Table 5.1: Conventions for motivated cooperation

agents, and we must consider these architecture-specific details. In the case where plan execution requires cooperation, the commitment involved is a group commitment amongst agents, initiated by the agent that selected the plan. In contrast to the notion of *joint intention* as a commitment towards a particular fully-elaborated sequence of actions, we are concerned with commitment to a partial plan containing subgoals. Thus, we relax Wooldridge and Jennings' definition slightly, so that joint intention is viewed as a joint commitment towards a *partial* plan, rather than a specific action sequence.

More importantly, since agents are guided by their motivations, our notion of cooperative intention must reflect the importance of an agent's motivations, in that agents must ensure that their actions and commitments are of motivational value, i.e. agents engage in *motivated* cooperation. Thus, an agent should only form an intention, whether individual or cooperative, if its formation gives rise to motivational benefit. In other words, an agent should only commit to executing a particular plan to achieve a goal if the achievement of the goal offers motivational value, or if the execution of the plan itself provides motivational benefit. For example, achieving the goal of emptying a glass by drinking the beer in it rather than throwing the beer away offers motivational value through the action of drinking beer, rather than actually emptying the glass. The notion of motivational benefit is also important in defining the duration of a cooperative intention, since an agent should drop any intention

-
1. All agents initially believe that the goal is not satisfied, and that it is achievable.
 2. Each agent has the goal until the termination condition is satisfied.
 3. Until the termination condition is satisfied,
 - (a) if any agent believes the goal is achieved, it should adopt the new goal of making this mutually believed, and keep this new goal until the termination condition is satisfied;
 - (b) if any agent believes the goal is unachievable, then it should adopt the goal of making this mutually believed, and keep this new goal until the termination condition is satisfied;
 - (c) if any agent believes the goal is no longer relevant, then it should adopt the goal of making this mutually believed, and keep this new goal until the termination condition is satisfied; and
 - (d) if any agent believes the goal is no longer of motivational value, then it should adopt the goal of making this mutually believed, and keep this new goal until the termination condition is satisfied.
 4. The termination condition is that it is mutually believed that the goal is achieved, unachievable, is no longer relevant, or the goal is no longer of motivational value to one or more of the agents in the group.
-

Table 5.2: Observations about group mental state after Wooldridge and Jennings

that is not of motivational benefit, as per an individual intention (described in Section 4.7 where we discuss the conditions under which an individual intention is discharged). If an agent's cooperative intention ceases to be of motivational value, it should drop that intention (informing the other members of the group that it is doing so).

With this in mind, we can consider the conventions defined by Wooldridge and Jennings for joint intention, which include the requirement that the goal should be *justified*. This is a broader condition than the goal being of motivational value, since there is no constraint on what may be used as justification; although requiring a goal to be motivationally valuable is valid justification, such a justification need not necessarily refer to an agent's motivations. Therefore, we modify Wooldridge and Jennings' notion of cooperative intention to require that a goal be *relevant* and of *motivational value* to an agent committed to it. This can be

thought of as decomposing the convention requiring a goal to be justified into two separate conventions requiring it to be both relevant and of motivational value. The resultant conventions for a cooperative intention are shown in Table 5.1 where for each row, if an agent comes to believe the re-evaluation condition it should adopt the corresponding goal. Such a cooperative intention is only terminated when a re-evaluation condition is believed to be true *and* the corresponding goal is achieved. For example, if an agent believes that the goal is achieved and has made others aware of this, then the cooperative intention is dissolved. The termination condition and mental state of a group that has a cooperative intention is described in Table 5.2.

5.2.3 Formalising Cooperative Intention

We can formalise the notion of a convention as the Cartesian product of a set of beliefs that represent a particular situation, and a goal that an agent must adopt if it believes that situation to be the case.

$$\textit{Convention} == \mathbb{P} \textit{Belief} \times \textit{Goal}$$

We formalise the notion of a cooperative intention in the following schema, which contains the goal and plan to which the commitment is towards, the identifiers of the agents who have the commitment, and a set of conventions (i.e. those given in Table 5.1) defining the duration of this commitment. The predicate part of the schema specifies, firstly, that the plan must achieve the goal, and each agent in the cooperative intention must have a corresponding individual intention towards the execution of the plan. Secondly, if a re-evaluation condition of a convention is believed by some agent, then the agent should adopt the corresponding goal defined in that convention.

CooperativeIntention

goal : *Goal*
plan : *Plan*
agents : \mathbb{P} *AgentID*
conventions : \mathbb{P} *Convention*

$\forall id : AgentID \mid id \in agents \bullet$
 $(\exists ag : Agent \bullet ag.agentID = id \wedge goal \in ag.goals \wedge$
 $(\exists i : Intention \mid i \in ag.intentions \bullet$
 $i.plans \ 1 = plan \wedge i.satisfies = goal)) \vee$
 $(\exists ag : Agent \bullet ag.agentID = id \wedge (\exists c : Convention \mid$
 $c \in conventions \bullet believes \ ag \ (first \ c) = true \wedge$
 $second \ c \in ag.goals \wedge (\exists i : Intention \mid i \in ag.intentions \bullet$
 $i.satisfies = second \ c)))$

5.3 Stages in Cooperation

There are a number of stages that occur in cooperation that surround the formation of a cooperative intention, which we introduce in this section. Given the SENARA reasoning cycle described in the previous chapter, cooperation can arise with respect to a particular agent for one main reason. When selecting a plan to achieve the most motivated of its goals, an agent might be faced with one or more plans that involve joint or concurrent actions, or an individual contribution that is beyond its capabilities. If the agent chooses such a plan, it must seek assistance from others before that plan can be achieved, and form an appropriate cooperative intention. This, however, gives rise to a second reason why an agent might enter into cooperation, namely in response to another's request for assistance. In both cases cooperation arises from a particular agent wishing to adopt a plan that contains actions it is unable to perform alone — in the first case the agent itself has the plan, while in second case it is another agent's plan that leads to the request for assistance. We arrive, therefore, at the following stages of cooperation.

Plan Selection An agent's motivations give rise to certain goals that must be adopted as intentions, by selecting an appropriate plan and forming a commitment to its execution.

Now, the set of applicable plans for a particular goal may include plans containing actions that are beyond the agent's capabilities, or joint or concurrent actions. We refer to such plans as *cooperative plans* since they can only be executed through cooperation with others. If an agent selects a cooperative plan, it is electing to cooperate for the achievement of its goal. Cooperation involves an inherent risk since agents may be unreliable, dishonest, or their motivations may change leading them to cease to be cooperative, in turn causing plan execution to fail. Therefore, in selecting from a set of applicable plans for a goal an agent must consider the risk associated with those plans that are cooperative.

Intention Adoption After selecting a plan for its goal an agent must commit to its execution by forming an intention. If the plan does not require assistance from others then it can simply be adopted as described in Section 4.8, but if it does require assistance the agent must solicit assistance from selected agents towards its execution. Note that for clarity we refer to the agent that selects a cooperative plan, and attempts to gain assistance for its execution, as the *initiating agent*, or the *initiator*. In order to gain assistance, the initiator must first determine which agents to request assistance from, and this is achieved by iterating through the steps of the plan, annotating each contribution with the identifier of the agent that the initiator considers the best to perform it, based on knowledge of their capabilities, and their believed honesty, reliability, etc. These agents can then be sent a request for assistance, to which they will agree if they consider cooperation to be of motivational value. If sufficient agents agree the initiator can form a commitment in the form of a cooperative intention amongst them.

Group Action Once a group of agents have formed a cooperative intention they can execute it. Execution of a cooperative intention is similar to that of an individual intention — each step of the plan in turn is either performed or elaborated according to whether it is an action, or a subgoal respectively. On the successful completion of the cooperative intention, the agents concerned dissolve their commitment and coopera-

tion is finished. Alternatively, if execution of the intention fails, the agent that first comes to believe this informs the others, and again their commitments are dissolved. In both cases agents can update the information they store about others to aid future decisions about cooperation. For example, if cooperation failed due to the behaviour of a particular agent, the others involved may be more wary of cooperating with that agent in future.

These stages are related to Wooldridge and Jennings' four stage model of cooperation which (as described in Section 2.7) contains the stages of: recognition of the potential for cooperation, team formation, plan formation, and team action. Their model is relatively abstract and, as they recognise, is intended to provide a top-level specification for a system, requiring more detail before it can be implemented. Our approach is based on their model, and we view it as providing an instantiation for some of the details that were left abstract. Wooldridge and Jennings also recognise that although the four stages in their model are presented as being sequential, in practice they may not occur strictly in the order they describe. Indeed, this is a significant difference between our model and theirs; in our approach an individual agent selects a plan that requires cooperation, and then seeks assistance, while in their approach an agent recognises the potential for cooperation, seeks assistance, and *then* the agents as a group form a plan.

This difference arises from our alternative view of the *potential for cooperation*, which in turn is a result of the nature of our agent architecture. They view the potential for cooperation as being where an agent has a goal that it is unable to achieve in isolation, or has a goal that it is able to achieve alone, but does not want to use the resources required to achieve it. Alternatively, in our framework the recognition of the potential for cooperation is implicit in an agent's choice of how to achieve its goal — an agent simply selects a plan to achieve its goal, and this plan may or may not require cooperation to execute. Therefore, in our model an agent seeks assistance *after* a plan has been selected, rather than before, since unless an agent knows how to achieve the goal it cannot consider the nature of cooperation

that may occur for that goal. We are specifically concerned with *why* an agent might enter into cooperation, in addition to the process of cooperation itself.

5.4 Risk in Cooperation

In interacting with others, an agent places itself open to a certain degree of risk. In particular, there are two main areas through which risk is introduced. Firstly, there is a risk that agents will not agree to cooperate for a given goal and plan to achieve it and, secondly, there is the risk that even if agents do agree and commit to cooperating, they may not fulfill their commitments at execution time.

How then to assess risk in interaction? Fortunately, as recognised by several researchers, this has a relatively simple solution in the form of *trust* [13, 23, 37, 65, 67]. The risk of whether to cooperate and with whom, may be determined by, among other things, the degree of confidence or *trust* in other agents. Despite the notion of *trust* being commonplace in our everyday interactions, there are few formal definitions. However, it is generally accepted that trust implies some form of risk, and that entering into a trusting relationship is choosing to take an uncertain path that can lead to either benefit or cost depending on the behaviour of others [69].

In this thesis, we view trust as the means through which an agent can approximate the risk involved in cooperation, in terms of an estimation of the degree of expectation that others will do what they agree to do, i.e. an *expectation of risk*. This is a synthetic notion of trust since, unlike Deutsch [23] and Luhmann [65], for example, we are not concerned with how trust operates in humans, but with how the concept of trust can be used in relation to cooperation between artificial agents. We are also primarily concerned with *how* an agent can use the degree of trust it has in another in reasoning about cooperation, rather than how an agent determines this degree of trust in the first place.

5.4.1 Trust

The perceived risk of cooperating with a particular agent is determined by that agent's reliability, honesty, etc., embodied by the notion of *trust*. Thus an agent can use its trust in others as a means of assessing the risk involved in cooperating with them. Describing *trust* in terms of *risk* allows us to consider the limits of trust more precisely, and to quantify it. An agent with a high trust value is more trusted than an agent with a low trust value, and represents less risk in terms of cooperation. This suggests an inverse relationship between trust and risk.

$$R = \frac{1}{T}$$

An agent's trust of another is dependent on a variety of factors, including the other's believed reliability, honesty, veracity, etc. However, modelling all such potentially relevant factors is excessive, and can add to the complexity of the solution, when typically they will not be needed. Consequently, we base our model of trust upon Marsh's formalism [67] and the work of Gambetta [37], and define the trust in an agent α , to be a value from the interval between 0 and 1: $T\alpha \in [0, 1]$. The numbers merely represent comparative values, and are not meaningful in themselves. Values approaching 0 represent complete distrust, and those approaching 1 represent complete, blind trust. In this thesis we are not concerned with how an agent should update its trust of others, but Marsh [67] describes a possible approach that will suffice, which we introduce below. This representation of trust corresponds to Marsh's notion of *general trust*. However, Marsh also introduces *situational trust*, where an agent's trust in another is dependent on the importance of the situation being considered. For example, while an agent may trust another to extract product information from a database, it might not trust it to determine which product represents the best value for money. Although conceptually situational trust is a more powerful mechanism than general trust, the computational overhead involved in identifying trust in *tasks* can be prohibitive, and so we do not use it here.

In order for this notion of trust to be useful to an agent, in its decisions about cooper-

ation, it must be able to assess what is an acceptable degree of risk. It is clear that if an agent is completely distrusted (and has a trust value approaching zero) then the risk is considered to be high, and cooperation with that agent must typically be avoided¹. However, some method is needed for determining the point at which an agent is considered sufficiently trusted to cooperate with. We address this through the introduction of a *minimum trust threshold*, such that agents trusted under that threshold are considered *distrusted*, and conversely, those of or above the threshold are trusted. If an agent is faced with the possibility of cooperating with a group of agents, then it can factor into its reasoning about cooperation whether or not the members of this group are trusted, as we discuss in the following two chapters.

5.4.2 Updating Trust of Others

At the end of a cooperative interaction, each agent involved updates its trust of the others. If the cooperative interaction was successful, and the goal achieved, then the trust an agent associates with the others involved is likely to increase. Conversely, if the goal was not achieved then the interaction was unsuccessful, and trust is likely to decrease. Since it is, in part, an agent's trust of others that determines whether or not it cooperates in a given situation, then modifying trust values after each cooperative interaction, ensures that trust can be used to assess the risk associated with cooperating with others. For example, if an agent α_1 has a cooperative intention with another, α_2 , and the interaction fails through some action on behalf of α_2 , then α_1 's trust of α_2 should decrease, reducing the likelihood of further interactions with it. The change in trust after each interaction should be relatively small, such that a single failed interaction will not prevent further interactions with an agent, but a series of repeated failures will.

Optimism and pessimism are identified by Marsh as two opposing dispositions such

¹An exception to this is if an agent's goal is sufficiently important to it, that it is better to have tried to achieve it, and failed, than to have not tried at all, even if this means cooperating with an agent of negligible trust.

that, in general, optimists trust others more than pessimists. Moreover, after a successful interaction with others, optimists increase their trust more than pessimists, and conversely, after an unsuccessful interaction, pessimists decrease their trust more than optimists [68]. Individual agents lie somewhere in the spectrum of optimism and pessimism, meaning that in a given situation different agents will change their trust by different degrees. In Marsh's view the magnitudes of alteration of trust are decided at run-time, and are dependent on a variety of factors, such as the existing trust, and cost or benefit of the situation [67, 68]. For reasons of simplicity, in our framework the magnitude of change in trust is based solely on the current trust and the agent's optimistic or pessimistic disposition. We view an agent's disposition as represented by two values, *trustIncrease* and *trustDecrease*, which determine the proportion of current trust level to increase or decrease by respectively. All that we require is that an agent has an instantiation of the following functions, which take the current trust and a value for *trustIncrease* or *trustDecrease* and return the increased or decreased trust respectively. On completing a cooperative interaction, an agent should update its trust of the other agents involved using the appropriate one of these functions.

$$\begin{array}{|l}
 \hline
 \textit{increaseTrust} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\
 \textit{decreaseTrust} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\
 \hline
 \forall \textit{trust}, \textit{trust}', \textit{trustIncrease} : \mathbb{R} \bullet \textit{increaseTrust} \textit{trust} \textit{trustIncrease} = \textit{trust}' \Rightarrow \\
 \quad \textit{trust}' > \textit{trust} \\
 \forall \textit{trust}, \textit{trust}', \textit{trustDecrease} : \mathbb{R} \bullet \textit{decreaseTrust} \textit{trust} \textit{trustDecrease} = \textit{trust}' \Rightarrow \\
 \quad \textit{trust}' < \textit{trust}
 \end{array}$$

5.4.3 Agent Models

In addition to associating a trust value with others, an agent needs knowledge about their capabilities if it is to reason about cooperation effectively. For example, if an agent is trusted, it does not mean that it is capable of performing a particular task on behalf of another. Durfee [28] notes that in order to cooperate effectively an agent needs to know certain information about others, about themselves, about how they view others and are viewed themselves, and so on. However, since an agent's reasoning is resource bounded, if taken to

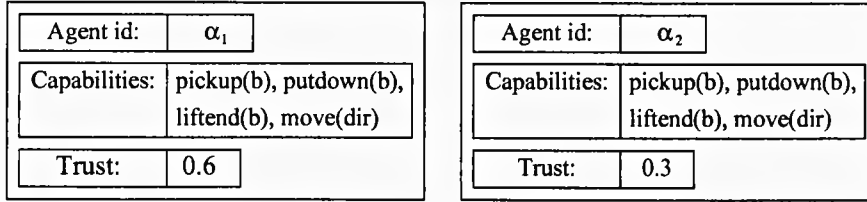


Figure 5.1: Example agent models

an extreme, the amount of knowledge an agent possesses to facilitate its cooperation might overwhelm its limited reasoning capabilities. Thus, agents need just enough knowledge to coordinate well and no more, since any additional knowledge may simply hinder the reasoning process of the agent.

In our framework we require an agent to have a *model* of each other agent with which it may interact, containing its knowledge of the other’s capabilities and the degree to which it is trusted. These agent models form part of the agent’s wider knowledge base, or beliefs. The conceptual form such models may take in an agent’s knowledge base is shown in Figure 5.1, which represents an agent’s models of two others, α_1 and α_2 . For each agent, the model contains a set of capabilities, and the degree of trust in that agent.

We formalise the notion of an agent model below, such that a model of a particular agent contains a set of contributions that it is believed capable of performing, and a trust value representing its perceived trustworthiness.

<i>AgentModel</i>	
<i>id</i> :	<i>AgentID</i>
<i>capabilities</i> :	\mathbb{P} <i>Contribution</i>
<i>trust</i> :	\mathbb{R}

In the remainder of this thesis, we frequently need to refer to the trust of a given agent. Therefore, we define the following function *trustOfAgent* which takes an agent identifier, and a set of agent models, and extracts the trust value associated with that agent.

$$\begin{array}{l}
\text{trustOfAgent} : \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \\
\hline
\forall \text{agID} : \text{AgentID}; \text{ms} : \mathbb{P} \text{AgentModel}; r : \mathbb{R} \bullet \\
\text{trustOfAgent agID ms} = r \Rightarrow r > 0 \wedge r < 1 \wedge \\
(\exists_1 m : \text{AgentModel} \mid m \in \text{ms} \bullet m.\text{id} = \text{agID} \wedge m.\text{trust} = r)
\end{array}$$

5.5 Overview of the Warehouse Domain

Now, in order to proceed with our analysis of the details of cooperation, we need a domain for illustration, implementation, and experimentation. Rather than invent a new scenario we adopt a variation on Norman’s Warehouse Domain [75], which we consider to be sufficiently complex and dynamic to be useful, while being simple enough to discuss easily. In the remainder of this thesis, we illustrate our discussions by considering agents situated in an example *warehouse domain*, which we introduce in this section. The warehouse has four areas: a delivery area, a standard storage area, a long term storage area, and a waste disposal area, such that boxes arrive in the delivery area and must then be moved to one of the storage areas (or rooms), which for simplicity are arranged linearly as shown in Figure 5.2. For an agent to move from the delivery area (room1) to the long term storage area (room3) it must, therefore, move through the standard storage area (room2). There is also a charge area (in room1), where agents can recharge their power, which decreases over time as they perform actions.

There are two types of box, urgent and non-urgent, which must be stored in the standard and long term storage areas respectively. Additionally, these boxes may be of different sizes, small and large, the former of which all agents can lift, while the latter can only be moved by particular agents, or by a group of agents though cooperation. Boxes leave the warehouse via the delivery area, and cannot be stored indefinitely — each box arriving in the warehouse is associated with some expiry time, and if it is not removed by that time then it must be moved to the waste disposal area.

The warehouse scenario is a complex domain, and so for reasons of simplicity in our

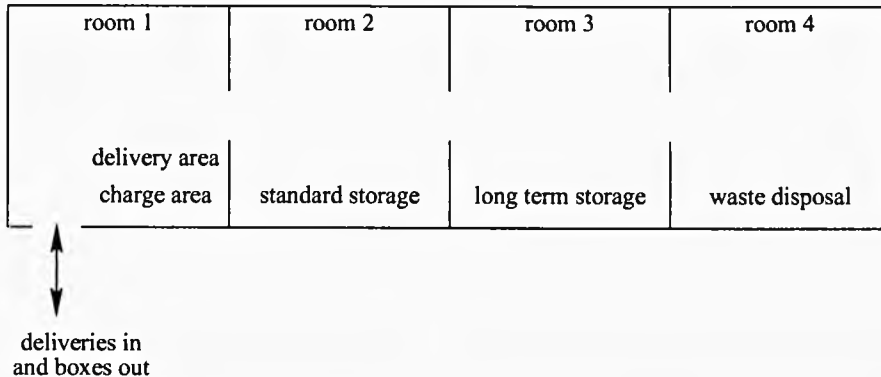


Figure 5.2: The warehouse environment

discussions of example situations, we consider a small group of four agents, each of which are given the same three motivations. The first of these, called *tidiness*, causes an agent to generate the goal of storing a particular box, and is triggered by a box being perceived in the delivery area. A *vitality* motivation generates in an agent the goal of recharging its power level, and is triggered by the power level dropping below some threshold. Finally, a *maintenance* motivation generates the goal of checking that boxes are correctly stored and have not been stored after their expiry time, and is triggered by an agent being idle.

Agents associate trust values with the others in their environment, enabling them to assess the risk of cooperating with a particular agent. These trust values are built up over time and change as a result of interactions with others, and come to represent the nature of others with a useful degree of accuracy. We provide initial values for an agent's trust of others, corresponding to those that might be arrived at through prior interactions, as given in the following matrix, where each row gives a particular agent's trust of others (for example *agent1* associates a trust value of 0.49 with *agent2*).

	agent1	agent2	agent3	agent4
agent1		0.49	0.52	0.55
agent2	0.82		0.13	0.6
agent3	0.78	0.53		0.67
agent4	0.96	0.5	0.2	

Agents are able to perform certain actions in the warehouse, in particular they are able to move around, pick up and put down boxes, and check that boxes are stored correctly. There are three types of lifting action: `pickup` which operates on small boxes, `liftend` which lifts one end of a large box, and `pickupBIG` which operates on large boxes. All agents are capable of performing these actions, with the exception of the `pickupBIG` action, which can only be performed by a specific agent, `agent3`.

For an agent to be able to achieve a goal, it requires a plan specifying the actions that are needed for its achievement. In SENARA, an agent is given a library of plans from which to select the most appropriate for a particular goal, rather than planning from first principles. Agents in the warehouse scenario must be able to move around their environment, store boxes that are delivered, check boxes are correctly stored, move boxes to the waste disposal area, and recharge their power levels when required, and we provide agents with a plan library to achieve this. Full details of these plans can be found in Appendix B.

In order to ensure the practicality of the work described in this thesis, and to demonstrate it, a SENARA testbed has been developed, based on the warehouse domain described above. The objective in constructing the testbed is to demonstrate the concepts presented in this work, and allow simple experimentation, rather than to develop a sophisticated finished product. Though we have performed a number of experiments using the testbed, and results of these support our later discussions of the framework, we do not wish to complicate the presentation of the concepts discussed in this thesis by introducing implementation level details. More details of the implementation can be found in Appendix B.

5.6 Summary

This chapter began by introducing the need for a group of cooperating agents to have an appropriate form of commitment to their interaction, which we call a cooperative intention. We presented a set of conventions that, along with appropriate commitments, form such

a cooperative intention and allow a group of agents to cooperate effectively. This was achieved by modifying the conventions defined by Wooldridge and Jennings to ensure that cooperation is *motivated* on behalf of each agent involved.

In Section 5.3 we identified the stages that are involved in motivated cooperation, and in doing so we provide the context for the following chapters. Where an agent cooperates with others, it places itself open to a certain degree of risk, and in Section 5.4 we discussed this risk, and described the notion of *trust* that can be used by an agent to assess it. Finally, in Section 5.5 we described the example scenario that we use for illustration throughout the remainder of this thesis.

Chapter 6

Plan Selection

6.1 Introduction

The first of the stages in our framework of motivated cooperation is plan selection; to achieve its goals an agent must select appropriate plans for them, and then adopt these plans as intentions. An agent selects a plan for a particular goal by determining the set of applicable plans — those plans that achieve the goal, whose preconditions are met — and then choosing the best one. Some decision mechanism is needed, therefore, for selecting a single plan from a set of applicable plans. Existing BDI architectures also require agents to select appropriate plans for their goals, and so include some means for plan selection. However, they are typically focussed on what might be called *standard* task planning and execution for *individual* agents, rather than for agents situated in a *cooperative* environment, and so do not consider the issues arising from cooperation. In particular, the questions of when to cooperate, with whom, and how, are not addressed. In this chapter we propose a method for plan selection that accounts for *why* an agent might choose to achieve such a goal through cooperation, even if it has an alternative plan it could perform alone.

In a cooperative environment, an agent's plans can contain individual, joint, and concurrent actions and as a consequence may require assistance from others for their execution.

An agent can use its beliefs about others to determine the set of agents from whom assistance may be required for a given plan and, moreover, can use its trust of them to aid its choice of plan. We begin in this chapter by describing the problem of plan selection, and examining the relevant factors that may be used in multi-agent domains, and then proceed to develop a detailed model of plan selection.

We begin this chapter by discussing the problems associated with plan selection, and the criteria that can be used to choose between plans. In Section 6.4 we describe our approach to plan selection and give details of how an agent can assess a plan in terms of its associated risk and cost. The the implications of partial plans on plan selection are discussed in Section 6.5. Finally, we conclude this chapter by giving an example from the warehouse domain, and discussing the contributions and limitations of our approach.

6.2 Cooperative Plan Selection

In the BDI model expounded by Bratman, and correspondingly in SENARA, an agent's actions are determined by its intentions. When an agent forms an intention to achieve a given goal, it does so by committing to a plan to achieve it. However, for any particular goal there may be several plans to achieve it that are *applicable* in the current situation, since their preconditions are satisfied. Some of these plans may contain actions that are beyond the agent's capabilities (or may contain joint or concurrent actions) and, if chosen, will require assistance from another agent for their execution.

Thus, an agent's choice of plan *determines* whether it must cooperate to achieve its goal. If all the applicable plans for a goal contain actions that cannot be performed by the agent alone, cooperation is *necessary*, otherwise it is *optional*. If choosing to cooperate in this latter case, there must be some inherent advantage to the cooperation, for example by minimising effort, since the goal can also be achieved by the agent alone.

To formalise this distinction we use the auxiliary function *plancontributions* (specified

in Appendix A), which retrieves all the contributions that are in the current (potentially partial) elaboration of a plan. Necessary cooperation is defined with respect to a goal and an agent in the following function, which returns true if cooperation is necessary, and false otherwise. For a given goal, if all plans in the plan library that achieve it contain one or more contributions that are beyond the agent’s capabilities then cooperation is considered necessary.

$$\begin{array}{|l}
 \hline
 \textit{nesscooperates} : \textit{Agent} \rightarrow \textit{Goal} \rightarrow \textit{bool} \\
 \hline
 \forall \textit{ag} : \textit{Agent}; \textit{g} : \textit{Goal} \mid \textit{g} \in \textit{ag.goals} \bullet \textit{nesscooperates} \textit{ ag g} = \textit{true} \\
 \Leftrightarrow (\forall \textit{p} : \textit{Plan} \mid \textit{p} \in \textit{ag.planLibrary} \bullet \textit{p.achieves} = \textit{g} \wedge \\
 \quad (\textit{plancontributions} \textit{ p} \setminus \textit{ag.capabilities}) \neq \emptyset)
 \end{array}$$

Optional cooperation is defined in a similar manner, again using a function that takes an agent and a goal as arguments, and returns true if cooperation is optional, and false otherwise. Cooperation is considered to be optional if, from the set of plans in the plan library that achieve the goal, there is at least one plan whose component contributions can be performed by the agent, and one other that contains a contribution that is beyond the agent’s capabilities.

$$\begin{array}{|l}
 \hline
 \textit{optcooperates} : \textit{Agent} \rightarrow \textit{Goal} \rightarrow \textit{bool} \\
 \hline
 \forall \textit{ag} : \textit{Agent}; \textit{g} : \textit{Goal} \mid \textit{g} \in \textit{ag.goals} \bullet \textit{optcooperates} \textit{ ag g} = \textit{true} \\
 \Leftrightarrow (\exists \textit{p}, \textit{q} : \textit{Plan} \mid \textit{p} \in \textit{ag.planLibrary} \wedge \textit{q} \in \textit{ag.planLibrary} \bullet \\
 \quad \textit{p.achieves} = \textit{g} \wedge \textit{q.achieves} = \textit{g} \wedge \\
 \quad (\textit{plancontributions} \textit{ p} \setminus \textit{ag.capabilities} \neq \emptyset) \wedge \\
 \quad (\textit{plancontributions} \textit{ q} \setminus \textit{ag.capabilities} = \emptyset))
 \end{array}$$

In existing work, several researchers have considered the situation where cooperation is necessary. However, the issues involved in determining why an agent might choose to cooperate when it is optional, have largely been unaddressed. One exception is Wooldridge and Jennings’ [102, 104] formalisation of cooperative problem-solving introduced in Section 2.7, which begins with an agent recognising the potential for cooperation. In their formalisation, the potential for cooperation is said to exist with respect to an agent’s goal if there is some group known to it that is believed can achieve the goal through cooperation,

and either

- the agent cannot achieve its goal alone, or
- it believes that for every action it could perform to achieve the goal, it has an additional goal of not performing it.

Wooldridge and Jennings recognise that this definition is overly strong, since it requires an agent to know the identity of the group it believes can achieve its goal. This rules out the situation where an agent attempts to find out the identity of a group by performing some action, and does not know their identity until after performing that action. However, in our view there is a second reason why this definition is too strong, namely that it does not allow for a situation of *optional* cooperation, where an agent has a choice between two or more ways of achieving its goal, and based on its preferences selects the one requiring cooperation, rather than achieving its goal alone.

6.3 Plan Selection Criteria

The problem of plan selection amounts to choosing the best plan — the plan that is most likely to be successful, with least cost in terms of time and resources, and the least *risk*. (While in some circumstances, such as gambling, the influence of these factors may be contradictory, requiring an agent to make a trade-off between the two, we assume that in general an agent's high-level desires are likely to be such as to attempt to minimise both the risk and the cost of its actions.) When the plans involved do not involve other agents, standard plan selection criteria, or planning heuristics, can be used to assess cost. However, when one or more of the agent's plans do involve others, an element of *risk* is introduced by the inherent uncertainty of interaction. A consequence of autonomy is that agents follow their own individual agendas, as determined by their motivations, and therefore whether a particular agent is cooperative or not is a direct function of its motivations. Thus, if an agent's motivations change during a cooperative interaction, the reason for its cooperation

may be removed, and it may drop its involvement in cooperation in favour of some other activity. In addition to a measure of the cost of a plan, therefore, we need to be able to assess the likelihood of *finding* an agent (or agents) for actions that are required for successful plan execution; the likelihood that once such agents are identified they will *agree* to cooperate; and the likelihood that once a commitment has been given, the agents concerned will *fulfill* their commitments.

We identify four primary factors relevant in comparing plans in respect of risk: knowledge of other's capabilities, risk from others, knowledge of view of self, and knowledge of other's preferences. Certainly, risk may be introduced for any number of other reasons, but these are the key domain-independent general issues.

Agent Capabilities Knowledge of others' capabilities helps to determine which agents might perform the required actions. If many agents are known to have the target capabilities, then successful execution of the plan is more likely than if fewer or no agents do so. However, in line with the motivating concerns of dynamic environments and uncertain and incomplete knowledge, we cannot assume that an agent's knowledge of others faithfully represents them, and success at execution time may be possible even if it is not anticipated at evaluation time, just as failure is also possible. In general, though, we assume that there is sufficient stability for this to be useful in assessing plans prior to execution.

Risk from Others Once potential cooperating agents are identified, they may be evaluated in terms of the risk involved in interacting with them. Plans involving agents with whom interaction is more likely to be successful, should be rated higher than those involving interactions less likely to be successful.

Risk from View of Self Knowledge of the view of oneself in the eyes of others, in terms of risk of interaction, may also be useful in assessing plans. It can provide a measure of the likelihood that another agent will agree to cooperate, since an agent is more likely to cooperate with another if it has confidence in the success of that interaction. Thus,

the agents identified in competing plans can be evaluated in respect of their view of the risk involved in cooperating with the planning agent. It is, however, difficult to maintain an assessment of how one is viewed by others.

Agent Preferences It might also be possible to assess plans in relation to the higher-level motivations of the agents involved in them, and whether cooperation would be likely. This would require a detailed model of the motivations and goals of other agents, however, which is unlikely to be accurate.

6.4 A Model of Cooperative Plan Selection

6.4.1 Plan Ratings

The problem of plan-selection is essentially the same as that of finding effective heuristics for plan construction. In that sense, we can apply standard domain-independent heuristics for evaluating plans which perform a valuable, if limited, service. These heuristics include, for example, the length of a plan as the number of its actions, the cost based on the cost of the actions it contains, and the duration of plan execution based on the duration of individual actions. We will not consider this further in the development of our framework for cooperation, since these issues are well addressed by textbooks (for example [86]), but suffice it to state that any such heuristics may be used to arrive at an assessment of a plan in terms of its *standard rating*. The heuristics used by a particular agent are embodied in its implementation of the function $sRating$, which takes a plan and returns the standard rating of that plan.

$$sRating : Plan \rightarrow \mathbb{R}$$

This evaluation of a plan does not, however, address our key concerns of assessing plans in relation to the dynamic multi-agent nature of the environment. If one or more of the plans available to an agent requires interaction with another the *standard rating* is inad-

equate, because this interaction introduces an element of risk. A second rating is therefore necessary in these terms, which we call the *cooperative rating*.

6.4.2 Assessing Contributions

In assessing the merit of a plan (i.e. determining the cooperative rating), an agent must make a judgement about the risk attached to each action in the plan requiring cooperation, by examining the trust value in its model of each of the possible cooperating agents. Before describing how to assess the risk involved in selecting a particular plan, we describe how an agent can assess the risk attached to actions, starting with an individual action, or *contribution*. Suppose that an agent knows of n others, $\alpha_1, \alpha_2, \dots, \alpha_n$, with the required capabilities for performing a given contribution, and ordered such that $T\alpha_{x-1} \geq T\alpha_x$, where $T\alpha_x$ denotes the trust in α_x . Several possibilities for assessing the risk involved in cooperating with others are discussed below.

We might only consider trust in the *most trusted* agent involved so that the risk of a particular contribution would be as follows.

$$riskC = \frac{1}{T\alpha_1}$$

Though simple, the problem with this approach is that the most trusted agent might not be the actual agent involved in the cooperative action, for any number of reasons. In particular, the autonomous nature of agents underlying this framework suggests that it is impossible to determine the behaviour of another agent in advance. As a consequence, cooperation with less trusted others may be needed, and this must be factored into the measure of risk. Alternatively, then, we might consider a second method in which the additive total of trust in all agents in the set of potential agents for the action, is considered.

$$riskC = \frac{1}{\sum_{i=1}^n T\alpha_i}$$

This avoids the problem of only considering the most trusted agent, and considers all agents to an equal extent, but does not address the decreased likelihood of cooperation with

less trusted agents. An agent will first try to cooperate with α_1 and, if unsuccessful, will then try α_2 , and so on, but for each successive agent, the likelihood of actually cooperating with it decreases (since it will only be asked if all preceding agents have declined). To address this, we can adjust the formula to increase the significance of more trusted agents, by dividing the trust of successive agents by a correspondingly increasing factor.

$$riskC = \frac{1}{\sum_{i=1}^n \frac{T\alpha_i}{i}}$$

To specify this final method of assessing the risk associated with a contribution we use the functions *orderedCapableAgents*, *orderedTrust*, *scaleTrustSeq*, and *sumSeq* (defined in Appendix A). The former of these takes a contribution and a set of agent models and returns a sequence of agents capable of performing that contribution, ordered according to their trust. This sequence of agents forms the argument to *orderedTrust*, which returns the corresponding sequence of trust values. The function *scaleTrustSeq* takes such a list of values, and scales them according to their position in the sequence (i.e. the i 'th value is divided by i), while the remaining function sums the resulting list of values. Combining these auxiliary functions as follows, in *riskC*, we specify how to determine the risk associated with a contribution, namely, by obtaining an ordered sequence of values corresponding to the trust of the agents capable of performing it, and scaling and summing this sequence appropriately.

$$\left| \begin{array}{l} riskC : Contribution \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R} \\ \hline \forall c : Contribution; ms : \mathbb{P} AgentModel; r : \mathbb{R} \bullet riskC \ c \ ms = r \\ \Leftrightarrow 1 / sumSeq (scaleTrustSeq (orderedTrust (\\ \quad orderedCapableAgents \ c \ ms) \ ms)) = r \end{array} \right.$$

Thus, trust in all relevant agents is considered, but only in relation to the likelihood of cooperation with them. Consider an action for which there are three agents that have the required capabilities α_1 , α_2 , and α_3 , which have associated trust values of 0.3, 0.9, and 0.5 respectively. Thus, using the above method of assessing risk, these agents are ordered

α_2 , α_3 and α_1 according to the degree they are trusted. The risk associated with action, according to the above equation, is $1/((0.9/1) + (0.5/2) + (0.3/3)) = 0.8$.

6.4.3 Assessing Joint Actions

We can extend this strategy for assessing the risk involved in a particular contribution to apply to joint and concurrent actions. For joint actions, we simply replace the agents and trust of agents in the equation with sets of agents that are capable of performing the action, and the trust of these sets of agents respectively. A set of agents is capable of performing a joint action if for each contribution there is exactly one agent capable of performing it, no agent in the set is required to perform more than one contribution at a given time, and each agent is required to perform the action. In other words, suppose that for a given action there are several sets of agents that are capable of performing it, then, in calculating the risk we would obtain the trust value for each set, and use these in the equation given above for assessing the risk associated with a contribution. The trust value of a set is determined by multiplying the trust values of the member agents. Thus, for a set containing n agents, $\alpha_1, \alpha_2, \dots, \alpha_n$, the trust of that set is given by the following equation.

$$trustSet = \prod_{i=1}^n T\alpha_i$$

In specifying how to assess the risk associated with a joint action, we make use of the auxiliary functions *orderedCapableAgentSets* and *orderedTrustSet*, which we define in Appendix A. The former of these takes a joint action and a set of agent models, and returns a sequence of sets of agent identifiers (ordered by trust), such that each set has the required capabilities to perform the joint action, while the latter takes this sequence and obtains a sequence of values, where each value represents the combined trust in the set of agents in the corresponding position in the sequence of capable agent sets. We specify how to assess the risk associated with a joint action by using these functions to find the set of sets of agents that are capable of performing the joint action, ordering these sets according to the trust of

the agents contained in them, and finally dividing successive trust values by an increasing factor.

$$\begin{array}{|l}
 \hline
 \textit{riskJA} : \mathbb{P} \textit{Contribution} \rightarrow \mathbb{P} \textit{AgentModel} \rightarrow \mathbb{R} \\
 \hline
 \forall cs : \mathbb{P} \textit{Contribution}; ms : \mathbb{P} \textit{AgentModel}; r : \mathbb{R} \bullet \textit{riskJA} \ cs \ ms = r \\
 \Leftrightarrow 1 / \textit{sumSeq} (\textit{scaleTrustSeq} (\textit{orderedTrustSet} \\
 \textit{orderedCapableAgentSets} \ cs \ ms) \ ms) = r
 \end{array}$$

6.4.4 Assessing Concurrent Actions

The approach for concurrent actions is an extension of that for joint actions since, instead of a set of contributions, a concurrent action comprises a set of sequences of steps, each to be performed concurrently with the others. A set of agents capable of performing each of the concurrent sequences of steps can be determined by combining the sets of agents capable of performing the actions in each sequence. The set of agents capable of performing the concurrent action can then be determined, using the assumption that an agent cannot perform more than one action simultaneously, and cannot typically not appear in more than one sequence of steps. Thus, in assessing a concurrent action, each sequence of steps is analogous to an individual contribution in a joint action.

Since a concurrent action contains a set of individual contributions or joint actions, we can use the above approach to assess the risk in these components, as specified below in *riskCAcomponent*. We can then multiply these values for each component of a given concurrent action to obtain an estimate of the risk associated with the concurrent action itself.

$$\begin{array}{|l}
 \hline
 \textit{riskCAcomponent} : \textit{CAcomponent} \rightarrow \mathbb{P} \textit{AgentModel} \rightarrow \mathbb{R} \\
 \hline
 \forall \textit{comp} : \textit{CAcomponent}; ms : \mathbb{P} \textit{AgentModel}; r : \mathbb{R} \bullet \\
 \textit{riskCAcomponent} \ \textit{comp} \ ms = r \Leftrightarrow (\exists c : \textit{Contribution} \bullet \\
 \textit{comp} = \textit{Contrib} \ c \wedge r = \textit{riskC} \ c \ ms) \vee \\
 (\exists cs : \mathbb{P} \textit{Contribution} \bullet \textit{comp} = \textit{JA} \ cs \wedge r = \textit{riskJA} \ cs \ ms)
 \end{array}$$

$$\text{riskCA} : \mathbb{P} \text{CAcomponent} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R}$$

$$\begin{aligned} \forall \text{comps} : \mathbb{P} \text{CAcomponent}; \text{ms} : \mathbb{P} \text{AgentModel}; \bullet \\ \text{riskCA comps ms} = \text{productSet}\{\text{ca} : \text{CAcomponent}; \text{r} : \mathbb{R} \mid \\ \text{ca} \in \text{comps} \wedge \text{r} = \text{riskCAcomponent ca ms} \bullet \text{r}\} \end{aligned}$$

6.4.5 Cooperative Rating of a Plan

Using these measures of risk of actions, we can determine the *cooperative rating* of a plan by summing the risk associated with each step in it. This risk is additive because each step offers a new and independent possibility of risk. Thus, a plan with few high risk steps may be rated better (or less risky) than a plan with many low risk steps. For a plan with m steps, ps_1, ps_2, \dots, ps_m , the cooperative rating, C , for that plan is given by the following equation, where Rps is the risk associated with the step ps .

$$C = \sum_{i=1}^m Rps_i$$

The risk for a given plan step can be determined by simply applying the appropriate risk assessment function, according to whether the step represents an individual contribution, a joint action, or a concurrent action. Based upon this, the risk for a plan is simply the sum of the risk associated with the steps in it. We specify this below, where *riskPlanStep* applies the corresponding assessment function to a plan step, and *cRating* takes a plan and applies *riskPlanStep* to each of the steps in its body, summing the results.

$$\text{riskPlanStep} : \text{PlanStep} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R}$$

$$\begin{aligned} \forall \text{ps} : \text{PlanStep}; \text{ms} : \mathbb{P} \text{AgentModel}; \text{r} : \mathbb{R} \bullet \text{riskPlanStep ps ms} = \text{r} \Leftrightarrow \\ (\exists \text{c} : \text{Contribution} \bullet \text{ps} = \text{Individual c} \wedge \text{r} = \text{riskC c ms}) \vee \\ (\exists \text{cs} : \mathbb{P} \text{Contribution} \bullet \text{ps} = \text{Joint cs} \wedge \text{r} = \text{riskJA cs ms}) \vee \\ (\exists \text{cacs} : \mathbb{P} \text{CAcomponent} \bullet \text{ps} = \text{Concurrent cacs} \wedge \text{r} = \\ \text{riskCA cacs ms}) \end{aligned}$$

$$\text{cRating} : \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R}$$

$$\begin{aligned} \forall \text{p} : \text{Plan}; \text{ps} : \text{seq PlanStep}; \text{ms} : \mathbb{P} \text{AgentModel}; \text{r} : \mathbb{R} \bullet \\ \text{cRating p ms} = \text{r} \Leftrightarrow \text{ps} = \text{p.body} \wedge (\exists \text{ratings} : \mathbb{P} \mathbb{R} \bullet \text{ratings} = \\ \{\text{r}' : \mathbb{R} \mid (\exists \text{s} : \text{PlanStep} \mid \text{s} \in \text{ran p.body} \bullet \text{r}' = \\ \text{riskPlanStep s ms}) \bullet \text{r}'\} \wedge \text{r} = \text{sumSet ratings}) \end{aligned}$$

6.4.6 Plan Quality

Once both the *standard* and *cooperative* ratings of a plan have been determined, they must be combined to form an overall measure of plan quality to select between alternative applicable plans. It would not be sensible simply to add the two values together, since one measures the *cost* of the plan, and the other the *risk* involved in it, and the relative importance of these may vary for each agent. We therefore include a weighting for these ratings for a particular agent in the overall quality measure, Q , as follows, where w_s and w_c represent the influence weighting applied to the *standard rating*, S , and *cooperative rating*, C , respectively.

$$Q = (w_s * S) + (w_c * C)$$

This is specified in the following function, which takes a plan, a set of agent models, and weightings for the standard and cooperative ratings, and returns the overall quality of the plan, as defined by the above equation.

$$\left| \begin{array}{l} \text{quality} : \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\ \hline \forall p : \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c, r : \mathbb{R} \bullet \\ \text{quality } p \text{ } ms \text{ } w_s \text{ } w_c = r \Leftrightarrow r = (s\text{Rating } p) * w_s + (c\text{Rating } p \text{ } ms) * w_c \end{array} \right.$$

Different agents may use different weightings, the values used reflecting, in part, an agent's predisposition, since agents that place greater importance on the *standard rating* are inclined to minimise the cost of achieving their goals, whether or not this requires cooperation. Conversely, agents that place most importance on the *cooperative rating* are predisposed to minimising the risk involved in cooperating with others, even if this increases the cost involved in achieving their goals. Thus, agents that place more importance on the standard rating are more inclined to take risks associated with cooperation in order to minimise the cost of their plans, when compared to agents that place more importance on the cooperative rating. The values of the weighting that provide the best selection of plans depends on an agent's environment.

6.5 Pre-Execution Plan Assessment

A result of the partial nature of an agent's plans is that the set of applicable plans for a goal are also partial, and may require further elaboration before they can be fully executed. Moreover, if a plan contains a subgoal and requires elaboration, then the set of actions that it will eventually contain are not known until it is actually elaborated (since it is not known which subplans will be chosen). Therefore, the cooperative rating of a partial plan cannot be directly assessed as described above, since it is not known what contributions will be required to achieve any subgoals it contains. A naive solution to assessing a partial plan would thus be to require an agent to fully elaborate each of its applicable plans in order to choose between them. While this would indeed allow direct use of the criteria described above, it also requires a premature commitment to a particular plan. Such a requirement would negate the benefit of using partial plans in the possibility of interleaving execution and deliberation to cope with the environmental change that is typical of multi-agent scenarios. More importantly, it demands a search through the entire tree of plans so that the quality of each possible path solution can be measured. This is prohibitively expensive to be performed in real-time.

If we are to avoid constructing the entire search tree at the time of plan selection, we must be able to make a choice based on a limited number of alternatives, such as the top-level applicable plans. An informed choice at this level is only possible, however, if we have some measure of the value of plans in terms of the standard and cooperative ratings, but clearly, this is not possible to do on the fly. Instead, we perform an off-line *pre-execution assessment* of the plan library in which all of the plans in it are evaluated in a coarse fashion with respect to the agents required for successful execution. This approach represents a compromise between the desire to minimise the computational overhead and that of maximising the quality of any measure of the value of a plan.

Starting with the plans that require no further elaboration, since these are the only ones which can be directly evaluated, the *standard* and *cooperative* ratings are determined. These

ratings must then be fed back into the other plans as values for subgoals within them. For each plan containing actions that cannot be performed by the planning agent, the set of all agents known to have the relevant capability is generated through inspection of its agent models, so that these ratings can be calculated as described earlier.

There are two possible approaches to incorporating these values for fully elaborated plans into the larger partial plans of which they might form subplans. Firstly, these values can be used in subsequent levels in the library for which the plans *best* satisfy subgoals, and so on until each plan has an overall quality measure. This quality measure is an assessment of the *best-case* solution. An alternative approach is to take into account *all* possible elaborations and calculate a *mean* rating for competing plans, so that there is less reliance on one individual plan that may not be applicable at execution time. This provides a less sensitive measure, but one which is more likely to be useful in a dynamic environment, since it does not rely on a single plan elaboration, but rather on all possible elaborations of the agent's plans. In the development of a framework for cooperation, we therefore require that an agent has an appropriate implementation of the functions for obtaining the best-case and mean-case ratings of plans.

For the purposes of specification we introduce a number of auxiliary functions: *plansubgoals*, *possibleSubplans*, *possibleSubplansRatings*, *minRating*, and *meanRating* (specified in Appendix A). The first of these, simply returns the subgoals that are contained within a plan. Each subgoal in a plan may typically be elaborated by a number of other plans, and the function *possibleSubplans* retrieves the possible subplans for a given subgoal in a particular plan, and checks that these plans are not recursive (for reasons described later in this section). The function *possibleSubplansRatings* takes a set of possible subplans, and returns a set of values corresponding to the plan ratings of those subplans. Finally, the functions *minRating* and *meanRating* take this set of ratings and return the minimum and mean values respectively.

Using these auxiliary functions we can specify how to obtain the best-case and mean

ratings of a plan. In both cases, if there are no subplans, then the rating is simply equal to the rating afforded by the plan steps contained in the plan. However, if the plan does contain subgoals then the best-case rating, $bcRating$, is obtained by adding the *best* rating (i.e. the one with the least numerical value) for each subgoal to the rating given by the plan steps. The mean rating, $mcRating$ is defined similarly (and specified in Appendix A).

$$\begin{array}{l}
 \hline
 bcRrating : Plan \rightarrow \mathbb{P} Plan \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\
 \hline
 \forall p : Plan; pLib : \mathbb{P} Plan; ms : \mathbb{P} AgentModel; w_s, w_c, r : \mathbb{R} \bullet \\
 \quad plansubgoals p = \emptyset \Leftrightarrow bcRating p pLib ms w_s w_c = \\
 \quad \quad Quality p ms w_s w_c \\
 \forall p : Plan; pLib : \mathbb{P} Plan; ms : \mathbb{P} AgentModel; w_s, w_c, r : \mathbb{R} \bullet \\
 \quad plansubgoals p \neq \emptyset \Leftrightarrow bcRating p pLib ms w_s w_c = \\
 \quad \quad Quality p ms w_s w_c + sumSet \{r : \mathbb{R}; g : Goal \mid \\
 \quad \quad \quad g \in plansubgoals p \wedge r = minRating (\\
 \quad \quad \quad \quad possibleSubplansRatings(possibleSubplans g p pLib) \\
 \quad \quad \quad \quad ms w_s w_c) \bullet r\}
 \end{array}$$

6.5.1 Best-case and Mean-case Advantage

The balance between the *best-case* and *mean* ratings amounts to a trade-off between an agent trying to find the best final plan and minimising the chance of the final plan being poor due to environmental change (in terms of these ratings). These best-case and mean ratings for agent plans will need periodic reassessment as the agent's knowledge of other's capabilities (and its trust in them) changes.

The *best-case advantage* (BCA) of one plan over other applicable plans is the advantage of that plan over others if its final elaboration has the best quality rating. Thus, for two applicable plans, p and q , with best-case ratings of $Q_b(p)$ and $Q_b(q)$ respectively the BCA is equal to the difference between the quality rating for p and that for q , as follows.

$$| Q_b(p) - Q_b(q) |$$

If there are more than two applicable plans, as is typical, then the BCA is equal to the difference between the minimum and maximum best-case ratings. Thus with applicable plans p, q, \dots, z the BCA is determined by the following equation.

$$bca = \max\{Q_b(p), Q_b(q), \dots, Q_b(z)\} - \min\{Q_b(p), Q_b(q), \dots, Q_b(z)\}$$

To specify this we use the functions *maxRating* and *minRating* (defined in Appendix A), which take a set of ratings and return the least and greatest values respectively. From the maximum and minimum ratings the best-case advantage can be determined as defined above.

$$\left| \begin{array}{l} bca : \mathbb{P} Plan \rightarrow \mathbb{P} Plan \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\ \hline \forall ps, pLib : \mathbb{P} Plan; ms : \mathbb{P} AgentModel; rs : \mathbb{P} \mathbb{R}; w_s, w_c, r : \mathbb{R} \bullet \\ \quad bca \ ps \ pLib \ ms \ w_s \ w_c = r \\ \quad \Leftrightarrow rs = \{r' : \mathbb{R} \mid (\exists p : Plan \mid p \in ps \bullet r' = \\ \quad \quad bcRating \ p \ pLib \ ms \ w_s \ w_c)\} \wedge \\ \quad \quad r = maxRating \ rs - minRating \ rs \end{array} \right.$$

The *mean-case advantage* (MCA) of one plan over other applicable plans is the typical (or mean) extra advantage. This is a general case measure that incorporates more information, since it takes into account all possible elaborations of the applicable plans. With mean ratings for p and q of $Q_m(p)$ and $Q_m(q)$, the MCA is equal to $|Q_m(p) - Q_m(q)|$. As above, if there are more than two applicable plans, the MCA is equal to the difference between the minimum and maximum mean ratings. Thus with plans p, q, \dots, z the MCA is as follows, and can be specified similarly to the BCA (as defined in Appendix A).

$$mca = \max\{Q_m(p), Q_m(q), \dots, Q_m(z)\} - \min\{Q_m(p), Q_m(q), \dots, Q_m(z)\}$$

6.5.2 Recursion

A problem arises with this approach when a plan is recursive, or a set of plans are mutually recursive, since it is not possible to obtain a rating for a subplan to feed into a higher level plan with respect to which it is recursive. The only solution to this is to use domain specific knowledge about the typical application of the plans, to estimate the limit of the recursion. Despite the recursion, a partial rating can be obtained for the plan in terms of the cost and risk of the actions it contains. We can then use the domain knowledge to estimate the limit

of the recursion, i.e. the number of times the plan will be executed, and we call this a *recursive multiplier* for the plan. The partial rating of the plan is multiplied by this value to obtain an estimated rating, since the plan is estimated to be used that number of times. Thus, if a particular recursive plan is on average called five times before the recursion terminates, then we would multiply the rating obtained from the actions in the plan and the non-recursive subgoals by five to estimate a cooperative rating for that plan.

To formalise this we use the function *existsRecursiveElaboration* (specified in Appendix A) which takes a plan and a library of plans and returns true if the plan can be elaborated recursively (i.e. using itself), and false otherwise. We use this to specify the function *scaleForRecursion*, which multiplies the rating of a plan by a recursive multiplier if a recursive elaboration exists for that plan.

$$\begin{array}{l}
 \hline
 \text{scaleForRecursion} : \text{Plan} \rightarrow \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\
 \hline
 \forall p : \text{Plan}; pLib : \mathbb{P} \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c, r, m : \mathbb{R} \bullet \\
 \quad \text{existsRecursiveElaboration } p \text{ } pLib = \text{false} \\
 \quad \Leftrightarrow \text{scaleForRecursion } p \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \text{ } m = \\
 \quad \quad \text{BCrating } p \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \wedge \\
 \quad \text{existsRecursiveElaboration } p \text{ } pLib = \text{true} \\
 \quad \Leftrightarrow \text{scaleForRecursion } p \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \text{ } m = \\
 \quad \quad \text{BCrating } p \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c * m
 \end{array}$$

6.5.3 Selecting Between Partial Plans

There is a trade-off between maximising the best-case and mean-case advantage. If the best-case advantage of a plan, p , over another, q , outweighs the mean-case advantage of q over p , then p should be selected; similarly if the mean-case advantage of q over p is greater than the best-case advantage of p over q , then q should be selected.

More generally, the advantage should be maximised, regardless of whether it is best-case or mean-case. If $BCA > MCA$ then the best-case rating should be used to select plan x , such that $Q_b(x) < Q_b(p) \wedge Q_b(x) < Q_b(q) \wedge \dots \wedge Q_b(x) < Q_b(z)$. Alternatively, if $MCA > BCA$ then the mean-case rating of the applicable plans should be used.

The following function, *useBCA*, formalises this and specifies the conditions under which the best-case ratings should be used to select a plan (i.e. whenever they offer the greater advantage). The conditions under which the mean-case ratings should be used can be defined similarly (as specified in Appendix A).

$$\begin{array}{l}
\hline
\text{useBCA} : \mathbb{P} \text{ Plan} \rightarrow \mathbb{P} \text{ Plan} \rightarrow \mathbb{P} \text{ AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{bool} \\
\hline
\forall ps, pLib : \mathbb{P} \text{ Plan}; ms : \mathbb{P} \text{ AgentModel}; w_s, w_c : \mathbb{R} \bullet \\
\text{useBCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{true} \\
\Leftrightarrow \text{BCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c > \text{MCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \wedge \\
\text{useBCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{false} \\
\Leftrightarrow \text{BCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c < \text{MCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c
\end{array}$$

Once the mean-case and best-case advantages have been considered a plan can be chosen using that criterion. We define a functions *selectByBCA* which selects the plan with the best (i.e. least numerical) rating using the best-case rating. An analogous function, *selectByMCA*, can be defined similarly (as specified in Appendix A). Using these functions, we also define *selectBestPlan*, that takes a set of applicable plans and returns the best one, using either the best-case or mean-case rating as appropriate.

$$\begin{array}{l}
\hline
\text{selectByBCA} : \mathbb{P} \text{ Plan} \rightarrow \mathbb{P} \text{ Plan} \rightarrow \mathbb{P} \text{ AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{Plan} \\
\hline
\forall ps, pLib : \mathbb{P} \text{ Plan}; ms : \mathbb{P} \text{ AgentModel}; w_s, w_c : \mathbb{R}; chosen : \text{Plan} \bullet \\
\text{selectByBCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{chosen} \\
\Leftrightarrow (\forall p' : \text{Plan} \mid p' \in ps \bullet \text{BCrating } chosen \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c > \\
\text{BCrating } p' \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c)
\end{array}$$

$$\begin{array}{l}
\hline
\text{selectBestPlan} : \mathbb{P} \text{ Plan} \rightarrow \mathbb{P} \text{ Plan} \rightarrow \mathbb{P} \text{ AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{Plan} \\
\hline
\forall ps, pLib : \mathbb{P} \text{ Plan}; ms : \mathbb{P} \text{ AgentModel}; w_s, w_c : \mathbb{R}; chosen : \text{Plan} \bullet \\
\text{selectBestPlan } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{chosen} \Leftrightarrow \\
(\text{useBCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{true} \wedge \\
\text{chosen} = \text{selectByBCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c) \vee \\
(\text{useMCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{true} \wedge \\
\text{chosen} = \text{selectByMCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c)
\end{array}$$

The plan selection mechanism described in this chapter provides an instantiation of the function *planForGoal* introduced in Section 4.8.1. To select the best plan for a given goal,

the agent must first determine the set of applicable plans, using the function *planSetForGoal* (again introduced in Section 4.8.1). A plan can then be selected from this set using the function *selectBestPlan* described above.

$$\begin{array}{|l}
 \hline
 \textit{planForGoal} : \mathbb{P} \textit{Belief} \rightarrow \mathbb{P} \textit{Intention} \rightarrow \mathbb{P} \textit{Plan} \rightarrow \textit{Goal} \rightarrow \textit{Plan} \\
 \hline
 \forall \textit{bel} : \mathbb{P} \textit{Belief}; I : \mathbb{P} \textit{Intention}; \textit{plib} : \mathbb{P} \textit{Plan}; g : \textit{Goal}; w_s, w_c : \mathbb{R}; p : \textit{Plan} \\
 \bullet \textit{planForGoal} \textit{bel} I \textit{plib} g = p \\
 \Leftrightarrow \textit{selectBestPlan} (\textit{planSetForGoal} g \textit{bel} \textit{plib}) \textit{plib} \\
 (\textit{extractAllModels} \textit{bel}) w_s w_c = p
 \end{array}$$

6.6 Warehouse Example

To illustrate this scheme, we use the example of the warehouse domain, introduced in Section 5.5, and we consider plan selection from the point of view of a individual agent, *agent1*. The sets of plans in this domain are those defined in Tables B.1 and B.2, which can be assessed as described above. Assessment begins with *stayPutPlan* since it contains no subgoals. However, it also contains no actions and so it is of zero risk.

The plan for staying in the current location may be used in the elaboration of the plans for moving right and left, and so its rating is used in determining their ratings. To obtain the rating for *moveRightPlan*, each of the steps in its body is considered in turn. Firstly, the rating for the step corresponding to the action of moving right is obtained by considering its standard cost and the trust of the agents that might perform it. Secondly, the subgoal of being in a particular location is assessed by considering the ratings of its possible elaborations. Two possible plans (apart from the plan for moving right itself) might be used in its elaboration: *stayPutPlan* and *moveLeftPlan*. The rating for the former of these can be incorporated, but the latter cannot since *moveLeftPlan* and *moveRightPlan* are mutually recursive meaning each might be a subplan of the other and therefore their ratings must be scaled for recursion. Thus *moveRightPlan* is assessed as follows.

```

assessing moveRightPlan
  current step: <action (move, [_agent, right])=_agent>

```

```

step is action, assessing... value is 1.03
current step: <goal [$(location, [_agent, _y])$]>
step is goal, finding BC and MC rating
  poss elaborations: [moveLeftPlan, stayPutPlan]
  considering subplan moveLeftPlan:
    plans are mutually recursive
  considering subplan stayPutPlan
    BC value is 0.0 MC value is 0.0
  scaling for recursion... factor is 3.0
plan's cooperative rating assessed as
  3.08 (BC) 3.08 (MC)
plan's quality assessed as
  6.08 (BC) 6.08 (MC)

```

We give full details of this pre-execution assessment in Section B.5.2 where we describe our demonstration of the model, and here we simply give the results of this assessment, which are as follows.

plan	best-case rating	mean-case rating
stayPutPlan	0	0
moveRightPlan	6.08	6.08
moveLeftPlan	6.08	6.08
storeSmallPlan	10.05	18.16
storeLargePlan	23.82	23.82
storeLargePlanCheap	13.85	21.96
checkPlan	6.08	6.08
rechargePlan	7.0	11.05

Now, since the trust placed in agents changes over time, these ratings may become out of step with the current situation, and so they must be reassessed periodically. For example, if agent1's trust of agent3 changes from 0.52 to 0.1, i.e. from a high to a low level of trust, then the ratings of risk associated with plans that might involve agent3 will increase when reassessed, to reflect the increased risk. For example, the actions in storeLargePlan must be performed by agent3 because it is the only agent capable of moving such a box. Therefore, if the plan library is reassessed the ratings of this plan will increase, giving best-case and mean ratings of 30.0 and 38.83 respectively.

Selecting a Plan

The ratings determined by an agent's pre-execution assessment of its plan library are used to select the best plan to achieve a goal. In the case where there is only one applicable plan, which can be performed individually, then plan selection is trivial — the applicable plan is selected. For example, if a small box is delivered to the warehouse, and `agent1` perceives this, it will form the goal of storing the box as a result of its `tidiness` motivation. In order to form an intention it must select a plan for this goal, and the only applicable plan is `storeSmallPlan`. Thus, the agent selects this plan, and forms an intention towards its execution.

However, where there are a number of applicable plans, which require cooperation to perform, then plan selection is more complex. For example, consider the situation where `agent1` perceives that a large box, `box1`, has arrived in the delivery area, and adopts the goal of moving it to the storage area. There are two applicable plans in this situation, `storeLargePlan` and `storeLargePlanCheap`; the former uses joint actions of two agents lifting the box and moving it, while the latter must be executed by an individual agent with the ability to lift a large box. Any of the other agents can assist for the execution of the former plan (since all agents have the required capabilities), but only `agent3` can assist for the latter, since it is the only agent able to perform the action of lifting a large box individually. The best plan should be selected based on the best-case and mean-case advantage as discussed in Section 6.5. Using the ratings calculated above, the best-case advantage of choosing `storeLargePlan` over `storeLargePlanCheap` is 9.97, and the mean-case advantage is 1.86. Thus, the best-case advantage is greater and so the plan with the lowest best-case rating should be selected, namely `storeLargePlanCheap`¹. The agent can then begin the procedures required to adopt this plan as an intention.

Alternatively, if the agent had a different level of trust in the others, then a different

¹In this case, `storeLargePlanCheap` would also have been chosen using the mean-case rating, but this is not always the case.

plan might have been chosen. For example, if `agent3` is little trusted and associated with a trust value of 0.1, instead of being trusted, then the rating for the `storeLargePlanCheap` changes as given above. The ratings for the other plans also change; in particular, the best-case and mean-case ratings for `storeLargePlan` both become 33.53. Here, the risk associated with `storeLargePlanCheap` is significantly increased, and the best-case and mean-case advantages become 3.53 and 5.3 respectively and therefore the mean-case rating should be used to select the best plan. The plan `storeLargePlan` has the lowest mean-case rating, and so is selected.

6.7 Summary

Plan selection is fundamental to our framework of cooperation, since an agent's chosen plan determines whether it must cooperate to achieve a particular goal or not. In this chapter we have presented a mechanism for plan selection, through which an agent can decide whether to cooperate even when it is *optional*, unlike other models which tend to concentrate on *necessary* cooperation.

Plan selection in other approaches, even in a cooperative domain, typically only considers the *cost* of a plan and not the *risk* associated with it. In practice an agent is often faced with a choice of two or more applicable plans for a given goal, where cooperation is required to execute those plans. Assistance is typically required for different actions, and so plans involve different agents, which in turn pose varying degrees of risk, as reflected in their trust values. Similarly, plans typically have different costs associated with them, according to the actions they contain. Using existing plan selection mechanisms of only considering the cost of a plan, an agent will always select the plan with the least cost, regardless of the risk from cooperation. However, using our approach this risk is factored into the plan selection process, and provided an agent's trust of others is broadly accurate² then

²Trust is not guaranteed to be accurate for two main reasons. Firstly, it is based on observation and is only intended to be an estimate. Secondly, it evolves over time and takes several interactions to reflect another's

the agent will avoid choosing a plan that requires cooperation with distrusted agents.

The main limitation to our method of plan selection arises from the reliance on the pre-execution assessment of the plan library (with periodic reassessment). In particular, because an agent's assessment will generally not be completely up to date with the changes in its trust of others, an agent may choose a plan that it perhaps would not have done, were its plan ratings reassessed. The implication of this is that if an agent goes from being reliable to unreliable (i.e. trusted to distrusted) then the agent will still rate plans involving that agent as low risk, and may select them, even though they are actually now of *high* risk.

An additional limitation arises from the cost of assessing the plan library, since although we reduce the computational cost of assessment by performing it off-line (or when the agent is idle in the case of re-assessment), there is still a significant cost involved, proportional to the number of plans in the library, the actions in them, and the number of agents with whom cooperation may occur. While this suggests a theoretical bound on the applicability of the method there are no practical problems with moderate numbers of plans and agents³.

Certainly, more sophisticated mechanisms involving the likelihood of particular elaborations of individual plans are possible, but these require much more extensive knowledge of the relationship of plans and environments, and the nature of change in environments, as well as significantly more costly computation. Given that the environment is largely unpredictable, there is unlikely to be any significant advantage, however.

This approach is suited to situations in which the likelihood of the environment and the agent models remaining the same is high, so that plan elaboration at execution time is likely to reflect the plan quality values determined in advance for the overall partial plans concerned. Reassessment of these quality measures will be required periodically to ensure they are consistent with the changes in trust of others. Although we do not address this issue in this chapter, a simple strategy is for an agent to perform this reassessment when it is not otherwise, thus if an agent has had few previous interactions with another then its trust of that agent may not give an accurate representation.

³We have run example scenarios with 30 plans and 30 agents.

erwise occupied, or when the change in its trust of others exceeds some threshold. Despite there being some significant computation involved, it is limited in the number of capable agents, the number of plans, and the number of actions in those plans. Moreover, since assessment is carried out in a *pre-execution* strategy combined with periodic reassessment, the overhead placed on an agent for plan selection at run time is relatively low, especially if computation relating to plan reassessment is performed when the agent is idle.

Chapter 7

Cooperative Intention Formation

7.1 Introduction

Once an appropriate plan has been selected for its goal, an agent must adopt it as an intention. The manner in which an intention is adopted is dependent on whether the plan requires cooperation. If the plan contains a joint or concurrent action, an action the agent cannot perform, or an action it would simply prefer to be performed by another agent, then the plan requires cooperation. Where the chosen plan is not cooperative then no further processing is required and the agent can adopt it as an intention using the mechanism described in Chapter 4. In the case where an agent's chosen plan is cooperative (as determined from its current elaboration) more processing is required before it can be fully executed; in particular, before an agent can act with others it must obtain some form of commitment from them, and it is the formation of this commitment that we discuss in this chapter.

We begin this chapter by giving an overview of the process of forming a cooperative intention. Sections 7.3 and 7.4 describe how an agent can annotate its plan with the identifiers of those with whom it wishes to cooperate. After plan annotation an agent attempts to gain the assistance of others, by requesting their cooperation as we describe in Sections 7.6 and 7.7. Those agents whose assistance is requested decide whether to accede to the

request based on their motivations and knowledge of the other agents involved and, if accepting, form a commitment to cooperation, as discussed in Sections 7.8 and 7.9. We then in Section 7.10 consider the formation of a full cooperative intention amongst a group of willing agents. In Section 7.11 we give an example of cooperative intention formation in the warehouse domain. Finally, we consider the two main strategies to forming a cooperative intention, in terms of the point at which formation occurs (namely, plan selection or execution time).

7.2 Overview

If a plan is non-cooperative then an agent can simply commit to its execution and form an intention. However, if the plan is cooperative then a *cooperative intention* must be formed. The establishment of a cooperative intention involves three distinct stages.

- Firstly, an agent (which we call the initiating agent, or *initiator*) must determine which agents it wishes to cooperate with, and then ask them for assistance. To decide which agents to ask the initiating agent iterates through the steps in its plan, annotating each one with the identifiers of the agents it wishes to perform it, based on its trust and knowledge of others.
- Secondly, on receiving a request for assistance, these agents inspect their own motivations and intentions to decide whether or not to agree, and send an appropriate response to the requesting agent; an agent's motivations determine whether it *wants* to cooperate, and its existing intentions determine whether it *can* cooperate (since intentions must be consistent).
- Finally, depending on the responses the initiating agent receives, it will either be able to establish a cooperative intention and begin executing it, need to find another group of agents to ask for assistance, or it will fail to establish a cooperative intention. If

sufficient agents agree to assist then a cooperative intention is formed with those agents. However, if insufficient agents agree there are two possibilities:

- if it is possible to choose another group of agents to ask, the initiator chooses such a group and asks them for assistance, and
- if it is not possible to choose another group, failure is conceded, and any agents that have already agreed to assist are informed.

This process is repeated until either a cooperative intention has been formed, or it is neither possible to reassign agents to actions, nor to request another group.

If sufficient agents agree to cooperate then a cooperative intention can be formed. For individual actions there simply has to be at least one agent agreeing to perform the action. For joint and concurrent actions there must be at least one agent that agrees to perform each contribution and, for the former, that agent must not be required for another simultaneous contribution. This second clause is a result of the possibility of redundant annotation, where an agent might be assigned more than one contribution, and may accept both, but cannot actually perform both (as described later in this chapter). The algorithm for determining whether sufficient agents have responded to a request is given in Table 7.1, which takes an annotated plan and a set of responses, and returns true if sufficient agents have agreed to cooperate. In the algorithm the function *accepted* returns the agents that have agreed to perform a particular action.

Assuming sufficient agents accept, the next step in forming a cooperative intention is to select which agents out of the positive responses will actually be part of the cooperation (since redundant annotation may have caused multiple agents to agree to cooperate for each action). In this case, the only basis for distinguishing between agents is their trustworthiness, and so the most trusted are chosen. The chosen agents are sent a *confirmation* message to inform them that enough agents have agreed to assist, and cooperation is proceeding. Agents that are not chosen but agreed to cooperate are sent a cancel message and the nominal commitment towards them is dropped.

Inputs:

plan — the plan for which cooperation is required

responses — the set of responses agreeing to cooperate

Outputs:

true if sufficient agents have agreed to cooperate, *false* otherwise

Algorithm:

actions = extractActions(*plan*)

for *act* **in** *actions* **do**

acceptedAgents = accepted(*act*, *responses*)

if individualAction(*act*) **then**

if not sizeOf(*acceptedAgents*) \geq 1 **then**

return *false*

else

for *c* **in** contributions(*act*) **do**

acceptedC = accepted(*c*, *responses*)

if not sizeOf(*acceptedC*) \geq 1 **then**

return *false*

return *true*

Table 7.1: Algorithm to determine whether responses are sufficient to enter into cooperation

Although we assume that agents are willing communicators, it is still possible for cooperative intention formation to fail due to communication problems. For example, an agent might not respond within a reasonable time, or a response may not be received due to communication errors. A timeout mechanism is therefore incorporated into the process, so that if no response is received from a given agent before the timeout, the agent is taken to be refusing. The algorithm for generating full commitment is part of the initiator's overall algorithm for forming a cooperative intention, as given in Table 7.2. The corresponding algorithm for a participant is described later in Section 7.9.

On receiving a confirmation message, an agent knows that sufficient agents have agreed to cooperation, and a cooperative intention has been formed. It can, therefore, begin performing the actions required of it. Conversely, if a cancel message is received then an agent's assistance is no longer required, and it can drop its nominal commitment, and reduce its trust of the requesting agent (as described in Section 5.4.1). However, if for some reason neither a confirmation nor a cancel message is received (after some timeout period), then the agent assumes that its assistance is no longer required, and behaves as though it received a cancel message, i.e. it drops its nominal commitment, and updates its trust of the requester.

7.3 Annotation Strategies

In order to determine which agents to ask for cooperation, the initiating agent must consider each of the contributions in its plan and determine which is the best agent to perform it. The chosen agent is associated with the contribution by *annotating* the contribution with the identifier of that agent. The initiator must annotate each cooperative action in its plan with the identifiers of the agents it will ask to perform it. For any given contribution several agents may be asked for assistance if they can all perform the required contribution, and more than one may be listed in the annotation, providing a degree of redundancy in case some decline to cooperate. We call such annotation of a contribution with more agents than

Inputs:

plan — the plan for which cooperation is needed

agentModels — the initiating agent's models of others

Outputs:

a cooperative intention if successful, otherwise nothing

Algorithm:

repeat:

if canAnnotate(*plan*, *agentModels*) **then**

plan = annotatePlan(*plan*, *agentModels*)

else

return

 requestAssistance(*plan*)

responses = waitForResponses()

if sufficient(*responses*, *plan*) **then**

cooperativeIntention = formCooperativeIntention(*plan*)

return *cooperativeIntention*

Table 7.2: The initiator's algorithm for cooperative intention formation

are required *redundant annotation*. Alternatively, annotating each contribution with just one agent we refer to as *minimal annotation*, since the plan is annotated with the minimum number of agents required for its execution.

With redundant annotation, even if some of the chosen agents decline to cooperate, cooperation may still be successful. For example, suppose that for each action three agents are asked for assistance. If all three agents accept then the initiator can simply enter into cooperation with the most trusted agent. However, if two agents decline, then cooperation can still go ahead with the third agent. This redundancy, however, comes at a price, primarily that the cost of communication and processing the responses will be increased over minimal annotation where a single agent might be asked for each action, in the ideal case of that agent accepting. Indeed, even if using minimal annotation when some actions need to be reassigned, the communication cost may still be less, since there may be fewer agents in total to send requests to. Note, however, that at a lower level, redundant annotation offers more scope for optimisation, for example through the use of targeted broadcast messages

(which may be cheaper than communicating with several agents individually), so that the cost of communication for each agent may be reduced if a relatively large number of agents are asked. Thus, it is not necessarily true to say that redundant annotation, where n agents are asked for each action, is equivalent in communication cost to minimal annotation where the n 'th agent agrees, since it may be cheaper to send a single broadcast than to send n individual messages.

7.3.1 Choice of Annotation Strategy

At this point, it is useful to introduce the notion of a *closely coupled* and *loosely coupled* view of agent systems. Where we are concerned with the behaviour and performance of a multi-agent system as a whole rather than with a specific individual in that system, as in a multi-agent system to perform a particular task, we say that we are taking a *closely coupled* view. Conversely, where we are concerned with maximising the performance of a particular agent, without concern for the effect on the system as a whole, as with an agent designed to compete against others, such as an auction agent, this is a *loosely coupled* view.

Now, in the closely coupled view, redundant annotation may have negative effects on the group's efficiency since there will obviously be some overhead involved in agents agreeing to cooperate. In particular an agent may be unnecessarily constrained while committed to cooperating in this way (though perhaps not actually being needed), which may have prevented it from doing something else beneficial to itself or the group as a whole. Thus, although redundant annotation increases the likelihood of getting agreement to cooperate without reassigning actions, it may be counter-productive in this respect.

In the loosely coupled view, when concerned with maximising individual performance without consideration of others, redundant annotation may not be successful over a period of time. If an agent is asked for assistance and agrees to provide it, only to be turned down later, its trust of the requesting agent will tend to decrease, since the requester did not honour the request and may have cost the provider time and caused it to constrain its

actions unnecessarily, etc. While the effect may be negligible in the short term, over an extended period the decreased trust may cause the provider to decline to cooperate. Thus, if at a later point there is only one agent with the appropriate capabilities, that agent may refuse to cooperate because it does not trust the requester; it has been inconvenienced too many times.

Ultimately, the best strategy in terms of redundant or minimal annotation is determined by both the domain itself and the overall perspective (of maximising system or individual performance). For example, if communication is relatively inexpensive and the aim is to maximise the individual performance of the agent seeking assistance in the short term, redundant annotation is sensible, whereas expensive communication and concern over unnecessarily constraining the actions of others suggests that minimal annotation be used.

Overarching these issues, however, is the importance to the initiator of its goal, since if a goal is important, redundant annotation may be justified despite any concern for the performance of the overall system. It is, therefore, desirable for an agent to be able to choose between these strategies dynamically, according to the current situation, and we consider both possibilities in the remainder of this chapter. In order to deal with this, we introduce the notion of a *redundancy threshold* to determine whether to use redundant annotation. If the motivational value of a goal is greater than this threshold then redundant annotation is used. However, since the redundant approach should only be used sparingly this threshold must be sufficiently high that the redundant approach is only occasionally used.

7.3.2 Pre-annotated Plans

In Chapter 6 we described how an agent chooses between applicable plans for its goal based, in part, on the agent's trust of the participants capable of performing actions in its plan. Since these are factors that must be considered again for plan annotation, it might be more efficient for an agent to store this information when selecting its plan, and reuse it at annotation time. Several issues affect whether this mechanism is appropriate, which we

discuss below.

Typically, an agent will have several intentions at any given time, the execution of which can be interleaved according to the intensities of its motivations. An agent acts upon the intention that is of most motivational importance and, as the intensities of its motivations fluctuate, the intention fitting this criterion may change. Hence, there may be a delay between selecting a plan and establishing a cooperative intention in its favour, during which time the trust of others may have changed. Also, if the selected plan is partial and contains subgoals then these subgoals are not dealt with until necessary, and there will certainly be a delay between selecting a plan, and satisfying any subgoals contained in it. The longer the delay, the higher the likelihood that an agent's trust in another will have changed, since trust changes over time. Any information stored at plan selection time about the trust of others may not be correct after such a delay. In addition to an agent's trust of others changing over time, its beliefs about their capabilities can also change. For example, if an agent is observed performing an action that was not known to be within its capabilities, then the model of that agent can be updated to include it.

In deciding whether an agent should store information at plan selection time to use for plan annotation, there is a trade-off between the computation saved at annotation time on the one hand, and the overhead of storing it and the cost of using outdated information on the other. It is our view that since such information is likely to change, the disadvantages of using stored information outweigh the potential advantages, and we do not consider this option further. Certainly, however, the model of cooperation we present can be changed to include this possibility while remaining within our underlying framework.

7.4 Plan Annotation

An agent annotates its plan by considering in turn the steps contained within it that require actions to be performed. There are three categories of action step that a plan might contain

(individual, joint, and concurrent) and we discuss each of these later in this section, but first we discuss the agents with which a plan can be annotated.

7.4.1 Agents to Annotate

The simplest method for annotating a plan with agents is for the initiating agent to choose agents for each contribution in turn. Since it is important to minimise the risk involved in interaction this annotation uses the most trusted agent (or agents) capable of performing an action.

If a plan contains only individual actions, and not joint or concurrent actions, then each action can be annotated with the most trusted agents that are believed to have the appropriate capabilities. The number of agents to be asked depends on whether redundant or minimal annotation is being used. In general, each action is annotated with the n most trusted agents, where n is an integer. With minimal annotation, $n = 1$, whereas in redundant annotation, $n > 1$. Note that if $n > 1$ and the number of agents having the required capabilities is less than n (but more than 1) the agent must simply annotate the plan with all those agents, rather than trying to find others with the required capabilities in order to annotate the plan with n agents. If no agents are known to have the required capabilities then plan annotation fails.

Although this considers whether agents are trusted, it does not consider whether they are *distrusted* (i.e. are trusted below a minimum trust threshold as discussed in Section 5.4). If the only agents that are believed to have the required capabilities are distrusted, then it may be better for the assignment of agents to actions to fail, rather than enter into cooperation with a group of distrusted agents, since they are considered likely to renege on their commitments. It is possible, therefore, for plan annotation to fail because all the agents having the required capabilities for a given action are not sufficiently trusted. In Section 5.4 we introduced the notion of a *minimum trust threshold*, such that agents trusted under that threshold are considered distrusted. Agents that are distrusted are not annotated to a plan;

thus if all the agents capable of performing a particular action are distrusted then plan annotation fails (since none of them can be annotated).

We formalise whether a particular agent should be considered for annotation as follows — if it is trusted above the minimum trust threshold then it is considered for annotation, otherwise it is not, as specified below.

$$\begin{array}{l}
 \text{considerForAnnotation} : \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \text{bool} \\
 \hline
 \forall id : \text{AgentID}; ms : \mathbb{P} \text{AgentModel}; t : \mathbb{R} \bullet \\
 \quad (\text{trustOfAgent } id \ ms \ > \ t) \Rightarrow \text{considerForAnnotation } id \ ms \ t = \text{true} \wedge \\
 \quad (\text{trustOfAgent } id \ ms \ \leq \ t) \Rightarrow \text{considerForAnnotation } id \ ms \ t = \text{false}
 \end{array}$$

7.4.2 Individual Action Annotation

Recall that a contribution is defined to be an action, along with the identifier of the agent that is to perform it. Where we are concerned with minimal annotation this is sufficient to represent the agent annotated to a contribution. However, when we consider redundant annotation, this is insufficient, since we need to associate a set of agent identifiers with a particular action. Therefore, before we can specify the function for annotating a contribution we must introduce the notion of an *AnnotatedContribution*, where an action is annotated with a *set* of identifiers. We formalise the annotation of a contribution in the schema *AnnotateContribution*, in which n and t represent the number of agents to annotate a contribution with, and the minimum trust threshold respectively. This function specifies that an individual contribution is annotated with the n most trusted agents, provided their associated trust values are greater than t .

$$\begin{array}{l}
 \text{AnnotatedContribution} \\
 \hline
 \text{symbol} : \text{ActSym} \\
 \text{terms} : \text{seq Term} \\
 \text{agents} : \mathbb{P} \text{AgentID}
 \end{array}$$

$$\begin{array}{l}
\text{AnnotateContribution} \\
\hline
\text{annotateContribution} : \text{Contribution} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \\
\rightarrow \text{AnnotatedContribution} \\
\hline
\forall c : \text{Contribution}; ms : \mathbb{P} \text{AgentModel}; n : \mathbb{Z}; t : \mathbb{R}; \\
ac : \text{AnnotatedContribution} \bullet c.\text{symbol} = ac.\text{symbol} \wedge \\
c.\text{terms} = ac.\text{terms} \wedge ac.\text{agents} = \{id : \text{AgentID} \mid \\
id \in \text{ran}(\{i : \mathbb{Z} \mid i \leq n\} \upharpoonright \text{orderedCapableAgents } c \text{ } ms) \wedge \\
\text{considerForAnnotation } id \text{ } ms \text{ } t = \text{true} \bullet id\}
\end{array}$$

7.5 Annotating Simultaneous Actions

The approach described above is only applicable for plans that do not contain joint or concurrent actions. Recall that joint and concurrent actions are made up of a set of contributions, each of which is an individual action that must be annotated with agents. The main consideration in annotating a plan containing such actions is that an agent must not be required to execute two or more contributions simultaneously, since we assume that agents can only perform one action at a given time. In minimal annotation this is simply achieved by not annotating an agent to more than one simultaneous contribution. Redundant annotation, however, is more complex, because an agent might be annotated to several simultaneous contributions, and its assistance requested for all of them.

Since an agent can only perform one action at a time, and its intentions must be consistent, an agent asked to assist for several simultaneous contributions can only agree to one of them at most (according to its motivations and intentions), or its intentions would become inconsistent. Redundant annotation of an agent to several simultaneous contributions allows that agent a choice about which, if any, of the contributions it performs. The key requirement when annotating the same agent to more than one simultaneous contribution is that agreement is necessary for at most one of them. For example, a joint action comprising two contributions each annotated with the same two agents is a valid annotation, because either agent can perform either contribution, as illustrated by the annotation on the left in

<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="border-bottom: 1px solid black; padding: 2px;">Contribution</th> <th style="border-bottom: 1px solid black; padding: 2px;">Annotation</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;"><i>contribution₁</i></td> <td style="padding: 2px;">α_1, α_2</td> </tr> <tr> <td style="padding: 2px;"><i>contribution₂</i></td> <td style="padding: 2px;">α_1, α_2</td> </tr> </tbody> </table>	Contribution	Annotation	<i>contribution₁</i>	α_1, α_2	<i>contribution₂</i>	α_1, α_2	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="border-bottom: 1px solid black; padding: 2px;">Contribution</th> <th style="border-bottom: 1px solid black; padding: 2px;">Annotation</th> </tr> </thead> <tbody> <tr> <td style="padding: 2px;"><i>contribution₁</i></td> <td style="padding: 2px;">α_1, α_2</td> </tr> <tr> <td style="padding: 2px;"><i>contribution₂</i></td> <td style="padding: 2px;">α_1, α_2</td> </tr> <tr> <td style="padding: 2px;"><i>contribution₃</i></td> <td style="padding: 2px;">α_1, α_2</td> </tr> </tbody> </table>	Contribution	Annotation	<i>contribution₁</i>	α_1, α_2	<i>contribution₂</i>	α_1, α_2	<i>contribution₃</i>	α_1, α_2
Contribution	Annotation														
<i>contribution₁</i>	α_1, α_2														
<i>contribution₂</i>	α_1, α_2														
Contribution	Annotation														
<i>contribution₁</i>	α_1, α_2														
<i>contribution₂</i>	α_1, α_2														
<i>contribution₃</i>	α_1, α_2														
valid annotation	invalid annotation														

Table 7.3: Valid and invalid annotations

Table 7.3. Alternatively, a joint action comprising three contributions, each annotated with the same two agents, is not a valid annotation, since even if both agents agree to perform a contribution, there will be a third contribution for which no agent has agreed (illustrated by the annotation on the right in Table 7.3). Where we are concerned with annotating concurrent actions it is possible for an agent to be annotated to more than one thread of execution since synchronisation is only required at the beginning and end of a concurrent action block, and all contributions do not necessarily have to be performed simultaneously.

7.5.1 Joint Actions

In formalising the annotation of joint actions we rely on a number of auxiliary functions, the specification of which can be found in Appendix A. Firstly, *contribSeq* takes a joint action as its argument and extracts the contributions that it comprises, returning them as a sequence. In turn, the function *allPossibleAnnotations* takes such a sequence of contributions, and determines the set of all possible minimal annotations, where an agent is associated with a contribution if it is capable of performing it and is trusted above the minimal trust threshold. We refer to these possible annotations as *candidate assignments*. Since the contributions in a joint action must be performed simultaneously, an agent must not be assigned to more than one contribution in a candidate assignment. If in a given candidate assignment no agent is assigned to more than one contribution it is said to be *valid* and, conversely, if a candidate assignment assigns an agent to more than one contribution, then

it is *invalid*. The function *validAssignments* takes the set of candidate assignments, and removes those that are not valid. Finally, the function *orderedAnnotations* takes the resulting set, and orders them according to the combined trust of the agents involved.

We can now introduce the notion of an *AnnotatedJointAction*, as a set of annotated contributions, and specify how to annotate a joint action. The function *annotateJointAction* takes a joint action, a set of agent models, an integer representing the number of agents to annotate each contribution with, and a minimum trust threshold and returns an annotated joint action. The annotation is determined by extracting the appropriate number of assignments from the front of the ordered list possible valid candidate assignments, using the auxiliary functions introduced above. Each of these assignments associates one agent with each contribution, and they are combined into a single (possibly redundant) annotation using the auxiliary function *annotate*.

<p><i>AnnotatedJointAction</i></p> <p><i>contributions</i> : \mathbb{P} <i>AnnotatedContribution</i></p> <p><i>#contributions</i> ≥ 2</p>
--

<p><i>AnnotateJointAction</i></p> <p>\exists <i>AnnotateJointActionAuxiliary</i></p> <p><i>annotateJointAction</i> : <i>JointAction</i> \rightarrow \mathbb{P} <i>AgentModel</i> \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \rightarrow <i>AnnotatedJointAction</i></p> <p><i>annotate</i> : $\text{seq}(\mathbb{P}(\text{Contribution} \times \text{AgentID})) \rightarrow$ <i>AnnotatedJointAction</i></p> <p>$\forall ja : \text{JointAction}; ms : \mathbb{P} \text{AgentModel}; n : \mathbb{Z}; t : \mathbb{R} \bullet$ <i>annotateJointAction</i> <i>ja</i> <i>ms</i> <i>n</i> <i>t</i> = <i>annotate</i> ($\{i : \mathbb{Z} \mid i \leq n\}$ <i>(orderedAnnotations (validAnnotations (allPossibleAnnotations</i> <i>(contribSeq ja) ms t)) ms)</i>)</p> <p>$\forall s : \text{seq}(\mathbb{P}(\text{Contribution} \times \text{AgentID})); aja : \text{AnnotatedJointAction} \mid$ <i>annotate</i> <i>s</i> = <i>aja</i> $\bullet \forall c : \text{Contribution} \bullet c \in \text{extractContributions (s 1)}$ $\Leftrightarrow (\exists_1 ac : \text{AnnotatedContribution} \bullet ac \in aja.\text{contributions} \wedge$ <i>c.symbol</i> = <i>ac.symbol</i> \wedge <i>c.terms</i> = <i>ac.terms</i> \wedge <i>ac.agents</i> = <i>agentsOfContributionAs (ran s) c)</i></p>

7.5.2 Concurrent Actions

In a similar manner, we can specify that an annotated concurrent action comprises a set of annotated contributions and joint actions. Annotation of a concurrent action involves annotating each of its components, and including the result in the annotated concurrent action, as specified below in the schema *AnnotateConcurrentAction*. The function *annotateCAcomponent* takes a component of a concurrent action (a contribution, or set of contributions representing a joint action) and annotates it using the mechanisms described above for actions and joint actions, as specified by *annotateConcurrentAction*.

$$ACcomponent ::= AContrib\langle\langle AnnotatedContribution \rangle\rangle \\ | AJA\langle\langle \mathbb{P} AnnotatedContribution \rangle\rangle$$

$\text{AnnotateConcurrentAction}$
$contributions : \mathbb{P} ACcomponent$
$\#contributions \geq 2$

$\text{AnnotateConcurrentAction}$
$\exists \text{AnnotateContribution}$
$\exists \text{AnnotateJointAction}$
$\text{annotateConcurrentAction} : ConcurrentAction \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \\ \rightarrow AnnotatedConcurrentAction$
$\text{annotateCAcomponent} : CAcomponent \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \\ \rightarrow ACcomponent$
$\forall cac : CAcomponent; ms : \mathbb{P} AgentModel; n : \mathbb{Z}; t : \mathbb{R}; \\ acac : ACcomponent \bullet \text{annotateCAcomponent } cac \ ms \ n \ t = acac \\ \Leftrightarrow (\exists c : Contribution \mid Contrib(c) = cac \bullet \\ acac = AContrib(\text{annotateContribution } c \ ms \ n \ t)) \vee \\ (\exists cs : \mathbb{P} Contribution; ja : JointAction \mid JA(cs) = cac \bullet \\ ja.contributions = cs \wedge acac = \\ AJA((\text{annotateJointAction } ja \ ms \ n \ t).contributions))$
$\forall ca : ConcurrentAction; ms : \mathbb{P} AgentModel; n : \mathbb{Z}; t : \mathbb{R}; \\ aca : AnnotatedConcurrentAction \bullet \\ \text{annotateConcurrentAction } ca \ ms \ n \ t = aca \wedge \\ aca.contributions = \{cac : CAcomponent \mid \\ cac \in ca.contributions \bullet \\ \text{annotateCAcomponent } cac \ ms \ n \ t\}$

7.5.3 Annotated Plans

The notion of an annotated plan is formalised below in the schema *AnnotatedPlan*, in which all contributions are annotated with a *set* of agents. Each contribution is annotated with a set, rather than the individual agent that will execute it since, at this stage, the annotation represents the agents to request assistance from. Thus, to allow for redundant annotation, a contribution is associated with a set of agents rather than an individual. However, before a cooperative intention can be formed, an agent must select one agent for each contribution and modify the annotated plan accordingly. The schema *AnnotatePlan* contains two functions, *annotateStep* and *annotatePlan*, the first of which takes a plan step and applies the appropriate annotation function (unless the step is a goal in which case it is not changed), and the second takes a plan and annotates each step in that plan, returning the corresponding annotated plan.

$$\begin{aligned} APlanStep ::= & AIndividual\langle\langle AnnotatedContribution \rangle\rangle \\ & | AJoint\langle\langle \mathbb{P} AnnotatedContribution \rangle\rangle \\ & | AConcurrent\langle\langle \mathbb{P} ACAcomponent \rangle\rangle \\ & | ASubgoal\langle\langle Goal \rangle\rangle \end{aligned}$$

AnnotatedPlan

achieves : *Goal*

preconditions : \mathbb{P} *Literal*

body : seq *APlanStep*

AnnotatePlan

\exists AnnotateContribution

\exists AnnotateJointAction

\exists AnnotateConcurrentAction

$annotatePlan : Plan \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \rightarrow AnnotatedPlan \bullet$

$annotateStep : PlanStep \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \rightarrow APlanStep$

$\forall p : Plan; ms : \mathbb{P} AgentModel; n : \mathbb{Z}; t : \mathbb{R}; ap : AnnotatedPlan \bullet$

$annotatePlan p ms n t = ap \Leftrightarrow p.achieves = ap.achieves \wedge$

$p.preconditions = ap.preconditions \wedge$

$(\forall n : \mathbb{Z} \mid n \leq \#p.body \bullet ap.body n =$

$annotateStep (p.body n) ms n t)$

$\forall ps : PlanStep; ms : \mathbb{P} AgentModel; n : \mathbb{Z}; t : \mathbb{R}; aps : APlanStep \bullet$

$annotateStep ps ms n t = aps$

$\Leftrightarrow (\exists c : Contribution \bullet Individual(c) = ps \wedge aps =$

$AIndividual(annotateContribution c ms n t)) \vee$

$(\exists cs : \mathbb{P} Contribution; ja : JointAction \mid ja.contributions = cs \bullet$

$Joint(cs) = ps \wedge aps = AJoint(($

$annotateJointAction ja ms n t).contributions)) \vee$

$(\exists cac : \mathbb{P} CComponent; ca : ConcurrentAction \mid$

$ca.contributions = cac \bullet Concurrent(cac) = ps \wedge aps =$

$AConcurrent((annotateConcurrentAction$

$ca ms n t).contributions)) \vee$

$(\exists g : Goal \bullet Subgoal(g) = ps \wedge aps = ASubgoal(g))$

An agent can represent an annotated plan, by simply associating the identifiers of the appropriate agents with each contribution in the body of the plan. For example, if the actions in the plan for storing a small box in the warehouse domain were assigned to agent2, then an agent might represent this as follows.

```
name: storeSmallPlan
  achieves: [$(location, [box, room]),
    not (holding, [agent2, box])$]
  preconditions: [(location, [box, loc]),
    (type, [box, small, shortTerm])]
  body: [<goal [$(location, [agent2, loc])$]>,
    <action (pickup, [agent, box])=agent2>,
    <goal [$(location, [agent2, room])$]>,
    <action (putdown, [agent, box])=agent2>]
```

7.6 Soliciting Commitment to Cooperate

After deciding which agents to try to cooperate with (by annotating its plan), an agent must request assistance from the agents with whose identifiers the plan is annotated. There are several options for how much information to include in a request for assistance. In particular an agent attempting to initiate cooperation can communicate either

1. the whole plan, but without annotations,
2. just the actions it wants the potential participant to perform,
3. the goal for which assistance is required, along with the actions it wishes the potential participant to perform,
4. the whole plan, annotated only with the actions it wishes the potential participant to perform, or
5. the whole annotated plan.

These options provide varying degrees of information to the receiver, and support different objectives, represented by the loosely coupled and closely coupled views, as we discuss below.

- The first alternative of communicating the whole plan without annotations, does not in general give sufficient information for the participant to make a decision about whether or not to cooperate, since it does not specify which actions it should perform. Without knowing which actions are requested of it, an agent cannot determine whether they will conflict with its intentions or their motivational value. There are a small number of exceptional circumstances in which an agent could make a decision; for example, if all actions in the plan, and the goal it achieves, are of motivational value, and the agent has no other intentions, then it can decide to cooperate. In general,

however, this is not the case, and more information is required in a request. Thus, we reject the first alternative.

- Remember that there must be some motivational justification for an agent choosing to perform a particular action, and although the overall goal must be of motivational value (or it would not have merited committing to), the particular actions required to achieve it might not be. For example, achieving the goal of getting a paper accepted for a conference is likely to have motivational value, but the actions involved in proof-reading and correcting are less likely to be valuable in themselves. Thus, while the *end* may have motivational value the *means* may not, if considered out of the context of the overall goal. In practice an agent's motivations are typically mitigated by the achievement of goals, rather than the performance of particular actions, although there are exceptions. Thus, an agent is unlikely to gain assistance for its goal if its request contains only the actions that it wishes to be performed, and not the goal that they achieve (as in the second alternative above). The exception to this is if the action is valued by the potential participant and the goal is not. For example, if you gain value from performing the action of driving, and I wish you to drive a getaway car in a robbery for me, then the negative motivational effect of achieving the goal would outweigh the benefit obtained from driving (assuming you are a law-abiding citizen). Thus, in this situation if I *know* that the goal is of zero or negative motivational value, then I might make my request giving only the action for which assistance is sought.
- The third alternative requests assistance from the potential participant for a particular set of contributions, and towards a particular goal. This allows an agent to consider both the motivational value of the actions it is requested to perform, and the value it would gain if the overall goal is achieved.
- The fourth alternative also includes the complete plan, without the annotations related to other agents. This additional information can influence the potential participant's decision about whether to cooperate. If the participant is informed of the plan then it

knows what other actions will be performed in the achievement of the goal. If it has a goal or intention that some action in the plan is not performed (by any agent), then it may refuse even if it would otherwise have accepted, based solely on the goal and actions *it* is to perform.

- The final alternative includes both the plan, and the complete set of annotations; if the participant is informed of the other annotations in the plan, it is given information about which agents are likely to be involved in the cooperative interaction. If it has a goal or intention of not cooperating with another of the annotated agents then it may also refuse, even if would accept were its choice based only on the goal and actions *it* is to perform. Note that as discussed earlier, communicating *redundant* annotations makes recipients aware of the redundancy and the potential unnecessary constraints this may impose upon them. Thus, if the fifth alternative is used, the requesting agent must process the annotations contained in the request to remove redundant annotation of the potential participant¹.

In our framework, therefore, an agent has a choice of the latter four options. The choice about which of these approaches to use is a macro level consideration determined by whether a loosely or closely coupled approach is being taken. We therefore simply assume that an agent uses one of these mechanisms, without specifying which. Formally, there are four possible types of request, containing the actions for which assistance is required, the actions and the overall goal, the plan annotated only with the requestee’s actions, or the whole annotated plan. We specify this as follows, where the actual request types (shown within “⟨⟨” and “⟩⟩”) are specified in Appendix A.

```
Request ::= ActionRequest⟨⟨RequestActions⟩⟩
          | GoalActionRequest⟨⟨RequestGoalActions⟩⟩
          | PartiallyAnnotatedPlanRequest⟨⟨RequestPartiallyAnnotatedPlan⟩⟩
          | AnnotatedPlanRequest⟨⟨RequestAnnotatedPlan⟩⟩
```

¹It could be argued that all redundant annotations should be removed in case an agent infers that if another is redundantly annotated, it may be treated similarly. However, we do not consider the case where an agent has such inference abilities, and so do not consider this situation.

As discussed above, an agent’s request for assistance should include only the actions for which it needs help if it believes that the goal is of zero or negative motivational value to the provider. This is specified in the function *useActionRequest* below, which takes an annotated plan and set of agent models, and returns true if the goal is believed to be of zero or negative motivational value to one or more of the agents being requested, and the agent’s request should be of this form. Our specification of the function relies upon auxiliary functions *allAgents* and *believedMV*, the former of which extracts the agents annotated in a given plan, and the latter of which returns the believed motivational value of a goal to another agent, based on information from the agent’s models of others. These auxiliary function are also specified in Appendix A.

<p><i>UseActionRequest</i> _____</p> <p>\exists<i>AnnotatedPlanAuxiliary</i></p> <p><i>useActionRequest</i> : <i>AnnotatedPlan</i> \rightarrow \mathbb{P} <i>AgentModel</i> \rightarrow <i>bool</i></p> <hr style="border: 0.5px solid black;"/> <p>\forall <i>ap</i> : <i>AnnotatedPlan</i>; <i>ms</i> : \mathbb{P} <i>AgentModel</i> •</p> <p style="padding-left: 20px;"><i>useActionRequest ap ms</i> = true</p> <p style="padding-left: 40px;">$\Leftrightarrow (\exists id : AgentID \mid id \in allAgents\ ap \bullet$</p> <p style="padding-left: 60px;"><i>believedMV ap.achieves id ms</i> \leq 0) \wedge</p> <p style="padding-left: 20px;"><i>useActionRequest ap ms</i> = false</p> <p style="padding-left: 40px;">$\Leftrightarrow (\forall id : AgentID \mid id \in allAgents\ ap \bullet$</p> <p style="padding-left: 60px;"><i>believedMV ap.achieves id ms</i> $>$ 0)</p>

7.7 Requesting Assistance

As described above, there are two steps in requesting assistance: first, for every agent annotated to each contribution in a plan, determine how much information to give them (about the goal, plan, and members of the proposed group), and secondly make the request itself. Since cooperative intention establishment may involve several rounds of requesting, some agents may have already been asked for assistance for a previous action, in which case it is possible that an agent may have already accepted a request. Here, some form of commitment to perform the (previously requested) action will have been formed, and if an agent

Inputs:

agent — the agent to request assistance from

plan — the plan for which cooperation is needed

acceptedAgents — the set of agents that have already agreed to cooperate

Outputs:

requests sent to the agents annotated in *plan*

Algorithm:

```
if agent in acceptedAgents then
  for act in actionsAccepted(agent) do
    if notAssigned(agent, act, plan) then
      cancel(agent, act)
    else
      if participantsChanged(act, plan) then
        if informed(agent, participants(act, plan)) then
          re-request(agent, act)
      for act in assigned(agent, plan) do
        if act not in actionsAccepted(agent) then
          request(agent, action, plan)
  else
    request(agent, plan)
return
```

Table 7.4: The initiator's algorithm for requesting assistance

has agreed to perform some action to which it is no longer annotated in the latest plan annotation, it must be informed that its commitment is unnecessary. Similarly, if the agent has already agreed to perform the same action that it is currently annotated to then there is no need to ask it again

If the action is part of a joint or concurrent action which is currently annotated with a different group of agents, and the agent was informed of the original annotation, its decision to cooperate may be affected by the composition of the group, and the agent must be informed of the changes. Finally, note that if a previous request for assistance was refused, it does not affect subsequent requests, since no commitment will have been formed. Table 7.4 presents the algorithm for requesting (or re-requesting assistance) from a particular agent.

If the agent has already agreed to cooperate then the actions for which it has agreed to cooperate are considered. If it is no longer assigned an action it has agreed to perform, then it is informed that cooperation is no longer required (for that action). Similarly, if the participants of a joint or concurrent action have changed, then it is informed of these changes. Finally, it is sent a request for any actions that it has not already agreed to perform.

7.8 Nominal Commitment

If an agent changes its mind after requesting assistance (for example if its motivations change and it drops its goal) other agents may have formed a commitment to assist and have constrained their actions unnecessarily — potentially causing their trust of the initiating agent to decrease when they discover this. Informing agents of such changes after a request is a means to safeguard against becoming distrusted. This issue arises only between requesting assistance and the formation of a cooperative intention (assuming others accept), since once a cooperative intention is formed, the agents involved are required to inform others as a consequence of their commitment.

To achieve this commitment to informing others we introduce the notion of a *nominal commitment* with respect to a set of agents such that, in the case of it dropping its goal, an agent will inform all other agents in that set. A nominal commitment, therefore, acts as a placeholder commitment until a full cooperative intention is established. Before requesting assistance, therefore, a nominal commitment must be formed with respect to the agents whose assistance is sought, ensuring that the initiating agent will inform the requested agents if it changes its mind about requiring assistance. The process of establishing a cooperative intention may involve several rounds of plan annotation before it is successful, in that others will be asked for assistance and may refuse, leading to another set of agents being chosen and asked. For example, suppose an agent annotates its plan with three other agents, α_1 , α_2 , and α_3 and requests assistance from them, then it must form a nominal commitment towards those agents, to inform them if assistance is no longer required. If α_1

Inputs:

plan — the plan to which nominal commitment is required

participants — the participants in *plan*

Outputs:

a nominal commitment towards the agents annotated in *plan*

Algorithm:

```
if not committedTo(plan) then
    form(nominalCommitment(plan, participants))
else
    currentCommitment = retrieveCommitment(plan)
    existingParticipants = extractAgents(currentCommitment)
    for a in (existingParticipants \ participants) do
        remove(a, currentCommitment)
    for a in (participants \ existingParticipants) do
        add(a, currentCommitment)
return
```

Table 7.5: Algorithm for updating nominal commitment

and α_2 accept, but α_3 declines to assist then the initiating agent must select another set of agents to cooperate with — it must re-annotate its plan, and request assistance from this new set of agents.

Each round of plan annotation involves forming a nominal commitment and requesting assistance. Therefore, if assistance has already been requested for a previous annotation of the plan, a nominal commitment will exist toward the agents whose assistance was requested. A new nominal commitment does not need to be formed; instead, the annotation of agents to whom the commitment is made towards are updated. Those agents that are not in the current annotation are removed from the commitment, since there is no need to inform them if assistance is no longer required, and any newly annotated agents are added. If no requests have previously been made for (a prior annotation of) the plan, then a new nominal commitment is formed to the agents contained in the current annotation. Consider the example of an agent requesting assistance, and forming a nominal commitment towards, three agents, α_1 , α_2 , and α_3 . Now, suppose α_3 declines and the agent re-annotates its plan with

agents α_1 , α_2 , and α_4 , such that the former two are given the same tasks and α_4 assigned to the task for which α_3 declined. The initiator must update its nominal commitment to be towards this new set of agents, i.e. it must modify its commitment to α_3 to be towards α_4 . This process is described by the algorithm for updating nominal commitment, given in Table 7.5.

We formalise the formation of a nominal commitment by first defining the representation of such a commitment as a plan and a set of agents to whom the commitment is made, as specified below in the schema *NominalCommitment*. The formation of a nominal commitment itself is specified in the schema *FormNominalCommitment*, which is based on the algorithm in Table 7.5. Additionally, we define the function *committedTo*, which takes an annotated plan and returns true if the agent already has a nominal commitment towards that plan, and false otherwise. This function is used to determine whether a new nominal commitment is to be adopted, or whether to update the agents in the existing commitment.

<i>NominalCommitment</i> <i>plan</i> : <i>AnnotatedPlan</i> <i>agents</i> : \mathbb{P} <i>AgentID</i>

FormNominalCommitment

Δ Agent

\exists AnnotatedPlanAuxiliary

newPlan : AnnotatedPlan

committedTo : AnnotatedPlan \rightarrow bool

$$\begin{aligned} & \forall ap : \text{AnnotatedPlan} \bullet \exists c : \text{NominalCommitment} \mid \\ & \quad c \in \text{nominalCommitments} \bullet c.\text{plan.achieves} = ap.\text{achieves} \\ & \quad \Leftrightarrow \text{committedTo } ap = \text{true} \\ & \forall ap : \text{AnnotatedPlan} \bullet \forall c : \text{NominalCommitment} \mid \\ & \quad c \in \text{nominalCommitments} \bullet c.\text{plan.achieves} \neq ap.\text{achieves} \\ & \quad \Leftrightarrow \text{committedTo } ap = \text{false} \\ & \text{committedTo } \text{newPlan} = \text{false} \\ & \Rightarrow (\exists c : \text{NominalCommitment} \bullet c.\text{plan} = \text{newPlan} \wedge \\ & \quad c.\text{agents} = \text{allAgents } \text{newPlan} \wedge \\ & \quad \text{nominalCommitments}' = \text{nominalCommitments} \cup \{c\}) \\ & \text{committedTo } \text{newPlan} = \text{true} \\ & \Rightarrow (\exists c, c' : \text{NominalCommitment} \mid c \in \text{nominalCommitments} \bullet \\ & \quad c.\text{plan.achieves} = \text{newPlan.achieves} \wedge \\ & \quad c \notin \text{nominalCommitments}' \wedge c' \in \text{nominalCommitments}' \wedge \\ & \quad c'.\text{plan} = \text{newPlan} \wedge c'.\text{agents} = \text{allAgents } \text{newPlan}) \end{aligned}$$

7.9 Committing to Cooperate

In this section we consider the criteria by which a participant decides whether or not to cooperate after receiving a request for assistance. There are two key factors in this decision that arise from the autonomous nature of the agents involved. Firstly, the trust ascribed to the requester determines the perceived risk of interacting with it and, secondly, the motivational value that would be attained from cooperating determines the potential benefit to the providing agent.

An agent receiving a request for assistance with respect to a set of actions must determine whether the requesting agent is sufficiently trusted before entering into a cooperative intention and, if it is, it must determine the motivational value of the actions it is asked to perform. If the request also includes information about the overall goal and plan for

which cooperation is required, the value of that goal and plan must also be considered. If the request has no motivational value then the agent will not cooperate and must inform the requester that it is declining, otherwise it must check whether the requested actions are compatible with its existing intentions.

An agent's intentions must be consistent, and if an agent receives a request for assistance from a trusted agent that carries some positive motivational value but would lead to an intention conflict, it must choose between existing intentions and the request. Such a conflict may be an explicit conflict of goals, or it may be that a known effect of performing the requested actions conflicts with an existing intention. Therefore, an agent must select which of the conflicting intentions it will discard. One possibility is to consider how much effort has already been invested in attempting to achieve the existing intention, how much is still required, and the trust in others expected to be involved. However, the computational cost it requires in itself, in particular with respect to estimating how much effort is still required, is prohibitive because it involves considering all possible plan elaborations. A more useful (and simple) alternative is to select the intention with the highest motivational value. Thus, if the existing intention has a higher value, the agent will decline to cooperate for the new request. Otherwise, the agent will drop its existing intention and agree to cooperate and form a nominal commitment.

An agent accepts a request, allowing cooperation to ensue only if

- the requester is considered trusted,
- there is no conflict with existing intentions, and
- the request is of positive motivational value.

As described in Section 7.8, when accepting a request an agent forms a *nominal commitment*, in that it becomes committed to informing others if it rescinds its acceptance. The algorithm for a recipient to process a request for assistance is shown in Table 7.6, which states that an agent must first consider the trust of the requester, then the motivational value

Inputs:

request — the request for assistance

trustThreshold — the threshold under which agents are considered distrusted

intentions — the agent's intentions

Outputs:

a message accepting or declining to cooperate, and

a nominal commitment if accepting

Algorithm:

```
initiator = sender(request)
if trust(initiator) < trustThreshold then
  decline(initiator, request)
  return false
else
  if motivationalValue(request) ≤ 0 then
    decline(initiator, request)
    return false
  else
    if conflict(request, intentions) then
      if higherMotivationalValue(request,
        conflicting(request, intentions)) then
        drop(conflicting(request, intentions))
      else
        decline(initiator, request)
        return false
    formNominalCommitment(initiator, request)
    accept(initiator, request)
return true
```

Table 7.6: The recipient's algorithm for processing a request

of the request, and finally whether the request conflicts with its existing intentions. If the conditions described above are met then the agent sends an acceptance message, and forms an appropriate nominal commitment, otherwise a message declining to assist is sent. We describe below how an agent considers the trust of the requester, and how it determines the motivational value of a request.

7.9.1 Trust in Requesting Agent

Before checking the motivational value of the request, and whether it conflicts with existing intentions, the requesting agent must be checked for adequate trustworthiness. The trust of the requester is checked before determining the motivational value of the request, since it is much cheaper in computational terms to check trustworthiness than to assess motivational value.

Cooperation should be avoided if the requester is not trusted, since if it drops its part of a cooperative intention, any commitment and action on behalf of the participant is likely to have been wasted. To determine whether the requester is trustworthy, its associated trust value must be assessed, with respect to a *cooperation threshold* over which (for this purpose) it is considered *sufficiently trusted*, and under which it is not. Note that this threshold is distinct from that described in Section 7.4 which is concerned with the notions of trust and distrust with respect to plan annotation; here we are concerned with ensuring an agent is sufficiently trusted to consider expending effort in assisting it — an agent might be trusted in the context of plan annotation, but not be sufficiently trusted to offer assistance to it. An agent will not assist another that is not sufficiently trusted, even if the request might appear to have positive motivational value. The function *consideredTrusted* below formalises this, and takes an agent, a set of agent models, and a cooperation threshold, and returns true if the trust of the agent is greater than the threshold.

$$\text{consideredTrusted} : \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \text{bool}$$

$$\forall \text{requester} : \text{AgentID}; \text{ms} : \mathbb{P} \text{AgentModel}; t : \mathbb{R} \bullet$$

$$\text{trustOfAgent requester ms} > t$$

$$\Rightarrow \text{consideredTrusted requester ms } t = \text{true} \wedge$$

$$\text{trustOfAgent requester ms} \leq t$$

$$\Rightarrow \text{consideredTrusted requester ms } t = \text{false}$$

7.9.2 Commitment to Actions

After checking that the requester is trusted, the motivational value of the request is determined. The minimum information that can be included in a request is the set of actions for which assistance is required (as described in Section 7.6). On receiving such a request, an agent can assess the motivational value that it would get from performing the requested actions. Where the request does not contain information about the overall goal or plan, the agent must base its decision to cooperate on the motivational value of these actions. Each action is considered in turn and assessed according to the motivational value associated with it, to decide whether to cooperate for that action. Those actions that are of motivational value (assuming the requesting agent is trusted) are candidates for cooperation, and are considered further to check that they do not conflict with existing intentions (as described below). Actions that are not of motivational value are rejected, and assistance is not offered. We formalise this below, where we define the function *considerFurtherContribution*, which takes a contribution, the identifier of the requesting agent, and a cooperation threshold and returns true if the requesting agent is sufficiently trusted, and the contribution is of positive motivational value.

ConsiderFurtherContribution

$\exists Agent$

considerFurtherContribution : *Contribution* \rightarrow *AgentID* $\rightarrow \mathbb{R} \rightarrow bool$

$\forall c : Contribution; requester : AgentID; t : \mathbb{R} \bullet$

considerFurtherContribution *c* *requester* *t* = true

$\Leftrightarrow consideredTrusted\ requester\ (extractAllModels\ beliefs)\ t =$
true $\wedge (\exists m : Motivation \mid m \in motivations \bullet$
mvContribution *m* *c* > 0) \wedge

considerFurtherContribution *c* *requester* *t* = false

$\Leftrightarrow consideredTrusted\ requester\ (extractAllModels\ beliefs)\ t =$
false $\vee (\forall m : Motivation \mid m \in motivations \bullet$
mvContribution *m* *c* $\leq 0)$

7.9.3 Commitment to Goals

A request for assistance may also include details of the overall goal in addition to the set of actions for which cooperation is required. It is possible that this goal may not be in the interests of the agent, so it must consider the motivational value that would arise from achieving it in addition to that associated with the actions it is requested to perform in its pursuit. Each action is considered in turn, with respect to the goal. If the goal's achievement has no value (or is negative) then the agent will not cooperate, unless the positive effects of performing the actions outweigh the negative influence of the goal. Conversely, an agent will agree to perform an action in favour of a goal with a positive value if this outweighs any negative value associated with the action.

Before formalising how to make this decision, recall from Chapter 3 that an agent has a set of mitigation functions that determine the motivational value (i.e. the amount by which the motivation is mitigated) of actions and goals to the motivation. In schema *ConsiderFurtherContributionGoal*, we define a function that embodies the decision about whether to consider further a request to cooperate for a contribution and a goal — if the combined motivational value of the contribution and the goal is positive, then the agent must consider the request further.

ConsiderFurtherContributionGoal

$\exists \text{Agent}$

considerFurtherContributionGoal : *Contribution* \rightarrow *Goal* \rightarrow *AgentID* $\rightarrow \mathbb{R}$
 $\rightarrow \text{bool}$

$\forall c : \text{Contribution}; g : \text{Goal}; \text{requester} : \text{AgentID}; t : \mathbb{R} \bullet$
considerFurtherContributionGoal *c g requester* *t* = true
 $\Leftrightarrow \text{consideredTrusted requester (extractAllModels beliefs) } t =$
true $\wedge (\exists m : \text{Motivation} \mid m \in \text{motivations} \bullet$
mvContribution *m c* > 0) $\wedge (\exists m : \text{Motivation} \mid$
m $\in \text{motivations} \bullet \text{mitigation } m g > 0) \wedge$
considerFurtherContributionGoal *c g requester* *t* = false
 $\Leftrightarrow \text{consideredTrusted requester (extractAllModels beliefs) } t =$
false $\vee (\forall m : \text{Motivation} \mid m \in \text{motivations} \bullet$
mvContribution *m c* $\leq 0) \wedge (\forall m : \text{Motivation} \mid$
m $\in \text{motivations} \bullet \text{mitigation } m g \leq 0)$

7.9.4 Commitment to Plans

The final type of request for assistance comprises the annotated plan for which cooperation is needed and, to decide whether to cooperate in this case, an agent must assess the value of the plan as a whole. The decision to cooperate is not made for individual actions, rather a choice is made for the plan as a single indivisible whole. Thus, the motivational value of the plan as a whole must be assessed.

The pre-execution assessment method used in plan selection, in which an agent considers all possible plan elaborations cannot be used here because the agent is being asked to cooperate for *another agent's* plan and cannot know how the plan will be elaborated, since elaboration will use plans from *the other agent's plan library*. Therefore, rather than trying to determine the value associated with any subgoals in the plan based on their possible elaboration, we can only consider the subgoals themselves. The motivational value of the request is then calculated simply by summing the values of the overall goal of the plan, the actions in it (to which the agent is assigned), and the value of any subgoals contained in it. If the plan has a positive motivational value associated with it (and the requester is

trusted), then it is considered further for cooperation. Otherwise, cooperation is rejected, and no further consideration is given to the request, as formalised below in the function *considerFurtherPlan*.

<p style="margin: 0;"><i>ConsiderFurtherPlan</i> _____</p> <p style="margin: 0;">$\exists Agent$</p> <p style="margin: 0;"><i>considerFurtherPlan</i> : $Plan \rightarrow AgentID \rightarrow \mathbb{R} \rightarrow bool$</p> <hr style="margin: 5px 0;"/> <p style="margin: 0;">$\forall p : Plan; requester : AgentID; t : \mathbb{R} \bullet$</p> <p style="margin: 0; padding-left: 20px;"><i>considerFurtherPlan</i> p $requester$ $t = true \Leftrightarrow$</p> <p style="margin: 0; padding-left: 40px;"><i>consideredTrusted</i> $requester$ (<i>extractAllModels</i> <i>beliefs</i>) $t =$</p> <p style="margin: 0; padding-left: 60px;">$true \wedge (\exists m : Motivation \mid m \in motivations \bullet$</p> <p style="margin: 0; padding-left: 80px;">$mvPlan$ m $p > 0) \wedge$</p> <p style="margin: 0; padding-left: 20px;"><i>considerFurtherPlan</i> p $requester$ $t = false \Leftrightarrow$</p> <p style="margin: 0; padding-left: 40px;"><i>consideredTrusted</i> $requester$ (<i>extractAllModels</i> <i>beliefs</i>) $t =$</p> <p style="margin: 0; padding-left: 60px;">$false \vee (\forall m : Motivation \mid m \in motivations \bullet$</p> <p style="margin: 0; padding-left: 80px;">$mvPlan$ m $p \leq 0)$</p>

7.10 Generating Full Commitment to Cooperation

When insufficient agents agree to cooperate a new set of agents can be chosen to ask for assistance, or failure can simply be accepted for the goal and plan concerned. In the former case, previously generated nominal commitments to actions may still be used, and only those where no agent agreed to cooperate need be pursued. Reassignment is performed in exactly the same manner as for the initial assignment.

Reassignment, however, raises the question of whether to assign an agent who has already declined one action to another action it is capable of performing. Clearly, this depends on the reason for declining originally. If an action was declined because the overall goal was not sufficiently valued, then reassignment in this way will have the same result, since the overall goal is the same (unless the new action has a significantly larger motivational value, but this unlikely). If the action was declined due to its motivational value, however, reassignment with a different action may be acceptable. Unfortunately, it is not generally possible to determine another's reason for declining without explicit explanation for de-

clining being provided. Thus, we must assume that reassignment of a declining agent to another action is not a valid option without such an explanation. If it is not possible to reassign because, for example, the initiator does not know of any other agents with required capabilities, then it must admit failure.

In principle, an initiating agent might attempt to negotiate assistance from agents that decline. However, negotiation is a large area of research in its own right, and it is not the focus of the work described in this thesis. Nevertheless, the result of any negotiation strategy is ultimately that an agent will agree or disagree, so that the inputs and outputs of this phase remain unchanged if more sophisticated negotiation is introduced. It should therefore be straightforward to incorporate such mechanisms if required for a particular application.

We can specify a response as a tuple comprising an agent identifier, contribution, and a boolean representing whether the response is accepting or declining to cooperate. The function *checkResponses* takes an annotated plan and a set of responses and returns true if sufficient agents have responded with an acceptance message, and false otherwise.

Response == *AgentID* × *Contribution* × *bool*

CheckResponses

∃ *UseActionRequest*

∃ *ExtractInfoJointAction*

checkResponses : *AnnotatedPlan* → ℙ *Response* → *bool*

∀ *ap* : *AnnotatedPlan*; *rs* : ℙ *Response*; *cid* : ℙ(*Contribution* × *AgentID*) |

cid = *extractAgentsPlan ap* • *checkResponses ap rs* = true

⇔ (∀ *c* : *Contribution* | *c* ∈ (*extractContributions cid*) •

(∃ *id* : *AgentID* • (*id, c, true*) ∈ *rs*)) ∧

checkResponses ap rs = false

⇔ (∀ *c* : *Contribution* | *c* ∈ (*extractContributions cid*) •

(∀ *id* : *AgentID* • (*id, c, true*) ∉ *rs*))

If sufficient agents agree to assistance, then a cooperative intention is formed. The function *formFinalPlan* takes an annotated plan, a set of accepting agents, and a set of

agent models and modifies the annotated plan such that only one agent (from the accepting agents) is annotated to each contribution. Where there is more than one agent accepting for a given contribution, the one that is the most trusted is selected. This specification relies on the auxiliary functions, *processStep*, which selects the most trusted agents that agree to cooperate for a plan step (and is defined in Appendix A).

$$\begin{array}{l}
\text{FormFinalPlan} \\
\exists \text{ConstructConfirmationsAuxiliary} \\
\exists \text{UseActionRequest} \\
\text{responses} : \mathbb{P} \text{Response} \\
\text{accepts} : \mathbb{P} \text{AgentID} \\
\text{formFinalPlan} : \text{AnnotatedPlan} \rightarrow \mathbb{P} \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \\
\rightarrow \text{AnnotatedPlan} \\
\text{accepts} = \{r : \text{Response} \mid r \in \text{responses} \wedge \text{Third}(r) = \text{true} \bullet \text{First}(r)\} \\
\forall \text{ap}, \text{ap}' : \text{AnnotatedPlan}; \text{ms} : \mathbb{P} \text{AgentModel} \bullet \\
\text{formFinalPlan ap accepts ms} = \text{ap}' \\
\Leftrightarrow (\forall n : \mathbb{Z} \mid n \leq \#\text{ap.body} \bullet \text{ap}'.\text{body } n = \\
\text{processStep (ap.body } n) \text{ accepts ms})
\end{array}$$

7.11 Cooperative Intentions in the Warehouse Domain

We can illustrate the process of cooperative intention formation by returning to the Warehouse example. Suppose that an agent, *agent1*, has selected a particular plan, *storeLargePlan*, to achieve its goal of storing a box. Before a cooperative intention can be formed, this plan must be annotated; to annotate its plan the agent considers each action step in the plan in turn, and annotates it with the appropriate agents, based on its trust of them. Of course, annotating a contribution to itself avoids the risk associated with cooperation, and so is better from a risk perspective, but it does require the agent to act and so will have an associated cost. Thus, an agent must decide whether to annotate itself to a contribution by balancing the risk and cost. After deciding which contributions in the plan to perform itself the agent goes through the remaining steps, annotating them with the most trusted agent (or set of agents in the case of joint and concurrent actions) that have the required capabilities.

In our example, suppose that `agent1` annotates itself to one of the contributions in the joint actions of picking up, moving, and putting down the box. The remaining contributions in the joint actions are annotated to the most trusted agent, in this case `agent4`. Thus, the resultant annotated plan is as follows.

```

name: storeLargePlan
  achieves: [$(location, [box1, room2]),
            not (holding, [agent1, box1])$]
  preconditions: [(location, [box1, loc1]),
                 (type, [box1, large, type2])]
  body: [
    <goal
      [$(location, [agent1, room1]),
       (location, [agent4, room1])$]>,
    <joint_action
      [<action (liftend, [agent1, box1])=agent1>,
       <action (liftend, [agent4, box1])=agent4>]>,
    <goal
      [$(location, [agent1, room2]),
       (location, [agent4, room2])$]>,
    <joint_action
      [<action (placeend, [agent1, box1])=agent1>,
       <action (placeend, [agent4, box1])=agent4>]>]

```

Once the plan is annotated, the agent sends a request for assistance to `agent4` and forms a nominal commitment. For simplicity, and since we are taking a *closely coupled* view, requests for assistance in the warehouse scenario are based upon communication of the complete annotated plan. Thus, `agent1`'s request message to `agent4` includes the complete plan.

At this stage of execution `agent1` has sent a request, and formed a nominal commitment, and `agent4` must process this request. Now, `agent4` will also have perceived the environment and the box in the delivery area, and so the intensity of its tidiness motivation will also be high. The plan contained in the request mitigates this motivation, and is considered to be of motivational value (the motivational effect outweighs the cost of performing the contributions). If `agent1` is trusted by `agent4`, which according to the trust matrix given earlier it is (with a trust value of 0.96), then it accepts the request, and forms a corresponding nominal commitment.

On receiving the acceptance message the initiating agent (`agent1`) can form a full commitment, since `agent4` is the only other agent involved, and send a confirmation message. When `agent4` receives this confirmation, it too can adopt a full commitment, and execution can begin.

7.12 Commitment Strategies

Cooperation as described here, involves a certain degree of cost and risk. The formation of a cooperative intention has computational costs associated with determining the agents to cooperate with, communicating with them, processing their responses, and forming the actual cooperative intention. In addition, there is an inherent cost in the limiting nature of cooperative intention that constrains future intentions (and therefore actions). In general, the longer the delay between obtaining commitments and relying upon them at execution time, the more time there is for an agent's motivations to change, and so the risk is increased.

When an agent adopts a plan containing an action for which cooperation is sought, it can solicit assistance and initiate the formation of the required cooperative intention

- as soon as the plan is selected, using an *immediate commitment strategy* (ICS), or
- later at execution time, using a *delayed commitment strategy* (DCS).

Both these strategies require an agent to request assistance from others and elicit commitments from them in the form of a cooperative intention, as described earlier. The choice, therefore, is between *when* to perform the tasks of assigning agents to actions, establishing commitments, and so on, rather than *whether* to perform them. If an ICS is chosen, then these tasks are performed at the time of plan adoption, while they are undertaken at execution time with a DCS.

Since plans are typically partial, they may be elaborated with a subplan that requires cooperation. If the parent plan is also cooperative then an agent must already have chosen a

strategy when adopting it. Continually re-evaluating which strategy to use for subplans may be prohibitively expensive when elaborating sophisticated plans, and is in any case unlikely also to provide a different answer. Consequently, the same strategy as for the parent plan is used, which has a lower overhead, leaving an agent more time to act. If a parent plan is not cooperative, but is elaborated with a cooperative subplan, the situation is no different from the general case, requiring a choice that can then be used subsequently.

7.12.1 Minimising Risk and Cost

In choosing between an ICS and a DCS, the aim is to minimise as much as possible both the risk and cost associated with cooperation. However, although the choice of strategy will certainly affect the cost of cooperation, since one strategy may result in the immediate establishment of a cooperative intention while the other may involve several failed attempts before successful establishment, it is not generally possible to determine this cost at the time of choosing, or which strategy offers the least cost. An estimate of the minimal cost can be obtained from the number of contributions in the plan that requires assistance, since assistance must be obtained for each of them. The final cost, however, is likely to be more than this estimate, since not all agents might initially agree to cooperate, and communication errors might occur.

Determining the exact cost of establishing a cooperative intention, requires knowledge of how agents will respond to requests for assistance. Responses are determined by the motivations of the individual agents, however, and since motivations are private any estimates can only be based on observed behaviour. Not only is this likely to be expensive when many agents are involved, due to the resource demands of observing and reasoning about behaviour and motivation, but even with sufficient computational resources the resulting information is likely to be inaccurate. Since there is, therefore, no practical means of determining which strategy will have the least cost, an agent should concentrate on minimising the risk of a plan failing through the actions of others. We now consider the factors that

affect the risk of failure under both an ICS and a DCS.

7.12.2 Choice Factors

When using an ICS, failure is most likely as a result of failing to secure commitments from others at plan adoption time (if they might be obtained at execution time²), or of agents not fulfilling commitments at execution time. The main potential failure point in a DCS arises through failing to secure commitments from others at execution time (possibly after performing part of the plan individually). Thus, the choice between an ICS and a DCS corresponds to a tradeoff between the risk of wasting effort acting on the individual part of the plan only to fail to get assistance subsequently for the cooperative part, and failing to obtain commitments at adoption time when commitment would have been obtained later.

If there are insufficient agents with the relevant capabilities, the plan will certainly fail. Conversely, the more agents with these capabilities, the more likely it is that enough of them will cooperate or will provide adequate cover for those renegeing on their commitments at execution time. Since agents act for motivational benefit, any plan in which they cooperate must also be motivationally valued. Now since motivation, at least in part, is determined by the environment, environmental change also results in motivational change, and an agent that agrees to cooperate at adoption time may not do so by execution time if there is no longer any motivational benefit. Thus, the extent to which the environment is dynamic influences the choice of strategy, and in a dynamic environment it may be better to postpone the establishment of commitments until execution time. Finally, the trustworthiness of others can be used as an indication of the likelihood that their commitments will be fulfilled. Higher trust suggests a greater perceived likelihood of fulfilling commitments. If other agents are generally distrusted, therefore, obtaining commitments at adoption time may be too expensive since they are more likely to renege on them.

²For example, a change in motivations between plan adoption and execution may result in positive motivational value for cooperation, or a conflicting intention at adoption time may no longer exist.

Trust in Choosing a Strategy

At the time of choosing between an ICS and a DCS (the point of plan adoption), it is not known which agents will be requested for assistance, since the plan has yet to be annotated. Consequently, it is not possible to consider only the trustworthiness of the agents whose assistance will be requested. Instead, the trust of *all* capable agents must be considered. Although the trust values may change between choosing between commitment strategies, these changes cannot be predicted and so a decision must be based on the current trust values. In Section 6.4, we discussed assessing the *cooperative rating* for a plan by ordering the capable agents for each action according to trust, and weighting the influence of successively less trusted agents by a corresponding increasing factor. We use this rating here as an estimate of the risk arising from the trust of others.

Environmental Dynamism

Both the number of agents with the requisite capabilities and trustworthiness are easily determined from agent models, but environmental dynamism is less immediately easy to assess. However, the relationship between the changes to the environment and changes to beliefs and motivations points to a solution. On each iteration through its control cycle, an agent perceives the environment and updates its beliefs and motivations to reflect any changes. The degree of dynamism in an agent's beliefs and motivations are therefore both candidates for estimating the environmental dynamism.

The number of beliefs that change at a given time correspond to the number of perceived changes in the environment. Similarly, the number of changes over a period of time characterise the degree of environmental dynamism, so the number of changes in beliefs can be averaged over a period to form an estimate of change per iteration. The length of the period determines the persistence of the effects of peaks and troughs, and can be regarded as the extent of an agent's *environmental memory*. Calculating the degree of dynamism in this manner is requires only minor extra effort, and by concentrating on changes in beliefs

no assumption is made about the nature of motivations of others. Where an agent has no knowledge of the motivations of others, it is better to consider *all* changes in the environment, rather than to assume that if its own motivations changes, then so do those of others.

Domain Dependence

The levels of importance placed on the various factors to be considered in choosing between an ICS and a DCS, however, will vary according to the domain. For example, in a domain where there are few other agents, the number of agents with the required capabilities may be more important than their trust, since if there are only a few agents having the required capabilities, then cooperation must be attempted with them, regardless of their trust (unless they are completely distrusted). Similarly, a highly dynamic environment is more important than agent trust, since trust may change between execution time and adoption time. It is therefore not possible to provide a (computationally realistic) strategy for choosing between an ICS and a DCS, that will give the *best* result in *all* domains. Indeed, even in a specific domain, the degree to which it is dynamic may change over time, and individuals may join or leave the system so that the number of agents with a given capability may also change. To address this, an agent must choose dynamically which strategy to use for a particular plan, based on its knowledge of the current situation. This choice is embodied in a decision function that takes relevant factors as input, and which must be instantiated according to the individual agent and its domain.

7.12.3 Strategy Choice

The input to the decision function includes threshold values representing when the environment is considered too dynamic to use an ICS, or there are too few trusted agents able to assist. In the majority of situations an ICS is suitable for establishing cooperative intention; it is only in the circumstances identified above that a DCS should be used. For this reason, we make an ICS the default choice of strategy, and provide an agent with a decision function

for choosing whether to override this default with a choice to use a DCS. A consequence of this is that since an ICS corresponds to the approach taken in the majority of existing models of cooperation, our model can be viewed as extending the general solution to cope with extreme conditions. We can now give the instantiation of this decision function, as follows, where dT and rT are the dynamism and risk (arising from trust) thresholds respectively. From the pre-execution assessment of its plan library an agent has an estimate of the risk arising from its trust of others, and if this value is less than the risk threshold, or if the perceived environmental dynamism exceeds the dynamism threshold, then a DCS should be used. We formalise this below in the function *useDCS*.

$$\begin{array}{l}
 \text{--- } \textit{UseDCS} \text{ ---} \\
 \hline
 \textit{useDCS} : \mathbb{P} \textit{AgentID} \rightarrow \mathbb{P} \textit{AgentModel} \rightarrow \textit{Plan} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\
 \rightarrow \mathbb{R} \rightarrow \textit{bool} \\
 \hline
 \forall \textit{capable} : \mathbb{P} \textit{AgentID}; \textit{ms} : \mathbb{P} \textit{AgentModel}; \\
 \textit{p} : \textit{Plan}; \textit{dynamism}, \textit{dT}, \textit{rT}, \textit{w}_s, \textit{w}_c : \mathbb{R} \bullet \\
 \textit{useDCS} \textit{capable} \textit{ms} \textit{p} \textit{dynamism} \textit{dT} \textit{rT} \textit{w}_s \textit{w}_c = \textit{true} \\
 \Leftrightarrow (\textit{quality} \textit{p} \textit{ms} \textit{w}_s \textit{w}_c) > \textit{rT} \vee \textit{dynamism} > \textit{dT} \wedge \\
 \textit{useDCS} \textit{capable} \textit{ms} \textit{p} \textit{dynamism} \textit{dT} \textit{rT} \textit{w}_s \textit{w}_c = \textit{false} \\
 \Leftrightarrow (\textit{quality} \textit{p} \textit{ms} \textit{w}_s \textit{w}_c) \leq \textit{rT} \vee \textit{dynamism} \leq \textit{dT}
 \end{array}$$

Note that if the number of capable agents is less than is required, then both strategies fail immediately. The effectiveness of this mechanism relies on judicious choice of the dynamism and risk thresholds. However, as the nature of the system may change over time, the ideal threshold values may also change, and to cope with such environmental change, an agent might dynamically modify these thresholds. For example, if an ICS is used and agents frequently break commitments, thresholds can be increased. Similarly, if a DCS is used without obtaining commitment at execution time, thresholds can be lowered to encourage use of an ICS. We do not, however, consider such matters further here.

7.13 Intention Execution

After forming a cooperative intention, the final stage of cooperation is for the agents to actually execute the plan to which they are committed. Each agent is committed to performing a particular contribution, either individually or as part of a joint or concurrent action. Now, in order for the execution of a plan to be successful, the steps in it must be performed in the correct order. For an individual agent executing an individual plan by itself, ensuring the steps in the plan are performed in order is trivial — the agent simply works through the plan step by step, performing each action as it is reached. When a group of agents act together towards the achievement of some goal, however, their individual contributions must be coordinated. Successful execution of a cooperative plan requires agents to perform their contributions according to a particular ordering, namely that specified by the plan.

We do not discuss how to achieve such ordering, since it is beyond the scope of this thesis. However, Kinny *et al.* offer a simple solution in their model of Planned Team Activity [55], which we adopt for completeness. Their solution is to require that the agent executing a given action informs the agent of the following action when execution is successfully performed. Correspondingly, the agent of the following action must not perform its contribution until it is informed that the previous action has been completed. In the case of joint and concurrent actions we extend this solution such that the *set* of agents performing joint or concurrent actions must inform the agent(s) of the following action when their contributions are complete. Similarly, each agent involved in the execution of the following action must wait until it has been informed of completion by *each* of the agents performing a contribution in the previous action. Action ordering can be achieved, therefore, by inserting appropriate communication and waiting actions into the plan prior to its execution. More details of this approach to constraining the ordering of actions are given in Appendix B where we describe our implementation of the framework.

At execution time, when a subgoal is reached it must be elaborated by selecting an appropriate plan, and incorporating it into the agent's existing intention. The mechanism

for plan selection described in Chapter 6 can also be used at execution time, along with the mechanisms for intention adoption described in this chapter. However, our earlier consideration of intention adoption was with respect to adopting a plan for a goal, rather than adopting a *subplan* for a *subgoal*. Although many of the issues described earlier are the same, the manner in which the intention adoption mechanisms are used will vary depending on whether the intention concerned is individual or cooperative. In particular, when elaborating a plan for which there already exists a cooperative intention, the intentions and motivations of the agents concerned must be considered.

The elaboration of an individual plan is straightforward, since there is no pre-existing cooperative intention. If the selected subplan is also individual, then it is simply inserted into the intention as described in Chapter 4. Alternatively, if the chosen plan requires cooperation, the agent must decide when to solicit commitments from others towards cooperation — it must choose between using an ICS or a DCS. If a DCS is chosen then the formation of a cooperative intention is delayed until execution reaches the cooperative part of the plan, and adoption occurs as for an individual subplan. If an immediate commitment strategy is chosen, then the agent attempts to establish a cooperative intention towards the subplan, before adding it to its intentions. The mechanisms used for cooperative intention establishment and adoption are as described in this chapter.

7.13.1 Cooperative Plan Elaboration

As with individual plans, elaborating a subgoal in a cooperative plan involves selecting an appropriate plan, and forming a corresponding intention. However, for a cooperative plan there is typically already a group of agents having a cooperative intention. Only if the plan was adopted using a DCS, and no cooperative actions have yet been executed, will there not be a such a group of agents. In this case, the plan can be elaborated as though it were an individual plan, and a DCS must be used for the subplan (since the strategy used for a parent plan should be used for its subplans). There are two main options for plan selection

for a cooperative plan where a cooperative intention already exists:

- elaboration can be centralised and the task of plan selection given to a particular individual, or
- it can be decentralised and the group can form a plan together.

In this thesis, we are concerned, with cooperation arising from individual agents, rather than individual action resulting from some social mental state. In other words, our focus is on the development of a framework for cooperation resulting from an individual agent wishing to gain assistance for a particular goal. Decentralised plan formation requires some mechanism through which members of the group offer potential plans, or assist in the construction of a plan. Some planning ability, is required, on behalf of the agents involved, along with some form of negotiation. However, we are concerned with agents whose planning is restricted to selecting from a predefined plan library, rather than planning from first principles. Decentralised planning is addressed elsewhere (for example in the notion of Shared Plans [42, 43]) and, although we do not give it further consideration here, it would be a relatively simple extension to our framework to incorporate one of the group planning mechanisms described elsewhere.

Therefore, we consider a centralised approach to planning in which the task of plan selection is assigned to an individual member of the group. This approach is analogous to that taken by Kinny *et al.* in their work on Planned Team Activity [55]. There are two options in centralised planning: assign the role of plan selection to the initiating agent, or assign it to some other agent. Since cooperation, in our framework, arises from the initiator's desire for assistance in achieving a goal, we assign the role of plan elaboration to the initiating agent. There are, however, two situations in which the initiator might prefer plan selection to be performed by an agent other than itself. Firstly, if there is some other agent with more knowledge of the problem that is better placed to choose an appropriate plan, or secondly, if the initiator has no applicable plans for the subgoal or those that it does have are of zero or negative motivational value, then plan elaboration should be assigned to

another agent. We formalise this below in the following schema, by defining the function *delegateElaboration* which returns true if the elaboration of a particular subgoal should be assigned to another agent.

$$\begin{array}{l}
 \text{---} \underline{\text{DelegateElaboration}} \text{---} \\
 \exists \text{Agent} \\
 \text{delegateElaboration} : \text{Goal} \rightarrow \text{bool} \\
 \hline
 \forall \text{subgoal} : \text{Goal} \bullet \text{delegateElaboration subgoal} = \text{true} \Leftrightarrow \\
 \quad (\text{planSetForGoal subgoal beliefs planLibrary}) = \emptyset \vee \\
 \quad (\forall p : \text{Plan}; m : \text{Motivation} \mid m \in \text{motivations} \wedge \\
 \quad \quad p \in (\text{planSetForGoal subgoal beliefs planLibrary}) \bullet \\
 \quad \quad \text{mvPlan } m \ p \leq 0) \\
 \forall \text{subgoal} : \text{Goal} \bullet \text{delegateElaboration subgoal} = \text{false} \Leftrightarrow \\
 \quad (\text{planSetForGoal subgoal beliefs planLibrary}) \neq \emptyset \wedge \\
 \quad (\exists p : \text{Plan}; m : \text{Motivation} \mid m \in \text{motivations} \wedge \\
 \quad \quad p \in (\text{planSetForGoal subgoal beliefs planLibrary}) \bullet \\
 \quad \quad \text{mvPlan } m \ p \geq 0)
 \end{array}$$

A group's cooperative intention is relative to the initiating agent's goal, since it is the reason for their cooperation. Therefore, even if another agent selects a plan, that plan must be accepted by the initiating agent and it must not conflict with the initiator's intentions, and must be of motivational value. The initiating agent, therefore, has a supervisory role in the process of plan elaboration. Where the initiating agent is responsible for selecting the plan to be used in elaboration, it must find the plan acceptable, otherwise it would not have chosen it. However, where plan selection is performed through another method, some check is need to ensure the initiator accepts the selected plan.

7.13.2 Centralised Elaboration

Using the mechanisms described in Chapter 6 the initiating agent selects the best plan for the subgoal. There are two main options for adopting this plan. Firstly, the plan could be communicated to the group, and adopted if it is of sufficient motivational value to each member of the group. This mechanism, however, requires that the group are informed of

the complete plan which, as described earlier in this chapter, is not necessarily the best approach. Secondly, the adoption of the subplan could be achieved through the application of the mechanisms for the adoption of a (parent) plan. Recall that this involves annotating the plan with agents to ask for assistance, requesting assistance, and then forming a commitment once assistance is offered. The advantage of using these mechanisms is that the group is not informed of the complete plan where it is not appropriate to do so, and we therefore use this latter option for the adoption of the selected plan. If the initiating agent has no appropriate plan, then it can ask for assistance, in the form of an appropriate plan being offered.

There is, however, one key difference between adopting a subplan of a cooperative intention, and adopting a plan for a goal, namely, that for the intention to be a cooperative intention some group of agents must already be committed to its achievement, or at the very least to performing certain actions in its favour. The implication of this is that these agents may be more likely to agree to assist in the performance of the subplan. If a given member of the cooperative intention was informed of the goal for which assistance was requested, then it is likely that the goal has motivational value, and so the subgoal may also have motivational value. However, if the agent was only asked to perform a particular action, and was not informed of the goal then its existing commitment does not indicate that the request is likely to be motivationally valuable. It does, however, indicate that the agent considered the initiating agent sufficiently trusted, and will not decline to cooperate on the basis of trust (unless its trust has changed). Therefore, annotating with agents that are already committed to the cooperative intention of which the subgoal is a part is, in general, likely to arrive at a commitment sooner than annotating with agents with whom there is no pre-existing commitment. This can be factored into the process of adopting a subplan if, at the annotation stage, the initiator first tries to annotate the plan with agents that are already part of the cooperative intention. Only if a commitment has not been established, and there are no more agents to ask that are part of the cooperative intention, are other agents asked.

7.14 Summary

In this chapter we have described a set of procedures through which an agent can establish a cooperative intention towards a plan, such that agents will only form and retain a cooperative intention if they expect to gain motivational value from doing so. Additionally, should an agent's motivations change in such a way as to make the cooperative intention no longer of motivational value, then the commitment is dissolved, keeping all agents informed.

The first stage in the formation of a cooperative intention is for the initiating agent to annotate its chosen plan with the agents whose assistance it will request. Plan annotation can be minimal or redundant as determined by the importance of the agent's goal. By default the minimal annotation strategy is used and the redundant approach is only used if the motivational value associated with the goal is over a particular threshold. In cases where the redundant approach is used, the initiating agent requests assistance from several agents for each action, and each agent that agrees forms a nominal commitment to perform that action. Once the initiator has received responses from the requested agents, one particular agent (the most trusted) is chosen for each contribution, and the others are informed that their assistance is no longer required. These agents then drop their nominal commitment, and reduce their trust of the initiator, since they have unnecessarily constrained their actions though the nominal commitment adopted on its behalf.

In general, cooperative intentions are discharged successfully if one of the agents involved changes its motivations such that the cooperative activity is no longer of value (or its commitment is dropped for some other reason). However, if an agent does not adhere to the appropriate conventions and simply drops its commitment without informing others, then they are left with commitments to a cooperative intention that will not be fulfilled. Until one of the remaining agents discovers that their intention is unachievable, or drops its commitment and informs others, the remaining members of the group will keep their commitment, thereby constraining their actions unnecessarily. However, provided agents follow the conventions given in Section 5.2 this problem is avoided.

The formation and maintenance of a cooperative intention requires agents to communicate certain information to each other. In order for communication to take place the agents concerned must have a commonly understood *agent communication language* (ACL). There are several existing ACLs such as KQML [47] and the FIPA ACL [33], and consideration of such languages is beyond the scope of this thesis. Instead, an instantiation of our framework can use one of the existing ACLs. All that is required is that agents are able to *request* assistance or information from another, *accept* a request for assistance, *decline* a request for assistance, and *inform* another of something.

Related Work

The process for establishing cooperative intention outlined above is related to the work of Cohen and Levesque, Kinny *et al.*, and Wooldridge and Jennings, and since our view of cooperative intention is based on their work it is useful to compare it with our approach. All of these models aim to establish some form of group commitment to a goal and, with the exception of Cohen and Levesque's work, to eventually obtain a commitment from agents to performing particular actions. Each of these approaches is based on the notion of some initiating agent having a goal for which it desires assistance, and this agent then requesting the cooperation of others. The details of how this is performed, however, are different in each approach.

Recall that Cohen and Levesque's notion of a group's commitment (or joint intention in their terminology) is based on the members committing to a particular goal, rather than to a goal and a specific *plan* to achieve it. As we discussed in Section 5.2, this view of joint commitment is insufficient for cooperation, because there is no requirement for agents to be committed to performing compatible plans to achieve their goal. Moreover, Cohen and Levesque do not consider the *motivational* reasons an agent might have for entering into a cooperative interaction.

Planned Team Activity (PTA) is a related approach (introduced in Chapter 2) to cooper-

ation which, unlike Cohen and Levesque's model, is concerned with obtaining commitment towards a specific plan. There are a number of differences between our approach and that of PTA. The most significant difference is that our model is based on the assumption that agents are *motivated*, whereas in PTA the issue of autonomy (and in particular motivation) is not addressed. Although Kinny *et al.* state that in deciding whether to agree to cooperate or not, agents should check their current commitments and preferences, they are not concerned with how this happens, or the form such preferences might take, whereas for us cooperation *must* be motivated on the behalf of the agents concerned. Similarly, PTA does not consider the potential risk of interacting with others, which we address through the notion of trust.

The final related work that we identify here is Wooldridge and Jennings' formalisation of the cooperative problem solving process [102, 104], in which cooperation is divided into the four stages of recognition, team formation, plan formation, and team action as described in Sections 2.7 and 5.3. Their work is especially relevant since we adopt an extended version of their notion of cooperative intention. However, their formalisation is an abstract model of cooperation and, as they themselves recognise, is idealised in the sense that it provides a top-level specification for a system, requiring more detail before it can be implemented. Our approach is based on their model, and we view it as instantiating some of the details that were previously left abstract. Their model, although concerned with autonomous agents, does not consider why an agent might enter into a cooperative intention, i.e. the *reasons* for doing so, and we address this through the introduction of motivational value.

Chapter 8

Conclusions

8.1 Introduction

Cooperation is fundamental to the operation of multi-agent systems in which a collection of autonomous agents interact to achieve their goals. Existing models of cooperation, however, are limited in that they typically do not consider the need for cooperation to be *motivated* on behalf of the agents involved, nor do they provide agents with a means to manage the risk involved in interacting with others. Of the few models that do consider the notion of motivation or of risk, they typically focus on one of these, leaving the issues arising from the (potentially conflicting) influences of each unexplored.

Those models that take the view that an agent should have some *reason*, or motivation, to cooperate tend not to give details of the form such a reason might take; instead they simply assume that agents check their preferences before cooperating, without defining what these preferences are, or how they operate (for example, [55, 104]). Similar limitations arise in work where the risk from cooperation is considered, in that either insufficient detail is given about the mechanisms involved [54], such as how risk is assessed and how it can be used in decision making, or the agents concerned are not motivated, and their decisions regarding cooperation are based primarily on an assessment of risk without consideration of

the potential benefits to be obtained from different (possibly risky) courses of action [69]. For any given goal, there are often several courses of action to achieve it, from which an agent must select the best. An agent may be faced with a choice about whether or not to achieve its goal through cooperation, and although achieving a goal cooperatively may be of a lower cost than achieving it alone, it is likely to have a higher risk of failure, due to the reliance on the actions others.

In our view, as argued in Chapter 6, the choice between courses of action in a cooperative environment, should consider both the cost and the risk associated with these options, and make an appropriate trade-off between them. Similarly, where an agent is asked to assist in the achievement of another's goal, it should consider both the motivational benefit of doing so, and the potential risk of failure in deciding whether to accede to the request. In this thesis we have described how an agent can make such judgements, and the processes that are involved in cooperating if a cooperative course of action is chosen. We give more detail of these contributions in the following section, and in Section 8.3 describe the relation of our framework to existing work. In Section 8.4 we discuss the limitations of our approach, and provide pointers to potential future work, and finally, in Section 8.5 we conclude this thesis.

8.2 Contributions

The contributions made in this thesis can be divided into three significant areas: the construction of a framework for cooperation, and the development of detailed models of plan selection and cooperative intention formation within that framework, as we describe in this section.

Framework for motivated cooperation We have presented a framework for *motivated* cooperation, in which an agent's motivations guide its behaviour, and govern any interactions it might have with others. The framework defines the form that cooper-

ation takes and the decision mechanisms that lead to it; where appropriate, however, the details are left open allowing it to be tailored and instantiated in a particular model for a specific domain.

Our framework is based on the notions of BDI [3], which are widely used to provide a balance between reactivity and deliberation. The BDI architecture alone, though, does not account for the *reasons* an agent might have for adopting particular goals, or choosing to cooperate with others. However, the additional mental component of motivation provides a suitable means for accounting for these reasons, as argued by Luck and d’Inverno [61]. Therefore, in our framework we extend the notions of BDI to include *motivations*, and define the SENARA motivated agent architecture. Motivations give flexibility, especially in areas such as choosing a course of action when there are multiple conflicting options. Similarly, motivations provide a means through which agents can choose when to cooperate, i.e. when to ask for assistance, and when to offer it. Existing work has not considered motivations in this context, and in this thesis we have accounted for the roles that they play.

Our framework also provides a method for dealing with the risk that arises where autonomous agents cooperate. In particular, we describe how the notion of trust can be used by an agent to manage the risk that arises from cooperation.

Plan Selection In our view, plan selection is a fundamental component of cooperation, since it embodies the choice of whether to cooperate or not. The process of plan selection addresses the problem of how to choose between (and elaborate) plans in a cooperative environment. Existing models of cooperation are limited, however, and tend to focus on situations where cooperation is *necessary*, and do not consider *optional* cooperation (or at least, not in sufficient detail). We present an approach to plan selection that is appropriate in situations where cooperation is necessary or optional, and we describe how a combination of standard planning heuristics and an assessment of the risk associated with each plan can be used to choose a plan to pursue. The notion of *trust* is introduced and, along with knowledge of agents’

capabilities, is used to determine the risk associated with a plan.

Cooperative Intention Formation The process of cooperative intention formation is concerned with forming a commitment between a group of agents to achieving a particular goal cooperatively, through the execution of a specific plan. In our view this commitment must be *motivated* on behalf of the agents involved, that is to say, an agent will only cooperate if it is in its own interest to do so, and this aspect of cooperation is also typically not considered in existing work. There are two sides to cooperative intention formation in our framework: soliciting commitment and offering it.

- When an agent selects a plan that requires cooperation for its successful execution, it solicits commitment from others towards assisting in the execution of the plan. We describe how an agent can assess others in terms of their capabilities and, more importantly, their trustworthiness in order to minimise the risk of failure at plan execution time.
- Those agents whose assistance is requested, must decide whether to cooperate or not. We propose a mechanism through which such agents can choose, based on an assessment of the value of cooperation in motivational terms, and of the risk of cooperating in terms of the trustworthiness of the other agents involved.

Our framework also defines the procedure through which a cooperative intention is formed, where sufficient agents offer assistance. In order to be applicable in dynamic environments, agents are given a choice about when to form such a commitment: at plan selection time, or at plan execution time.

8.3 Relation to Existing Work

The contributions described above correspond to the key areas in which this thesis addresses limitations in existing work on cooperation. There are, however, a number of other areas

that this thesis is related to, and in which it extends existing work, and we consider these in this section.

Our framework is based on the SENARA agent architecture which extends the BDI model to include the additional mental component of *motivation*, based on Luck and d’Inverno’s work on agent autonomy [61, 62]. Luck and d’Inverno, however, are concerned with the development of a general framework for autonomous agency based on the notion of motivation, rather than with the development of a specific agent architecture, and therefore some details of their model are left abstract. For example, they do not specify precisely how to instantiate the mechanisms for assessing the motivational value of generating, satisfying, and removing goals. SENARA, however, is a complete implemented architecture and can be seen as an instantiation of their model, in which we provide the details that were previously left abstract.

Together with motivation, *trust* is a fundamental component of our framework, providing agents with a means to manage the risk associated with interacting with others. Our view of trust is derived from Marsh’s work [67] and we incorporate it into our framework of cooperation. Unlike us, however, Marsh is concerned with the issues surrounding trust itself, rather than with its relation to cooperation as a whole or with the development of a model of cooperation using the notion trust, and in that sense our work can be seen as providing the details that are needed to incorporate this notion of trust into a framework for cooperation.

Our view of cooperation is loosely based on Wooldridge and Jennings’ four stage approach comprising: recognition, group formation, deciding on a course of action, and group action. Their model is abstract and intended as a high-level specification rather than a complete model, and therefore requires some of the details to be instantiated before it can be used practically in an implemented system. They also recognise that although the four stages in their model are presented as being sequential, in practice they may not occur strictly in the order they describe. Indeed, as discussed in Chapter 5 this is the key differ-

ence between our model and theirs; in our approach an individual agent selects a plan that requires cooperation and then seeks assistance, whilst in their approach an agent recognises the potential for cooperation, seeks assistance, and then the agents as a group form a plan. Plan selection in our model is analogous to deciding on a course of action, and recognising whether this requires cooperation. If so, then a cooperative intention is formed (group formation in Wooldridge and Jennings' terms), and this intention can then be executed (i.e. group action).

There are other existing models of cooperation, such as STEAM [98] and Planned Team Activity [55], but these also do not consider either trust or motivations, and in this respect our framework can again be seen as extending existing models such as these.

8.4 Limitations and Future Work

In the development of our framework we have made certain assumptions and decisions, which not only shape the framework, but also give rise to limitations within it, which in turn indicate areas of potential future work. In particular we concentrate on *motivated* agents, as embodied by the SENARA architecture, and we assume that these agents have knowledge of others' capabilities and trustworthiness. Trust, in particular, is fundamental to our view of cooperation, and we assume that agents have appropriate estimates of the trustworthiness of others, which are used to determine when to cooperate, and with whom. While in general this assumption, and our use of these trust values, is effective, there are certain aspects in which the framework is limited, in particular with respect to updating trust values, plan assessment (for selecting between plans), and plan annotation (for choosing agents to request assistance from).

In this thesis, we are concerned with the use of trust in making decisions about cooperation, rather than with obtaining and maintaining trust values in themselves, and we use a simple procedure for updating trust values after interactions. However, this procedure is

limited in that once an agent becomes distrusted to the extreme in the eyes of another it remains distrusted, and it cannot regain trust. Typically, this issue does not arise since it only occurs with extreme distrust, and agents that are distrusted to the extreme are unlikely to change their nature. However, there are exceptional cases in which this is a problem, such as if an agent experiences some temporary difficulty that causes it to renege on its commitments, and become distrusted. If this difficulty is later addressed, there is no mechanism through which the agent can regain the trust of others. Thus, future work might consider the development of a more sophisticated mechanism for updating trust, such that agents can regain trust.

The procedures for plan assessment and plan annotation are the two most computationally expensive areas of the framework, and may cause problems with large numbers of agents. The computational cost of plan assessment to an agent is proportional to the number of plans in its library and the number of agent models, while the cost of plan annotation is proportional to the number of agents. Both areas require further investigation, with the aim of improving their efficiency.

The initial assessment of plans is performed off-line prior to execution, and so its cost does not directly affect an agent. However, the environment changes over time and with interactions, and therefore so does an agent's trust of others, requiring new judgements to be made about appropriate plans, i.e. re-assessment of the plan library is needed. Such re-assessment, however, despite being performed at run-time, can be carried out while the agent is idle, thereby reducing the effect of the computational cost on the agent's operation. Plan annotation, on the other hand, must be performed at execution time, since before an agent can request assistance it must annotate its chosen plan. Therefore, although plan assessment is more costly than plan annotation, it is plan annotation that is most significant, since an agent has no choice about when to perform it, and cannot wait until it is idle.

In our framework, decisions about cooperation are made on the basis of motivations and trust, and in particular, the motivational value of a given course of action to the agent

making the choice, and the expected risk. Where an agent solicits assistance for a plan, others' decisions about whether to cooperate are based on their motivations and the degree to which they trust the requester. Now, as discussed in Chapter 6, we do not consider others' motivations as a factor in decisions about cooperation, since an agent does not have direct access to information about them. However, if the framework is extended to incorporate learning based on observation, then others' motivations could be factored into the decision making process. If an agent is observed performing a given action, then that action must be of motivational benefit to it, either indirectly through the goal for which the action is performed, or directly from the action itself. Over time, an agent can build up a picture of the actions that others are seen to perform, and can use this to estimate the likelihood of particular agents agreeing to cooperate for the actions in a plan. Similarly, an agent's estimate of others' trustworthiness can be improved by observation. For example, if an agent is seen to renege on its commitments to another, then the trust associated with it might be decreased, since it is observed to be untrustworthy.

There are certain aspects that are commonly associated with the cooperative process that our framework does not consider, in particular the issues relating to negotiation and group planning. When soliciting assistance for a goal an agent might enter into negotiation with another in order to persuade it to offer its assistance. In our framework, when a group of agents that are executing a cooperative plan reach a subgoal in that plan, the elaboration of the subgoal is performed by an individual agent, using its individual plan library. Multi-agent planning [41, 42] offers an alternative approach where a group of agents construct a plan together, by pooling their knowledge about how to achieve a goal. Both of these areas provide scope for incorporating other existing work into our framework.

8.5 Summary

Cooperation is fundamental to multi-agent systems, and is the building block that allows a loose collection of individuals to act together and achieve goals that might otherwise be

unachievable. This thesis can be seen as extending existing work in a number of areas, as follows. In particular,

- the BDI approach has been extended by giving agents motivations, which provide the reasons for their behaviour, and allows them to be more flexible,
- we have instantiated previous work on motivation in a complete framework for cooperation, and accounted for the roles that motivations play in the cooperative process,
- cooperation with autonomous agents involves an inherent risk, and we have extended previous work on trust to provide means for managing this risk, and
- we have addressed some of the limitations in existing models of cooperation, in particular with respect to motivation and trust.

We focus in particular on the problems of risk, flexibility, and dynamism, and we use the notions of trust and motivation to address the issues that arise, and we do not address certain other aspects, such as negotiation and multi-agent planning. In that sense, while the work has moved the state of the art forwards a substantial amount, just as in any significant and valuable endeavour, the path ahead offers opportunity for further work.

References

- [1] L. P. Beaudoin and A. Sloman. A study of motive processing and attention. In A. Sloman, D. Hogg, G. Humphreys, D. Partridge, and A. Ramsay, editors, *Prospects for Artificial Intelligence*, pages 229–238, Amsterdam, 1993. IOS Press.
- [2] M. E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.
- [3] M. E. Bratman. What is intention? In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 15–32. MIT Press, 1990.
- [4] M. E. Bratman. Shared cooperative activity. *Philosophical Review*, 101(2):327–341, April 1992.
- [5] M. E. Bratman, D. Israel, and M. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.
- [6] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.
- [7] R. A. Brooks. Elephants don't play chess. In P. Maes, editor, *Designing Autonomous Agents*. MIT Press, 1990.
- [8] R. A. Brooks. Intelligence without reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991.

- [9] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.
- [10] C. Castelfranchi. Social power. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I.: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, pages 49–62. Elsevier Science Publishers B.V., 1990.
- [11] C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Proceedings of the First International Workshop on Agent Theories, Architectures and Languages (ATAL-94)*, pages 56–70. Springer-Verlag, 1995.
- [12] C. Castelfranchi and R. Conte. Distributed artificial intelligence and social science: Critical issues. In G. M. P. O’Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 527–542. John Wiley & Sons, 1996.
- [13] C. Castelfranchi and R. Falcone. Principles of trust for MAS: Cognitive anatomy, social importance, and quantification. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 72–79, Paris, France, 1998.
- [14] C. Castelfranchi, M. Miceli, and A. Cesta. Dependence relations among autonomous agents. In E. Werner and Y. Demazeau, editors, *Decentralized A.I. 3: Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 215–227. Elsevier Science Publishers B.V., 1992.
- [15] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
- [16] P. R. Cohen and H. J. Levesque. Persistence, intention, and commitment. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 33–69. MIT Press, 1990.

- [17] P. R. Cohen and H. J. Levesque. Confirmations and joint action. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 951–957, Sydney, Australia, 1991.
- [18] P. R. Cohen and H. J. Levesque. Teamwork. *Nou̇s*, 25:487–512, 1991.
- [19] P. R. Cohen, H. J. Levesque, and I. Smith. On team formation. In J. Hintikka and R. Tuomela, editors, *Contemporary Action Theory*. Synthese, 1997.
- [20] R. Conte, M. Miceli, and C. Castelfranchi. Limits and levels of cooperation: Disentangling various types of prosocial interaction. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I. 2: Proceedings of the Second European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-90)*, pages 147–157. Elsevier Science Publishers B.V., 1990.
- [21] I. Craig. *Formal Specification of Advanced AI Architectures*. Ellis Horwood, 1991.
- [22] D. C. Dennett. *Brainstorms: Philosophical essays on mind and psychology*. Harvester Press, Hassocks, Sussex, 1978.
- [23] M. Deutsch. Cooperation and trust: Some theoretical notes. In M. R. Jones, editor, *Nebraska Symposium on Motivation*, pages 275–319. University of Nebraska Press, 1962.
- [24] M. d’Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In M. P. Singh, A. Rao, and M. J. Wooldridge, editors, *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages (ATAL-97)*, pages 155–176. Springer-Verlag, 1998.
- [25] M. d’Inverno and M. Luck. Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):233–260, 1998.
- [26] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

- [27] A. F. Dragoni. A model for belief revision in a multi-agent environment. In E. Werner and Y. Demazeau, editors, *Decentralized A.I. 3: Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 103–112. Elsevier Science Publishers B.V., 1992.
- [28] E. H. Durfee. Blissful ignorance: Knowing just enough to coordinate well. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 406–413, San Francisco, CA, 1995.
- [29] I. A. Ferguson. *TouringMachines: An architecture for dynamic, rational, mobile agents*. PhD thesis, University of Cambridge, November 1992. Technical Report No. 273.
- [30] I. A. Ferguson. Integrated control and coordinated behaviour: A case for agent models. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Proceedings of the First International Workshop on Agent Theories, Architectures and Languages (ATAL-94)*, pages 203–218. Springer-Verlag, 1995.
- [31] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.
- [32] K. Fischer, Müller J. P., and M. Pischel. A pragmatic BDI architecture. In *Intelligent Agents II: Proceedings of the Second International Workshop on Theories, Architectures and Languages (ATAL-95)*, pages 203–218, Wooldridge, M. J. and Müller, J. P. and Tambe, M., 1996. Springer-Verlag.
- [33] Foundation for Intelligent Physical Agents (FIPA). Agent communication language, April 1999.
- [34] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III: Proceedings of the Third International Workshop on Agent*

- Theories, Architectures and Languages (ATAL-96)*, pages 21–35. Springer-Verlag, 1997.
- [35] N. Friedman and J. Y. Halpern. Modeling belief in dynamic systems, Part I: Foundations. *Artificial Intelligence*, 95(2):257–316, 1997.
- [36] J. R. Galliers. Autonomous belief revision and communication. In P Gärdenfors, editor, *Belief Revision*, pages 220–246. Cambridge University Press, 1992.
- [37] D. Gambetta. Can we trust trust? In D. Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*, pages 213–237. Basil Blackwell, 1988.
- [38] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.
- [39] R. Goodwin. Formalizing properties of agents. Technical report, Carnegie Mellon University, 1993.
- [40] N. Griffiths and M. Luck. Cooperative plan selection through trust. In F. J. Garjjo and M. Boman, editors, *Multi-Agent System Engineering: Proceedings of the Ninth European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99)*. Springer-Verlag, 1999.
- [41] B. Grosz. Collaborative systems. *AI Magazine*, 17(2):67–85, 1996.
- [42] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–358, 1996.
- [43] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, 1990.

- [44] A. Haddadi and K. Sundermeyer. Belief-Desire-Intention agent architectures. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 169–185. John Wiley & Sons, 1996.
- [45] J. R. P. Halperin. Machine motivation. In J.-A. Meyer and H. Roitblat, editors, *From Animals to Animats, Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 213–221. MIT Press, 1991.
- [46] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the Association for Computing Machinery*, 37(3):549–587, 1990.
- [47] ARPA Knowledge Sharing Initiative. Specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group working paper, July 1993.
- [48] N. R. Jennings. On being responsible. In E. Werner and Y. Demazeau, editors, *Decentralized A.I. 3: Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 93–102. Elsevier Science Publishers B.V., 1992.
- [49] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *Knowledge Engineering Review*, 8(3):223–250, 1993.
- [50] N. R. Jennings. Specification and implementation of a Belief-Desire-Joint-Intention architecture for collaborative problem solving. *International Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.
- [51] N. R. Jennings. *Cooperation in industrial multi-agent systems*, volume 43 of *World Scientific series in computer science*. World Scientific, Singapore, 1994.
- [52] N. R. Jennings and E. H. Mamdani. Using joint responsibility to coordinate collaborative problem solving in dynamic environments. In *Proceedings of the Tenth Na-*

- tional Conference on Artificial Intelligence (AAAI-92)*, pages 269–275, San Diego, CA, 1992.
- [53] N. R. Jennings, K. P. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–36, 1998.
- [54] C. M. Jonker and J. Treur. Formal analysis of models for the dynamics of trust based on experiences. In F. G. Garijo and M. Boman, editors, *Multi-Agent System Engineering: Proceedings of the Ninth European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-99)*, pages 221–231, 1999.
- [55] D. Kinny, M. Ljungberg, A. Rao, E. Sonenberg, G. Tidhar, and E. Werner. Planned team activity. In *Proceedings of the Forth European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-92)*, pages 227–256, 1992.
- [56] J. E. Laird, A. Newell, and P.S. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.
- [57] V. R. Lesser and D. D. Corkill. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. In *AI Magazine*, pages 15–33, Fall 1983.
- [58] H. J. Levesque, P. R. Cohen, and J. H. T. Nunes. On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, Boston, MA, 1990.
- [59] M. Luck. *Motivated Inductive Discovery*. PhD thesis, UCL, University of London, 1993.
- [60] M. Luck. Foundations of multi-agent systems: Issues and directions. *Knowledge Engineering Review*, 12(3):307–308, 1997.

- [61] M. Luck and M. d’Inverno. A formal framework for agency and autonomy. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press/The MIT Press, 1995.
- [62] M. Luck and M. d’Inverno. Structuring a Z specification to provide a formal framework for autonomous agent systems. In J. P. Bowen and M. G. Hinchey, editors, *Proceedings of the Ninth International Conference of Z Users (ZUM-95)*, pages 47–62, Heidelberg, 1995. Springer-Verlag.
- [63] M. Luck and M. d’Inverno. Motivated behaviour for goal adoption. In C. Zhang and D. Lukose, editors, *Multi-Agent Systems Methodologies and Applications: Proceedings of the Fourth Australian Workshop on Distributed Artificial Intelligence*, pages 53–73. Springer-Verlag, 1998.
- [64] M. Luck, N. Griffiths, and M. d’Inverno. From agent theory to agent construction: A case study. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages (ATAL-96)*, pages 49–63. Springer-Verlag, 1997.
- [65] N. Luhmann. Familiarity, confidence, trust: Problems and alternatives. In D. Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*, pages 94–107. Basil Blackwell, 1988.
- [66] P. Maes. How to do the right thing. *Connection Science*, 1(3):291–323, 1989.
- [67] S. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, 1994.
- [68] S. Marsh. Optimism and pessimism in trust. In *Proceedings of the Ibero-American Conference on Artificial Intelligence (IBERAMIA '94)*, 1994.
- [69] S. Marsh. Trust in distributed artificial intelligence. In C. Castelfranchi and E. Werner, editors, *Artificial Social Systems*, pages 94–112. Springer-Verlag, 1994.

- [70] A. R. Mele. Motivational internalism: The powers and limits of practical reasoning. *Philosophia*, 19(4), 1989.
- [71] B. G. Milnes. A specification of the Soar architecture in Z. Technical report, Carnegie Mellon University, 1992.
- [72] B. Moulin and B. Chaib-Draa. An overview of distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 3–55. John Wiley & Sons, 1996.
- [73] J. P. Müller. Architectures and applications of intelligent agents: A survey. *Knowledge Engineering Review*, 13(4):353–380, 1998.
- [74] J. P. Müller and M. Pischel. Modelling interacting agents in dynamic environments. In A. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 709–713. John Wiley & Sons, 1995.
- [75] T. J. Norman. *Motivation-based direction of planning attention in agents with goal autonomy*. PhD thesis, University of London, 1996.
- [76] T. J. Norman and D. Long. Goal creation in motivated agents. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Proceedings of the First International Workshop on Agent Theories, Architectures and Languages (ATAL-94)*, pages 277–290. Springer-Verlag, 1995.
- [77] T. J. Norman and D. Long. Alarms: An implementation of motivated agency. In M. J. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II: Proceedings of the Second International Workshop on Theories, Architectures and Languages (ATAL-95)*, pages 219–234. Springer-Verlag, 1996.
- [78] H. S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–244, 1996.

- [79] H. S. Nwana and M. Wooldridge. Software agent technologies. *BT Technology Journal*, 14(4):68–78, October 1996.
- [80] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z*. Prentice Hall, 1996.
- [81] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96)*, pages 42–55. Springer-Verlag, 1996.
- [82] A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 318–324, Chambéry, France, 1993.
- [83] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, San Francisco, 1995. AAAI Press/The MIT Press.
- [84] A. S. Rao, M. P. Georgeff, and E. A. Sonenberg. Social plans. In E. Werner and Y. Demazeau, editors, *Decentralized A.I. 3: Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 57–76. Elsevier Science Publishers B.V., 1992.
- [85] J. S. Rosenschein and M. R. Genesereth. Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 91–99, Los Angeles, CA, 1985.
- [86] S. Russell and P. Norvig. *Artificial intelligence: A modern approach*. Prentice Hall, 1995.
- [87] J. R. Searle. *Intentionality: An essay in the philosophy of mind*. Cambridge University Press, 1983.

- [88] J. R. Searle. Collective intentions and actions. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 401–415. MIT Press, 1990.
- [89] J. S. Sichman and Y. Demazeau. Exploiting social reasoning to deal with agency level inconsistency. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 352–359, San Francisco, CA, 1995. AAAI Press/The MIT Press.
- [90] J. S. Sichman, Y. Demazeau, R. Conte, and C. Castelfranchi. A social reasoning mechanism based on dependence networks. In A. Cohn, editor, *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 188–192, Amsterdam, The Netherlands, 1995. John Wiley & Sons.
- [91] H. Sidgwick. *The methods of ethics*. Macmillan, London, 1966.
- [92] H. A. Simon. Motivational and emotional controls of cognition. *Psychological Review*, 74:29–39, 1967.
- [93] A. Sloman. Motives mechanisms and emotions. *Cognition and Emotion*, 1(3):217–234, 1987.
- [94] R. G. Smith. The contract net: A formalism for the control of distributed problem solving. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, page 472, Cambridge, MA, 1977.
- [95] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):61–70, 1981.
- [96] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, Hemel Hempstead, 2nd edition, 1992.
- [97] M. Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, Rhode Island, Providence, 1997.

- [98] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.
- [99] M. Tambe and W. Zhang. Towards flexible teamwork in persistent teams. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, Paris, France, 1998.
- [100] R. Tuomela and K. Miller. We-intentions. *Philosophical Studies*, 53:367–389, 1988.
- [101] B. Williams. Formal structures and social reality. In D. Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*, pages 3–13. Basil Blackwell, 1988.
- [102] M. Wooldridge and N. R. Jennings. Formalizing the cooperative problem solving process. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence (IWDAI-94)*, pages 403–417, Lake Quinhalt, WA, 1994.
- [103] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.
- [104] M. Wooldridge and N. R. Jennings. Cooperative problem-solving. *Journal of Logic and Computation*, 9(4):563–592, 1999.
- [105] M. J. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, University of Manchester, August 1992.
- [106] J. B. Wordsworth. *Software Development with Z*. Addison-Wesley, 1992.

Appendix A

Specification

A.1 Introduction

In this appendix we present the complete specification of the framework developed in this thesis, and give examples of how particular parts of it can be instantiated. The purpose of this appendix is to give the supporting specification for the parts of the framework that are specified elsewhere in the relevant chapters, where the components they specify are introduced. In view of this aim, we let the specification in this chapter stand for itself, and give only a brief supporting text indicating the purpose of each part of the specification.

A.2 Primitives

In this section we define the primitive types on which the specification is based, namely the notions of terms and predicates.

$$[Const, Var, PredSym]$$
$$\begin{aligned} Term ::= & \text{const}\langle\langle Const \rangle\rangle \\ & | \text{var}\langle\langle Var \rangle\rangle \end{aligned}$$

<i>Predicate</i> <i>symbol</i> : <i>PredSym</i> <i>terms</i> : seq <i>Term</i>
--

A.3 Environment

An environment comprises a set of perceivable features, or attributes, where an attribute is simply represented as a predicate. The schema *Env* represents the particular environment in which an agent is situated.

Attribute == *Predicate*
Environment == \mathbb{P} *Attribute*

<i>Env</i> <i>environment</i> : <i>Environment</i>

A.4 Perceptions

A view is a set of perceivable features, or attributes, and a perception action is a function that takes an environment and returns a view, corresponding to the features in the environment that an agent perceives.

View == \mathbb{P} *Attribute*
PerceptionAction == *Environment* \rightarrow *View*

A.5 Beliefs

We define a literal as a predicate or its negation. A belief is then defined as a single literal, and an agent's beliefs as a set of such beliefs.

Literal ::= *pos* $\langle\langle$ *Predicate* $\rangle\rangle$
| *not* $\langle\langle$ *Predicate* $\rangle\rangle$

$Belief == Literal$
 $Beliefs == \mathbb{P} Belief$

A.6 Goals

A goal is a situation that an agent wishes to bring about, and is defined to be a set of literals.

$Goal == \mathbb{P} Literal$

A.7 Actions

A contribution is an action that can be performed by an individual agent, and is defined to comprise an action symbol, a sequence of terms (representing the parameters of the action), and an agent identifier that corresponds to the agent that should perform it. The effects of performing a contribution are defined by the function *contributionEffects*, and its preconditions by the function *contributionPreconditions*.

$[ActSym, AgentID]$

Contribution

$symbol : ActSym$
 $terms : seq Term$
 $agentID : AgentID$

$| contributionEffects : Contribution \rightarrow Environment \rightarrow Environment$

$Precondition == Literal$

$| contributionPreconditions : Contribution \rightarrow \mathbb{P} Precondition$

A.8 Joint and Concurrent Actions

In this section we define joint and concurrent actions, both of which are composite actions made up of individual contributions; the contributions in a joint action must be synchronised and performed together, while those in a concurrent action do not require such synchronisation.

<i>JointAction</i>
<i>contributions</i> : \mathbb{P} <i>Contribution</i>
<i>#contributions</i> ≥ 2

CAcomponent ::= *Contrib* $\langle\langle$ *Contribution* $\rangle\rangle$
| *JA* $\langle\langle$ \mathbb{P} *Contribution* $\rangle\rangle$

<i>ConcurrentAction</i>
<i>contributions</i> : \mathbb{P} <i>CAcomponent</i>
<i>#contributions</i> ≥ 2

A.9 Plans

In order to specify a plan, we first specify that a step in a plan is either an individual, joint or concurrent action, or a subgoal. We then specify a plan to comprise a sequence of steps to achieve a goal, under a particular set of preconditions.

PlanStep ::= *Individual* $\langle\langle$ *Contribution* $\rangle\rangle$
| *Joint* $\langle\langle$ \mathbb{P} *Contribution* $\rangle\rangle$
| *Concurrent* $\langle\langle$ \mathbb{P} *CAcomponent* $\rangle\rangle$
| *Subgoal* $\langle\langle$ *Goal* $\rangle\rangle$

<i>Plan</i>
<i>achieves</i> : <i>Goal</i>
<i>preconditions</i> : \mathbb{P} <i>Belief</i>
<i>body</i> : seq <i>PlanStep</i>

A.10 Intentions

In this section we define an intention as containing a stack of plans (represented by a sequence), a relevance condition, and the goal to which the intention is towards. The sequence of plans are constrained such that the plan at position $i + 1$ is a subplan of that at position i .

$bool ::= true \mid false$

$$\frac{isSubPlanOf : Plan \rightarrow Plan \rightarrow bool}{\forall p, q : Plan \bullet isSubPlanOf \ p \ q = true \Leftrightarrow (\exists g : Goal \bullet Subgoal(g) = head(q.body) \wedge p.achieves = g)}$$

$$\frac{extractPlan : (\mathbb{N}_1 \times seq\ Plan) \rightarrow Plan}{\forall i : \mathbb{N}_1; pseq : seq\ Plan \mid i \leq \#pseq \bullet \exists_1 p : Plan \bullet extractPlan \ (i, pseq) = p \wedge \{p\} = ran(\{i\} \upharpoonright pseq)}$$

<p><i>Intention</i></p> <p>$plans : seq\ Plan$ $relevance : \mathbb{P}\ Belief$ $satisfies : Goal$</p> <hr style="border: 0.5px solid black;"/> <p>$\forall i : \mathbb{N}_1 \mid i \leq \#plans - 1 \bullet$ $isSubPlanOf \ (extractPlan \ (i + 1, plans)) \ (extractPlan \ (i, plans))$ $= true$</p>
--

A.11 Motivations

We introduce a given set to represent the set of motivation symbols, and define a motivation to have a name (in the form of a motivation symbol), intensity, threshold, and a set of goals that it can generate.

$[MotiveSym]$

Motivation

name : *MotiveSym*

intensity : \mathbb{R}

threshold : \mathbb{R}

goals : \mathbb{P} *Goal*

The following functions are applied to motivations. The first determines the intensity that the motivation should take in a given believed situation, while the latter four return the motivational value of a goal, contribution, plan, and intention respectively.

assessSituation : *Motivation* \rightarrow \mathbb{P} *Belief* \rightarrow \mathbb{R}

mitigation : *Motivation* \rightarrow *Goal* \rightarrow \mathbb{R}

mvContribution : *Motivation* \rightarrow *Contribution* \rightarrow \mathbb{R}

mvPlan : *Motivation* \rightarrow *Plan* \rightarrow \mathbb{R}

mvIntention : *Motivation* \rightarrow *Intention* \rightarrow \mathbb{R}

A.12 Agent Mental Components

Before specifying the mental components that comprise a SENARA agent, we introduce a given set to represent a nominal commitment; we define nominal commitment later, and so this is simply a forward definition. We specify an agent as having a unique identifier and a set of capabilities, perceiving capabilities, beliefs, goals, intentions, motivations, and nominal commitments, along with a plan library from which it can select the most appropriate for its goals.

[*NominalCommitment*]

Agent

agentID : *AgentID*
capabilities : \mathbb{P} *Contribution*
perceivingCapabilities : \mathbb{P} *PerceptionAction*
beliefs : \mathbb{P} *Belief*
goals : \mathbb{P} *Goal*
intentions : \mathbb{P} *Intention*
motivations : \mathbb{P} *Motivation*
planLibrary : \mathbb{P} *Plan*
nominalCommitments : \mathbb{P} *NominalCommitment*

motivations $\neq \emptyset$

$\forall c : \text{Contribution} \bullet c \in \text{capabilities} \Leftrightarrow c.\text{agentID} = \text{agentID}$

A.13 Perceiving the Environment

In the following schema we specify perception by defining an agent's current view to be the combined result of applying its perceiving capabilities to the current environment.

AgentPerception

$\exists Env$

$\exists Agent$

view : *View*

$view = \bigcup \{v : View \mid (\exists pAct : PerceptionAction \mid pAct \in perceivingCapabilities \bullet v = pAct \text{ environment})\}$

A.14 Updating Beliefs

An agent updates its beliefs in the light of its perceptions, by translating the attributes in its current view into a set of candidate beliefs, and then it revising its beliefs to include those candidate beliefs that are appropriate.

UpdateBeliefs

Δ Agent

\exists AgentPerception

interpretView : View \rightarrow \mathbb{P} Belief

reviseBeliefs : \mathbb{P} Belief \rightarrow \mathbb{P} Belief \rightarrow \mathbb{P} Belief

candidateBeliefs : \mathbb{P} Belief

candidateBeliefs = *interpretView* view

beliefs' = *reviseBeliefs* *beliefs* *candidateBeliefs*

goals' = *goals*

intentions' = *intentions*

motivations' = *motivations*

A.15 Updating Motivations

In this section we formalise the process through which an agent updates the intensities of its motivations. We define the notion of an intensity association, which is a 3-tuple containing a set of beliefs, a motivation, and an intensity change, such that the motivation's intensity should change proportionally by the specified amount if the agent holds the beliefs. The schema *UpdateMotivations*, defines how an agent should update the intensity of its motivations using a set of intensity associations.

[X, Y, Z]

First : $X \times Y \times Z \rightarrow X$

Second : $X \times Y \times Z \rightarrow Y$

Third : $X \times Y \times Z \rightarrow Z$

$\forall x : X; y : Y; z : Z \bullet \text{First}(x, y, z) = x \wedge$

$\text{Second}(x, y, z) = y \wedge \text{Third}(x, y, z) = z$

iAssociation == \mathbb{P} Belief \times Motivation \times \mathbb{R}

IntensityAssociation

iAssociations : \mathbb{P} *iAssociation*

$$\begin{aligned} &\forall B : \mathbb{P} \textit{Belief}; m : \textit{Motivation}; i : \mathbb{R}; a : \textit{iAssociation} \bullet \\ &\quad a = (B, m, i) \wedge a \in \textit{iAssociations} \\ &\quad \Rightarrow (\forall j : \mathbb{R}; a' : \textit{iAssociation} \mid a' = (B, m, j) \wedge \\ &\quad \quad a' \in \textit{iAssociations} \bullet i = j) \end{aligned}$$

UpdateMotivations

Δ *Agent*

\exists *IntensityAssociation*

getIntensity : \mathbb{P} *iAssociation* \rightarrow \mathbb{P} *Belief* \rightarrow *Motivation* \rightarrow \mathbb{R}

$$\begin{aligned} &\forall IA : \mathbb{P} \textit{iAssociation}; B : \mathbb{P} \textit{Belief}; m : \textit{Motivation}; i : \mathbb{R} \bullet \\ &\quad \textit{getIntensity} \textit{IA} \textit{B} \textit{m} = i \\ &\quad \Leftrightarrow (\exists \textit{iA} : \textit{iAssociation} \bullet \textit{iA} \in \textit{IA} \wedge B = \textit{First}(\textit{iA}) \wedge \\ &\quad \quad m = \textit{Second}(\textit{iA}) \wedge i = \textit{Third}(\textit{iA})) \\ &\forall m : \textit{Motivation} \mid m \in \textit{motivations} \bullet (\exists_1 m' : \textit{Motivation} \mid \\ &\quad m' \in \textit{motivations}' \bullet m'.\textit{name} = m.\textit{name} \wedge \\ &\quad \quad m'.\textit{intensity} = m.\textit{intensity} * \\ &\quad \quad (\textit{getIntensity} \textit{iAssociations} \textit{beliefs} \textit{m}) \wedge \\ &\quad \quad m'.\textit{threshold} = m.\textit{threshold} \wedge m'.\textit{goals} = m.\textit{goals}) \\ &\textit{beliefs}' = \textit{beliefs} \\ &\textit{goals}' = \textit{goals} \\ &\textit{intentions}' = \textit{intentions} \end{aligned}$$

A.16 Ensuring Goals are Motivated

An agent's goals must be of motivational value, and we formalise this in the following schema.

DropUnmotivatedGoals

$\Delta Agent$

$\forall g : Goal \mid g \in goals \bullet g \in goals' \Leftrightarrow (\exists m : Motivation \mid$
 $m \in motivations \bullet (mitigation\ m\ g) > 0 \wedge m.intensity > 0)$
 $\forall g : Goal \mid g \in goals \bullet g \notin goals' \Leftrightarrow (\forall m : Motivation \mid$
 $m \in motivations \bullet mitigation\ m\ g = 0 \vee m.intensity = 0)$
 $beliefs' = beliefs$
 $intentions' = intentions$
 $motivations' = motivations$

A.17 Goal Generation

In this section we specify how an agent generates new goals according to its motivations. The function *generateGoals*, takes a particular motivation, an agent's beliefs and its other motivations, and returns a set of goals that are generated by that motivation given the current believed situation.

$\mid generateGoals : Motivation \rightarrow \mathbb{P} Belief \rightarrow \mathbb{P} Motivation \rightarrow \mathbb{P} Goal$

Now, different agents may utilise different strategies for generating goals, meaning that agents may have different instantiations of the above function. However, a simple approach is to mirror that taken in assessing the intensity of motivations, such that agents have a set of associations (in the form of a set of 3-tuples) that determine the goals that are generated in a particular situation.

$gAssociation == \mathbb{P} Belief \times Motivation \times \mathbb{P} Goal$

GoalGenerationAssociation

$gAssociations : \mathbb{P} gAssociation$

$\forall B : \mathbb{P} Belief; m : Motivation; G : \mathbb{P} Goal; a : gAssociation \bullet$
 $a = (B, m, G) \wedge a \in gAssociations$
 $\Rightarrow (\forall H : \mathbb{P} Goal; a' : gAssociation \mid a' = (B, m, H) \wedge$
 $a' \in gAssociations \bullet G = H)$

GoalGeneration

$\Delta Agent$

activeMotivations : $\mathbb{P} Motivation$

generatedGoals : $\mathbb{P} Goal$

activeMotivations =

$\{m : Motivation \mid m \in motivations \wedge m.intensity \geq m.threshold\}$

generatedGoals =

$\cup \{G : \mathbb{P} Goal \mid (\exists m : Motivation \mid m \in activeMotivations \bullet$
 $G = generateGoals\ m\ believes\ motivations)\}$

goals' = *goals* \cup *generatedGoals*

beliefs' = *beliefs*

intentions' = *intentions*

motivations' = *motivations*

A.18 Ensuring Intentions are Appropriate

In the same way that an agent's goals must be of motivational value, so must its intentions. However, an intention should also be dropped if it is believed to be achieved, unachievable, or irrelevant, and we formalise this below.

DropInappropriateIntentions

Δ Agent

$isAchieved : Intention \rightarrow \mathbb{P} Belief \rightarrow bool$

$isAchievable : Intention \rightarrow \mathbb{P} Belief \rightarrow bool$

$isRelevant : Intention \rightarrow bool$

$isMotivated : Intention \rightarrow bool$

$\forall i : Intention \mid i \in intentions \bullet isRelevant i = true$
 $\Leftrightarrow (\forall b : Belief \mid b \in i.relevance \bullet b \in beliefs) \wedge$
 $isRelevant i = false \Leftrightarrow (\exists b : Belief \mid b \in i.relevance \bullet b \notin beliefs)$

$\forall i : Intention \mid i \in intentions \bullet isMotivated i = true$
 $\Leftrightarrow (\exists m : Motivation \mid m \in motivations \bullet$
 $m.intensity > 0 \wedge (mitigation m i.satisfies) > 0) \wedge$
 $isMotivated i = false \Leftrightarrow (\forall m : Motivation \mid m \in motivations \bullet$
 $mitigation m (i.satisfies) = 0 \vee m.intensity = 0)$

$\forall i : Intention \mid i \in intentions \bullet i \in intentions'$
 $\Leftrightarrow isAchieved i beliefs = false \wedge isAchievable i beliefs = true \wedge$
 $isRelevant i = true \wedge isMotivated i = true \wedge$
 $i \notin intentions'$
 $\Leftrightarrow isAchieved i beliefs = true \vee isAchievable i beliefs = false \vee$
 $isRelevant i = false \vee isMotivated i = false$

$beliefs' = beliefs$
 $goals' = goals$
 $motivations' = motivations$

A.19 Intention Adoption

For each of its active motivations an agent should attempt to adopt an intention for the goal generated by that motivation, or the most motivated goal if more than one goal is generated. When incompatibilities are found they should be resolved in such a way as to afford the highest motivational value to the agent.

$preconMet : \mathbb{P} Literal \rightarrow Beliefs \rightarrow bool$

$pMet : Literal \rightarrow Beliefs \rightarrow bool$

$\forall l : Literal; bel : Beliefs \bullet pMet l bel = true$

$\Leftrightarrow (\exists p : Predicate \bullet l = pos(p) \wedge pos(p) \in bel) \vee$
 $(\exists p : Predicate \bullet l = not(p) \wedge pos(p) \notin bel)$

$\forall l : Literal; bel : Beliefs \bullet pMet l bel = false$

$\Leftrightarrow (\exists p : Predicate \bullet l = pos(p) \wedge pos(p) \notin bel) \vee$
 $(\exists p : Predicate \bullet l = not(p) \wedge pos(p) \in bel)$

$\forall L : \mathbb{P} Literal; bel : Beliefs \bullet preconMet L bel = true$

$\Leftrightarrow (\forall l : Literal \mid l \in L \bullet pMet l bel = true)$

$\forall L : \mathbb{P} Literal; bel : Beliefs \bullet preconMet L bel = false$

$\Leftrightarrow (\exists l : Literal \mid l \in L \bullet pMet l bel = false)$

$planSetForGoal : Goal \rightarrow \mathbb{P} Belief \rightarrow \mathbb{P} Plan \rightarrow \mathbb{P} Plan$

$\forall g : Goal; bel : \mathbb{P} Belief; plib : \mathbb{P} Plan \bullet planSetForGoal g bel plib$
 $= \{p : Plan \mid p \in plib \wedge p.achieves = g \wedge$
 $preconMet p.preconditions bel = true\}$

$planForGoal : \mathbb{P} Belief \rightarrow \mathbb{P} Intention \rightarrow \mathbb{P} Plan \rightarrow Goal \rightarrow Plan$

IntentionAdoption

$\Delta Agent$

$activeMotivations : \mathbb{P} Motivation$

$activeGoals : \mathbb{P} Goal$

$currentIntendedGoals : \mathbb{P} Goal$

$newIntendedGoals : \mathbb{P} Goal$

$resolveIncompatibilities : \mathbb{P} Goal \rightarrow \mathbb{P} Goal \rightarrow \mathbb{P} Goal$

$activeMotivations =$

$\{m : Motivation \mid m \in motivations \wedge m.intensity \geq m.threshold\}$

$activeGoals = \{g : Goal \mid g \in goals \wedge$

$(\exists m : Motivation \mid m \in activeMotivations \wedge mitigation m g > 0 \bullet$

$(\forall g' : Goal \mid g' \in goals \wedge g' \neq g \bullet$

$(mitigation m g) \geq (mitigation m g'))\})\}$

$currentIntendedGoals =$

$\{g : Goal \mid (\exists i : Intention \mid i \in intentions \bullet i.satisfies = g)\}$

$newIntendedGoals = resolveIncompatibilities$

$currentIntendedGoals activeGoals$

$intentions' = \{i : Intention \mid i.satisfies \in newIntendedGoals \wedge$

$i \in intentions\} \cup \{i : Intention \mid i.satisfies \in newIntendedGoals \wedge$

$i \notin intentions \wedge head(i.plans) =$

$planForGoal beliefs intentions planLibrary i.satisfies\}$

A.20 Intention Selection

In order to act, an agent must select an intention to focus upon, as determined by its motivations. To select an intention, an agent selects the motivation that is currently of the highest importance, and selects an intention to pursue, by choosing the one that currently offers the greatest motivational value to this motivation.

IntentionSelection

\exists Agent

activeMotivations : \mathbb{P} Motivation

chosenMotivation : Motivation

chosenIntention : Intention

activeMotivations =

$\{m : \text{Motivation} \mid m \in \text{motivations} \wedge m.\text{intensity} \geq m.\text{threshold}\}$

activeMotivations $\neq \emptyset \Rightarrow (\exists m : \text{Motivation} \mid m \in \text{activeMotivations} \bullet$

$(\forall m' : \text{Motivation} \mid m' \in \text{activeMotivations} \wedge m' \neq m \bullet$

$m.\text{intensity} - m.\text{threshold} \geq m'.\text{intensity} - m'.\text{threshold} \wedge$

$\text{chosenMotivation} = m))$

activeMotivations = $\emptyset \Rightarrow (\exists m : \text{Motivation} \mid m \in \text{motivations} \bullet$

$(\forall m' : \text{Motivation} \mid m' \in \text{motivations} \wedge m' \neq m \bullet$

$m.\text{intensity} - m.\text{threshold} \geq m'.\text{intensity} - m'.\text{threshold} \wedge$

$\text{chosenMotivation} = m))$

$(\exists_1 i : \text{Intention} \mid i \in \text{intentions} \bullet (\forall i' : \text{Intention} \mid i' \in \text{intentions} \wedge$

$i \neq i' \bullet \text{mitigation chosenMotivation } i.\text{satisfies} \geq$

$\text{mitigation chosenMotivation } i'.\text{satisfies} \wedge$

$\text{chosenIntention} = i))$

A.21 Action and Deliberation

After determining its chosen intention, an agent works towards it — if the next step in the intention is an individual contribution then an agent can execute it, if the step is a subgoal then the agent must elaborate the plan, otherwise if the step is a cooperative action the agent must initiate cooperation. In this section we give specification for the former two cases, action and deliberation.

| *believedChanges* : Contribution \rightarrow Environment \rightarrow \mathbb{P} Belief

AgentHistory
history : seq *Contribution*

AgentAction
 Δ *Agent*
 Δ *Env*
 Δ *AgentHistory*
 \exists *IntentionSelection*
 \exists *UpdateBeliefs*
nextStep : *PlanStep*

nextStep = head(*last chosenIntention.plans*).*body*
 $\exists a$: *Contribution* • *Individual*(*a*) = *nextStep*
 \Leftrightarrow *history'* = *history* $\hat{\ } \langle a \rangle \wedge$ *environment'* =
contributionEffects a environment \wedge *beliefs'* =
reviseBeliefs beliefs (believedChanges a environment)

AgentDeliberation
 Δ *Agent*
 \exists *IntentionSelection*
nextStep : *PlanStep*

nextStep = head(*last chosenIntention.plans*).*body*
 $\exists g$: *Goal* • *Subgoal*(*g*) = *nextStep*
 \Leftrightarrow *chosenIntention'.plans* = *chosenIntention.plans*
 $\hat{\ } \langle (\text{planForGoal beliefs intentions planLibrary } g) \rangle$
 \wedge *chosenIntention'.relevance* = *chosenIntention.relevance*
 \wedge *chosenIntention'.satisfies* = *chosenIntention.satisfies*

The following schema brings together the various functions described above, and corresponds to the agent control mechanisms in the SENARA architecture.

AgentControl

\exists *Agent*

AgentPerception

UpdateBeliefs

UpdateMotivations

DropUnmotivatedGoals

GoalGeneration

DropInappropriateIntentions

IntentionAdoption

IntentionSelection

AgentAction

AgentDeliberation

A.22 Necessary and Optional Cooperation

In this section we formalise the notions of *necessary* and *optional* cooperation.

$stepcontributions : PlanStep \rightarrow (\mathbb{P} Contribution)$

$CAcomponentcontributions : CAcomponent \rightarrow (\mathbb{P} Contribution)$

$plancontributions : Plan \rightarrow (\mathbb{P} Contribution)$

$\forall c : Contribution; cs : \mathbb{P} Contribution; cacs : \mathbb{P} CAcomponent; g : Goal \bullet$

$stepcontributions (Individual\ c) = \{c\} \wedge$

$stepcontributions (Joint\ cs) = cs \wedge$

$stepcontributions (Concurrent\ cacs) =$

$\bigcup \{cs' : \mathbb{P} Contribution; cac : CAcomponent \mid cac \in cacs \wedge$

$cs' = CAcomponentcontributions\ cac \bullet cs'\} \wedge$

$stepcontributions (Subgoal\ g) = \emptyset$

$\forall comp : CAcomponent; cs : \mathbb{P} Contribution \bullet$

$CAcomponentcontributions\ comp = cs \Leftrightarrow$

$(\exists c : Contribution \bullet comp = (Contrib\ c) \wedge$

$cs = stepcontributions (Individual\ c)) \vee$

$(\exists cs : \mathbb{P} Contribution \bullet comp = (JA\ cs) \wedge$

$cs = stepcontributions (Joint\ cs))$

$\forall p : Plan \bullet plancontributions\ p =$

$\bigcup \{s : PlanStep \mid s \in (ran\ p.body) \bullet stepcontributions\ s\}$

$$\begin{array}{|l}
\hline
\text{nesscooperates} : \text{Agent} \rightarrow \text{Goal} \rightarrow \text{bool} \\
\hline
\forall \text{ag} : \text{Agent}; \text{g} : \text{Goal} \mid \text{g} \in \text{ag.goals} \bullet \text{nesscooperates ag g} = \text{true} \\
\Leftrightarrow (\forall \text{p} : \text{Plan} \mid \text{p} \in \text{ag.planLibrary} \bullet \text{p.achieves} = \text{g} \wedge \\
\quad (\text{plancontributions p} \setminus \text{ag.capabilities}) \neq \emptyset) \\
\hline
\text{optcooperates} : \text{Agent} \rightarrow \text{Goal} \rightarrow \text{bool} \\
\hline
\forall \text{ag} : \text{Agent}; \text{g} : \text{Goal} \mid \text{g} \in \text{ag.goals} \bullet \text{optcooperates ag g} = \text{true} \\
\Leftrightarrow (\exists \text{p}, \text{q} : \text{Plan} \mid \text{p} \in \text{ag.planLibrary} \wedge \text{q} \in \text{ag.planLibrary} \bullet \\
\quad \text{p.achieves} = \text{g} \wedge \text{q.achieves} = \text{g} \wedge \\
\quad (\text{plancontributions p} \setminus \text{ag.capabilities}) \neq \emptyset) \wedge \\
\quad (\text{plancontributions q} \setminus \text{ag.capabilities} = \emptyset)
\end{array}$$

A.23 A Model of Cooperative Plan Selection

In Chapter 6 we described a procedure for plan selection in a cooperative environment, based on standard planning heuristics and knowledge of others capabilities and trustworthiness. We introduce two ratings for plans, a *standard* rating arrived at by applying standard heuristics, and a *cooperative* rating that is based on the risk associated with cooperation. In this section we formalise the procedure for assessing the quality of a plan, using these ratings.

The following function represents the heuristic for determining the standard rating of a plan.

$$\begin{array}{|l}
\text{sRating} : \text{Plan} \rightarrow \mathbb{R}
\end{array}$$

The cooperative rating of a plan is based on an agent's knowledge of others' capabilities and its trust of them, and so before formalising the procedure for obtaining the cooperative rating of plan we introduce the notion of an agent model, and formalise how to extract the trust associated with a particular agent from a set of such models.

$$\begin{array}{|l}
\text{AgentModel} \\
\hline
\text{id} : \text{AgentID} \\
\text{capabilities} : \mathbb{P} \text{Contribution} \\
\text{trust} : \mathbb{R}
\end{array}$$

$$\begin{array}{l} \text{extractAllModels} : \mathbb{P} \text{Belief} \rightarrow \mathbb{P} \text{AgentModel} \\ \text{extractModel} : \mathbb{P} \text{Belief} \rightarrow \text{AgentID} \rightarrow \text{AgentModel} \end{array}$$

$$\text{trustOfAgent} : \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R}$$

$$\begin{array}{l} \forall \text{agID} : \text{AgentID}; \text{ms} : \mathbb{P} \text{AgentModel}; r : \mathbb{R} \bullet \\ \text{trustOfAgent agID ms} = r \Rightarrow r > 0 \wedge r < 1 \wedge \\ (\exists_1 m : \text{AgentModel} \mid m \in \text{ms} \bullet m.\text{id} = \text{agID} \wedge m.\text{trust} = r) \end{array}$$

$$\text{sumSeq} : \text{seq } \mathbb{R} \rightarrow \mathbb{R}$$

$$\begin{array}{l} \forall s : \text{seq } \mathbb{R}; n : \mathbb{R} \bullet \text{sumSeq } s = n \Leftrightarrow \\ (\#s = 1 \wedge n = s1) \vee (\#s > 1 \wedge n = s1 + \text{sumSeq}(\text{tail } s)) \end{array}$$

$$\text{scaleTrustSeq} : \text{seq } \mathbb{R} \rightarrow \text{seq } \mathbb{R}$$

$$\begin{array}{l} \forall \text{ts}, \text{scaledTs} : \text{seq } \mathbb{R} \bullet \text{scaleTrustSeq } \text{ts} = \text{scaledTs} \Leftrightarrow \\ \# \text{ts} = \# \text{scaledTs} \wedge (\forall n : \mathbb{Z} \mid n > 0 \wedge n \leq \# \text{ts} \bullet \\ \text{scaledTs } n = (\text{ts } n) / n) \end{array}$$

$$\text{capableAgents} : \text{Contribution} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{P} \text{AgentID}$$

$$\text{orderedCapableAgents} : \text{Contribution} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \text{seq } \text{AgentID}$$

$$\text{orderedTrust} : \text{seq } \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \text{seq } \mathbb{R}$$

$$\begin{array}{l} \forall c : \text{Contribution}; \text{ms} : \mathbb{P} \text{AgentModel} \bullet (\text{capableAgents } c \text{ ms}) = \\ \{ a : \text{AgentID}; m : \text{AgentModel} \mid (m \in \text{ms}) \wedge \\ (a = m.\text{id}) \wedge (c \in m.\text{capabilities}) \bullet a \} \\ \forall c : \text{Contribution}; \text{ms} : \mathbb{P} \text{AgentModel}; \text{orderedCapable} : \text{seq } \text{AgentID} \bullet \\ \text{orderedCapableAgents } c \text{ ms} = \text{orderedCapable} \wedge \\ \text{ran } \text{orderedCapable} = (\text{capableAgents } c \text{ ms}) \wedge \\ (\forall n : \mathbb{Z} \mid n \geq 2 \wedge n \leq \# \text{orderedCapable} \bullet \\ \text{trustOfAgent } (\text{orderedCapable } n) \text{ ms} > \\ \text{trustOfAgent } (\text{orderedCapable } (n - 1)) \text{ ms}) \\ \forall \text{orderedC} : \text{seq } \text{AgentID}; \text{ms} : \mathbb{P} \text{AgentModel}; \text{orderedT} : \text{seq } \mathbb{R} \bullet \\ \text{orderedTrust } \text{orderedC} \text{ ms} = \text{orderedT} \Leftrightarrow \\ \# \text{orderedC} = \# \text{orderedT} \wedge \\ (\forall n : \mathbb{Z} \mid n > 0 \wedge n \leq \# \text{orderedC} \bullet \\ \text{orderedT } n = \text{trustOfAgent } (\text{orderedC } n) \text{ ms}) \end{array}$$

We can now formalise the risk associated with a particular contribution — determined by considering the risk associated with each of the capable agents, such that the risk from less trusted agents is divided by a correspondingly increasing factor.

$$\begin{array}{l}
\text{riskC} : \text{Contribution} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \\
\hline
\forall c : \text{Contribution}; ms : \mathbb{P} \text{AgentModel}; r : \mathbb{R} \bullet \text{riskC } c \text{ } ms = r \\
\quad \Leftrightarrow 1 / \text{sumSeq} (\text{scaleTrustSeq} (\text{orderedTrust} (\\
\quad \quad \text{orderedCapableAgents } c \text{ } ms)) = r
\end{array}$$

A.23.1 Assessing Joint Actions

In this section we extend this strategy to apply to joint actions, by simply replacing the agents and trust of an agent in the equation with sets of agents that are capable of performing the action, and the trust of these sets of agents respectively.

$$\begin{array}{l}
\text{capableAgentSets} : \mathbb{P} \text{Contribution} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{P}(\mathbb{P} \text{AgentID}) \\
\hline
\forall cs : \mathbb{P} \text{Contribution}; ms : \mathbb{P} \text{AgentModel} \bullet \text{capableAgentSets } cs \text{ } ms = \\
\quad \{ agts : \mathbb{P} \text{AgentID}; c : \text{Contribution} \mid c \in cs \wedge \\
\quad (\exists a : \text{AgentID} \bullet a \in agts \wedge a \in \text{capableAgents } c \text{ } ms \wedge \\
\quad (\forall a' : \text{AgentID} \mid a' \neq a \wedge a' \in agts \bullet \\
\quad \quad a' \notin \text{capableAgents } c \text{ } ms)) \bullet agts \}
\end{array}$$

$$\begin{array}{l}
\text{trustOfAgentSet} : \mathbb{P} \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \\
\text{productSet} : \mathbb{P} \mathbb{R} \rightarrow \mathbb{R} \\
\hline
\forall agIDs : \mathbb{P} \text{AgentID}; ms : \mathbb{P} \text{AgentModel} \bullet \\
\quad \text{trustOfAgentSet } agIDs \text{ } ms = \text{productSet} \\
\quad \{ r : \mathbb{R}; a : \text{AgentID} \mid a \in agIDs \wedge r = \text{trustOfAgent } a \text{ } ms \bullet r \} \\
\forall s : \mathbb{P} \mathbb{R}; n : \mathbb{R} \bullet \text{productSet } s = n \Leftrightarrow \\
\quad (\#s = 1 \wedge s = \{n\}) \vee \\
\quad (\#s > 1 \wedge (\exists r : \mathbb{R} \mid r \in s \bullet n = r * \text{productSet}(s \setminus \{r\})))
\end{array}$$

$$\begin{aligned} \text{orderedCapableAgentSets} &: \mathbb{P} \text{Contribution} \rightarrow \mathbb{P} \text{AgentModel} \\ &\rightarrow \text{seq}(\mathbb{P} \text{AgentID}) \\ \text{orderedTrustSet} &: \text{seq}(\mathbb{P} \text{AgentID}) \rightarrow \mathbb{P} \text{AgentModel} \\ &\rightarrow \text{seq } \mathbb{R} \end{aligned}$$

$$\begin{aligned} \forall cs : \mathbb{P} \text{Contribution}; ms : \mathbb{P} \text{AgentModel}; \\ \text{orderedCapable} : \text{seq}(\mathbb{P} \text{AgentID}) \bullet \\ \text{orderedCapableAgentSets } cs \ ms = \text{orderedCapable} \wedge \\ \text{ran } \text{orderedCapable} = (\text{capableAgentSets } cs \ ms) \wedge \\ (\forall n : \mathbb{Z} \mid n \geq 2 \wedge n \leq \#\text{orderedCapable} \bullet \\ \text{trustOfAgentSet } (\text{orderedCapable } n) \ ms > \\ \text{trustOfAgentSet } (\text{orderedCapable } (n - 1)) \ ms) \\ \forall \text{orderedAgts} : \text{seq}(\mathbb{P} \text{AgentID}); ms : \mathbb{P} \text{AgentModel}; \text{orderedT} : \text{seq } \mathbb{R} \bullet \\ \text{orderedTrustSet } \text{orderedAgts} \ ms = \text{orderedT} \\ \Leftrightarrow \#\text{orderedAgts} = \#\text{orderedT} \wedge \\ (\forall n : \mathbb{Z} \mid n > 0 \wedge n \leq \#\text{orderedAgts} \bullet \\ \text{orderedT } n = \text{trustOfAgentSet } (\text{orderedAgts } n) \ ms) \end{aligned}$$

$$\text{riskJA} : \mathbb{P} \text{Contribution} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R}$$

$$\begin{aligned} \forall cs : \mathbb{P} \text{Contribution}; ms : \mathbb{P} \text{AgentModel}; r : \mathbb{R} \bullet \text{riskJA } cs \ ms = r \\ \Leftrightarrow 1/\text{sumSeq } (\text{scaleTrustSeq } (\text{orderedTrustSet} \\ (\text{orderedCapableAgentSets } cs \ ms) \ ms)) = r \end{aligned}$$

A.23.2 Assessing Concurrent Actions

The mechanism for concurrent actions is an extension of that for joint actions since, instead of a set of contributions, a concurrent action comprises a set of sequences of steps, each to be performed concurrently with the others.

$$\text{riskCAcomponent} : \text{CAcomponent} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R}$$

$$\begin{aligned} \forall \text{comp} : \text{CAcomponent}; ms : \mathbb{P} \text{AgentModel}; r : \mathbb{R} \bullet \\ \text{riskCAcomponent } \text{comp} \ ms = r \Leftrightarrow (\exists c : \text{Contribution} \bullet \\ \text{comp} = \text{Contrib } c \wedge r = \text{riskC } c \ ms) \vee \\ (\exists cs : \mathbb{P} \text{Contribution} \bullet \text{comp} = \text{JA } cs \wedge r = \text{riskJA } cs \ ms) \end{aligned}$$

$$\text{riskCA} : \mathbb{P} \text{CAcomponent} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R}$$

$$\begin{aligned} \forall \text{comps} : \mathbb{P} \text{CAcomponent}; ms : \mathbb{P} \text{AgentModel}; \bullet \\ \text{riskCA } \text{comps} \ ms = \text{productSet}\{ca : \text{CAcomponent}; r : \mathbb{R} \mid \\ ca \in \text{comps} \wedge r = \text{riskCAcomponent } ca \ ms \bullet r\} \end{aligned}$$

A.23.3 Cooperative Rating of a Plan

Using this measure of risk of actions, we can determine the *cooperative rating* of a plan by summing the risk associated with each step in it.

$$\begin{array}{|l}
 \hline
 \text{riskPlanStep} : \text{PlanStep} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \\
 \hline
 \forall ps : \text{PlanStep}; ms : \mathbb{P} \text{AgentModel}; r : \mathbb{R} \bullet \text{riskPlanStep } ps \ ms = r \Leftrightarrow \\
 (\exists c : \text{Contribution} \bullet ps = \text{Individual } c \wedge r = \text{riskC } c \ ms) \vee \\
 (\exists cs : \mathbb{P} \text{Contribution} \bullet ps = \text{Joint } cs \wedge r = \text{riskJA } cs \ ms) \vee \\
 (\exists cacs : \mathbb{P} \text{CAcomponent} \bullet ps = \text{Concurrent } cacs \wedge r = \\
 \text{riskCA } cacs \ ms) \\
 \\
 \hline
 \text{cRating} : \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \\
 \text{sumSet} : \mathbb{P} \mathbb{R} \rightarrow \mathbb{R} \\
 \hline
 \forall p : \text{Plan}; ps : \text{seq } \text{PlanStep}; ms : \mathbb{P} \text{AgentModel}; r : \mathbb{R} \bullet \\
 \text{cRating } p \ ms = r \Leftrightarrow ps = p.\text{body} \wedge (\exists \text{ratings} : \mathbb{P} \mathbb{R} \bullet \text{ratings} = \\
 \{r' : \mathbb{R} \mid (\exists s : \text{PlanStep} \mid s \in \text{ran } p.\text{body} \bullet r' = \\
 \text{riskPlanStep } s \ ms) \bullet r'\} \wedge r = \text{sumSet ratings}) \\
 \forall s : \mathbb{P} \mathbb{R}; n : \mathbb{R} \bullet \text{sumSet } s = n \Leftrightarrow \\
 (\#s = 1 \wedge s = \{n\}) \vee \\
 (\#s > 1 \wedge (\exists r : \mathbb{R} \mid r \in s \bullet n = r + \text{sumSet}(s \setminus \{r\})))
 \end{array}$$

A.23.4 Plan Quality

Once both the *standard* and *cooperative* ratings of a plan have been determined, they are combined to form an overall measure of plan quality.

$$\begin{array}{|l}
 \hline
 \text{quality} : \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\
 \hline
 \forall p : \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c, r : \mathbb{R} \bullet \\
 \text{quality } p \ ms \ w_s \ w_c = r \Leftrightarrow r = (\text{sRating } p) * w_s + (\text{cRating } p \ ms) * w_c
 \end{array}$$

A.24 Cooperation in Partial Plans

In Section 6.5 we described the pre-execution assessment mechanism for dealing with partial plans, where each plan is assessed prior to execution time. Since it is not known how a

partial plan will later be elaborated, two ratings are associated with each plan, a *best-case* and a *mean-case* rating.

$$\begin{array}{|l} \hline \text{plansubgoals} : \text{Plan} \rightarrow \mathbb{P} \text{Goal} \\ \hline \forall p : \text{Plan} \bullet \text{plansubgoals } p \\ = \{g : \text{Goal} \mid (\exists s : \text{PlanStep} \mid s \in (\text{ran } p.\text{body}) \bullet s = \text{Subgoal } g) \bullet g\} \end{array}$$

$$\begin{array}{|l} \hline \text{recursive} : \text{Plan} \rightarrow \text{Plan} \rightarrow \text{bool} \end{array}$$

$$\begin{array}{|l} \hline \text{possibleSubplans} : \text{Goal} \rightarrow \text{Plan} \rightarrow \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{Plan} \\ \hline \forall g : \text{Goal}; p : \text{Plan}; ps : \mathbb{P} \text{Plan} \bullet \text{possibleSubplans } g p ps = \\ \{p' : \text{Plan} \mid p' \in ps \wedge p'.\text{achieves} = g \wedge (\text{recursive } p p' = \text{false})\} \end{array}$$

$$\begin{array}{|l} \hline \text{possibleSubplansRatings} : \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{P} \mathbb{R} \\ \hline \forall ps : \mathbb{P} \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c, r : \mathbb{R} \\ \bullet \text{possibleSubplansRatings } ps ms w_s w_c = \\ \{r : \mathbb{R} \mid (\exists p : \text{Plan} \mid p \in ps \bullet \text{quality } p ms w_s w_c = r) \bullet r\} \end{array}$$

$$\begin{array}{|l} \hline \text{minRating} : \mathbb{P} \mathbb{R} \rightarrow \mathbb{R} \\ \hline \forall rs : \mathbb{P} \mathbb{R} \bullet (\exists r : \mathbb{R} \mid r \in rs \bullet \text{minRating } rs = r \wedge \\ (\forall r' : \mathbb{R} \mid r' \in rs \bullet r \leq r')) \end{array}$$

$$\begin{array}{|l} \hline \text{meanRating} : \mathbb{P} \mathbb{R} \rightarrow \mathbb{R} \\ \hline \forall rs : \mathbb{P} \mathbb{R} \bullet \text{meanRating } rs = \text{sumSet } rs / \#rs \end{array}$$

$$\begin{array}{|l} \hline \text{bcRating} : \text{Plan} \rightarrow \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\ \hline \forall p : \text{Plan}; pLib : \mathbb{P} \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c, r : \mathbb{R} \bullet \\ \text{plansubgoals } p = \emptyset \Leftrightarrow \text{bcRating } p pLib ms w_s w_c = \\ \text{quality } p ms w_s w_c \\ \forall p : \text{Plan}; pLib : \mathbb{P} \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c, r : \mathbb{R} \bullet \\ \text{plansubgoals } p \neq \emptyset \Leftrightarrow \text{bcRating } p pLib ms w_s w_c = \\ \text{quality } p ms w_s w_c + \text{sumSet } \{r : \mathbb{R}; g : \text{Goal} \mid \\ g \in \text{plansubgoals } p \wedge r = \text{minRating} (\\ \text{possibleSubplansRatings}(\text{possibleSubplans } g p pLib) \\ ms w_s w_c) \bullet r\} \end{array}$$

$$\begin{array}{l}
\overline{mcRating : Plan \rightarrow \mathbb{P} Plan \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}} \\
\forall p : Plan; pLib : \mathbb{P} Plan; ms : \mathbb{P} AgentModel; w_s, w_c, r : \mathbb{R} \bullet \\
\quad plansubgoals p = \emptyset \\
\quad \Leftrightarrow mcRating p pLib ms w_s w_c = quality p ms w_s w_c \\
\forall p : Plan; pLib : \mathbb{P} Plan; ms : \mathbb{P} AgentModel; w_s, w_c, r : \mathbb{R} \bullet \\
\quad plansubgoals p \neq \emptyset \\
\quad \Leftrightarrow mcRating p pLib ms w_s w_c = quality p ms w_s w_c + \\
\quad \quad sumSet \{r : \mathbb{R}; g : Goal \mid g \in plansubgoals p \wedge r = \\
\quad \quad \quad meanRating (possibleSubplansRatings(\\
\quad \quad \quad \quad possibleSubplans g p pLib) ms w_s w_c) \bullet r\}
\end{array}$$

A.24.1 Best-case and Mean-case Advantage

The balance between the *best-case* and *mean* rating amounts to a trade-off between an agent trying to find the best final plan and minimising the chance of the final plan being poor due to environmental change (in terms of these ratings). We define the *best-case advantage* of one plan over the other applicable plans to be advantage of that plan over others if its final elaboration has the best quality rating, and define *mean-case advantage* similarly.

$$\begin{array}{l}
\overline{maxRating : \mathbb{P} \mathbb{R} \rightarrow \mathbb{R}} \\
\forall rs : \mathbb{P} \mathbb{R} \bullet (\exists r : \mathbb{R} \mid r \in rs \bullet maxRating rs = r \wedge \\
\quad (\forall r' : \mathbb{R} \mid r' \in rs \bullet r \geq r'))
\end{array}$$

$$\begin{array}{l}
\overline{bca : \mathbb{P} Plan \rightarrow \mathbb{P} Plan \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}} \\
\forall ps, pLib : \mathbb{P} Plan; ms : \mathbb{P} AgentModel; rs : \mathbb{P} \mathbb{R}; w_s, w_c, r : \mathbb{R} \bullet \\
\quad bca ps pLib ms w_s w_c = r \\
\quad \Leftrightarrow rs = \{r' : \mathbb{R} \mid (\exists p : Plan \mid p \in ps \bullet r' = \\
\quad \quad bcRating p pLib ms w_s w_c)\} \wedge \\
\quad \quad r = maxRating rs - minRating rs
\end{array}$$

$$\begin{array}{l}
\overline{mca : \mathbb{P} Plan \rightarrow \mathbb{P} Plan \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}} \\
\forall ps, pLib : \mathbb{P} Plan; ms : \mathbb{P} AgentModel; rs : \mathbb{P} \mathbb{R}; w_s, w_c, r : \mathbb{R} \bullet \\
\quad mca ps pLib ms w_s w_c = r \\
\quad \Leftrightarrow rs = \{r' : \mathbb{R} \mid (\exists p : Plan \mid p \in ps \bullet r' \\
\quad \quad = mcRating p pLib ms w_s w_c)\} \wedge \\
\quad \quad r = maxRating rs - minRating rs
\end{array}$$

A.24.2 Recursion

Where a plan is recursive it is not possible to obtain a rating for a subplan to feed into a higher level plan with respect to which it is recursive. Our solution to this is to use domain specific knowledge to estimate the limit of the recursion.

$$\begin{array}{l}
 \text{existsRecursiveElaboration} : \text{Plan} \rightarrow \mathbb{P} \text{Plan} \rightarrow \text{bool} \\
 \hline
 \text{scaleForRecursion} : \text{Plan} \rightarrow \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\
 \forall p : \text{Plan}; pLib : \mathbb{P} \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c, r, m : \mathbb{R} \bullet \\
 \quad \text{existsRecursiveElaboration } p \text{ } pLib = \text{false} \\
 \quad \Leftrightarrow \text{scaleForRecursion } p \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \text{ } m = \\
 \quad \quad bcRating \text{ } p \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \wedge \\
 \quad \text{existsRecursiveElaboration } p \text{ } pLib = \text{true} \\
 \quad \Leftrightarrow \text{scaleForRecursion } p \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \text{ } m = \\
 \quad \quad bcRating \text{ } p \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c * m
 \end{array}$$

A.24.3 Selecting Between Partial Plans

In selecting a plan, the advantage should be maximised, regardless of whether it is best-case or mean-case. The following functions, *useBCA* and *useMCA*, formalise this and specify the conditions under which the best-case and mean-case ratings should be used (i.e. whichever offers the greater advantage).

$$\begin{array}{l}
 \text{useBCA} : \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{bool} \\
 \hline
 \forall ps, pLib : \mathbb{P} \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c : \mathbb{R} \bullet \\
 \quad \text{useBCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{true} \\
 \quad \Leftrightarrow bca \text{ } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c > mca \text{ } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \wedge \\
 \quad \text{useBCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{false} \\
 \quad \Leftrightarrow bca \text{ } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c < mca \text{ } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \\
 \hline
 \text{useMCA} : \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{Plan} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{bool} \\
 \hline
 \forall ps, pLib : \mathbb{P} \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c : \mathbb{R} \bullet \\
 \quad \text{useMCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{true} \\
 \quad \Leftrightarrow mca \text{ } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c > bca \text{ } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c \\
 \forall ps, pLib : \mathbb{P} \text{Plan}; ms : \mathbb{P} \text{AgentModel}; w_s, w_c : \mathbb{R} \bullet \\
 \quad \text{useMCA } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c = \text{false} \\
 \quad \Leftrightarrow mca \text{ } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c < bca \text{ } ps \text{ } pLib \text{ } ms \text{ } w_s \text{ } w_c
 \end{array}$$

Once the mean-case and best-case advantages have been considered to decide which criteria to use for plan selection, a plan can be chosen using that criteria.

$$\begin{array}{|l}
\hline
\textit{selectByBCA} : \mathbb{P} \textit{Plan} \rightarrow \mathbb{P} \textit{Plan} \rightarrow \mathbb{P} \textit{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \textit{Plan} \\
\hline
\forall ps, pLib : \mathbb{P} \textit{Plan}; ms : \mathbb{P} \textit{AgentModel}; w_s, w_c : \mathbb{R}; chosen : \textit{Plan} \bullet \\
\textit{selectByBCA} ps pLib ms w_s w_c = chosen \\
\Leftrightarrow (\forall p' : \textit{Plan} \mid p' \in ps \bullet bcRating chosen pLib ms w_s w_c > \\
bcRating p' pLib ms w_s w_c) \\
\hline
\textit{selectByMCA} : \mathbb{P} \textit{Plan} \rightarrow \mathbb{P} \textit{Plan} \rightarrow \mathbb{P} \textit{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \textit{Plan} \\
\hline
\forall ps, pLib : \mathbb{P} \textit{Plan}; ms : \mathbb{P} \textit{AgentModel}; w_s, w_c : \mathbb{R}; chosen : \textit{Plan} \bullet \\
\textit{selectByMCA} ps pLib ms w_s w_c = chosen \\
\Leftrightarrow (\forall p' : \textit{Plan} \mid p' \in ps \bullet mcRating chosen pLib ms w_s w_c > \\
mcRating p' pLib ms w_s w_c) \\
\hline
\textit{selectBestPlan} : \mathbb{P} \textit{Plan} \rightarrow \mathbb{P} \textit{Plan} \rightarrow \mathbb{P} \textit{AgentModel} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \textit{Plan} \\
\hline
\forall ps, pLib : \mathbb{P} \textit{Plan}; ms : \mathbb{P} \textit{AgentModel}; w_s, w_c : \mathbb{R}; chosen : \textit{Plan} \bullet \\
\textit{selectBestPlan} ps pLib ms w_s w_c = chosen \Leftrightarrow \\
(\textit{useBCA} ps pLib ms w_s w_c = \textit{true} \wedge \\
chosen = \textit{selectByBCA} ps pLib ms w_s w_c) \vee \\
(\textit{useMCA} ps pLib ms w_s w_c = \textit{true} \wedge \\
chosen = \textit{selectByMCA} ps pLib ms w_s w_c)
\end{array}$$

The plan selection mechanism described in this chapter provides an instantiation of the function *planForGoal* introduced in Section A.19.

$$\begin{array}{|l}
\hline
\textit{planForGoal} : \mathbb{P} \textit{Belief} \rightarrow \mathbb{P} \textit{Intention} \rightarrow \mathbb{P} \textit{Plan} \rightarrow \textit{Goal} \rightarrow \textit{Plan} \\
\hline
\forall bel : \mathbb{P} \textit{Belief}; I : \mathbb{P} \textit{Intention}; plib : \mathbb{P} \textit{Plan}; \\
g : \textit{Goal}; w_s, w_c : \mathbb{R}; p : \textit{Plan} \bullet \textit{planForGoal} bel I plib g = p \\
\Leftrightarrow \textit{selectBestPlan} (\textit{planSetForGoal} g bel plib) plib \\
(\textit{extractAllModels} bel) w_s w_c = p
\end{array}$$

A.25 Cooperative Intention

In this section we specify the notion of a cooperative intention — a group’s commitment to a particular course of action. A *convention* specifies the conditions under which a commitment can be abandoned, and how an agent should behave in such a circumstance. A

cooperative intention is specified to contain the goal and plan to which the commitment is towards, the identifiers of the agents who have the commitment, and a set of conventions defining the duration of this commitment.

$$\text{Convention} == \mathbb{P} \text{Belief} \times \text{Goal}$$

$$\text{believes} : \text{Agent} \rightarrow \mathbb{P} \text{Belief} \rightarrow \text{bool}$$

$$\begin{aligned} & \forall ag : \text{Agent}; sit : \mathbb{P} \text{Belief} \bullet \\ & \quad (\forall b : \text{Belief} \bullet b \in sit \wedge b \in ag.\text{beliefs} \Rightarrow \text{believes } ag \text{ } sit = \text{true}) \\ & \forall ag : \text{Agent}; sit : \mathbb{P} \text{Belief} \bullet \\ & \quad (\exists b : \text{Belief} \bullet b \in sit \wedge b \notin ag.\text{beliefs} \Rightarrow \text{believes } ag \text{ } sit = \text{false}) \end{aligned}$$

CooperativeIntention

goal : Goal

plan : Plan

agents : $\mathbb{P} \text{AgentID}$

conventions : $\mathbb{P} \text{Convention}$

$$\begin{aligned} & \forall id : \text{AgentID} \mid id \in \text{agents} \bullet \\ & \quad (\exists ag : \text{Agent} \bullet ag.\text{agentID} = id \wedge goal \in ag.\text{goals} \wedge \\ & \quad \quad (\exists i : \text{Intention} \mid i \in ag.\text{intentions} \bullet \\ & \quad \quad \quad i.\text{plans } 1 = \text{plan} \wedge i.\text{satisfies} = \text{goal})) \vee \\ & \quad (\exists ag : \text{Agent} \bullet ag.\text{agentID} = id \wedge (\exists c : \text{Convention} \mid \\ & \quad \quad c \in \text{conventions} \bullet \text{believes } ag \text{ } (\text{first } c) = \text{true} \wedge \\ & \quad \quad \quad \text{second } c \in ag.\text{goals} \wedge (\exists i : \text{Intention} \mid i \in ag.\text{intentions} \bullet \\ & \quad \quad \quad \quad i.\text{satisfies} = \text{second } c))) \end{aligned}$$

A.26 Cooperative Plans

In this section we formalise the circumstances in which an agent might decide to allocate an action in a plan to another agent, when that action is within the agent's capabilities.

$$\mid \text{costC} : \text{Contribution} \rightarrow \mathbb{R}$$

$$\text{allocateContribution} : \text{Contribution} \rightarrow \mathbb{P} \text{Belief} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{bool}$$

$$\begin{aligned} & \forall c : \text{Contribution}; \text{bel} : \mathbb{P} \text{Belief}; w_c, w_r : \mathbb{R} \bullet \\ & \quad w_c * \text{cost}C \ c > \text{risk}C \ c \ (\text{extractAllModels} \ \text{bel}) * w_r \\ & \quad \Rightarrow \text{allocateContribution} \ c \ \text{bel} \ w_c \ w_r = \text{true} \wedge \\ & \quad w_c * \text{cost}C \ c < \text{risk}C \ c \ (\text{extractAllModels} \ \text{bel}) * w_r \\ & \quad \Rightarrow \text{allocateContribution} \ c \ \text{bel} \ w_c \ w_r = \text{false} \end{aligned}$$

$$\text{isCooperativePlan} : \text{Agent} \rightarrow \text{Plan} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \text{bool}$$

$$\begin{aligned} & \forall ag : \text{Agent}; p : \text{Plan}; w_c, w_r : \mathbb{R} \bullet \\ & \quad (\exists c : \text{Contribution} \bullet c \in \text{plancontributions} \ p \wedge c \notin \text{ag.capabilities}) \vee \\ & \quad (\exists c : \text{Contribution} \bullet c \in \text{plancontributions} \ p \wedge \\ & \quad \quad \text{allocateContribution} \ c \ \text{ag.beliefs} \ w_c \ w_r = \text{true}) \vee \\ & \quad (\exists cs : \mathbb{P} \text{Contribution} \bullet (\text{Joint} \ cs) \in \text{ran} \ p.\text{body}) \vee \\ & \quad (\exists cas : \mathbb{P} \text{CAcomponent} \bullet (\text{Concurrent} \ cas) \in \text{ran} \ p.\text{body}) \\ & \quad \Rightarrow \text{isCooperativePlan} \ ag \ p \ w_c \ w_r = \text{true} \\ & \forall ag : \text{Agent}; p : \text{Plan}; w_c, w_r : \mathbb{R} \bullet \\ & \quad (\forall c : \text{Contribution} \bullet c \in \text{plancontributions} \ p \wedge c \in \text{ag.capabilities}) \wedge \\ & \quad (\forall c : \text{Contribution} \bullet c \in \text{plancontributions} \ p \wedge \\ & \quad \quad \text{allocateContribution} \ c \ \text{ag.beliefs} \ w_c \ w_r = \text{false}) \wedge \\ & \quad (\forall cs : \mathbb{P} \text{Contribution} \bullet (\text{Joint} \ cs) \notin \text{ran} \ p.\text{body}) \wedge \\ & \quad (\forall cas : \mathbb{P} \text{CAcomponent} \bullet (\text{Concurrent} \ cas) \notin \text{ran} \ p.\text{body}) \\ & \quad \Rightarrow \text{isCooperativePlan} \ ag \ p \ w_c \ w_r = \text{false} \end{aligned}$$

A.27 Plan Annotation

In order to determine which agents to ask for cooperation, the initiating agent must consider each of the contributions in its plan and determine which is the best agent to perform it. The chosen agent is associated with the contribution by *annotating* the contribution with the identifier of that agent. In this section we consider the annotation of the different action types that a plan can contain, along with the annotation of a complete plan.

A.27.1 Action Annotation

The simplest action type to annotate is an individual contribution; an agent should be considered for annotation to a contribution if it is trusted above a *minimum trust threshold*, as

specified below.

$$\begin{array}{l}
 \text{considerForAnnotation} : \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{R} \rightarrow \text{bool} \\
 \hline
 \forall id : \text{AgentID}; ms : \mathbb{P} \text{AgentModel}; t : \mathbb{R} \bullet \\
 \quad (\text{trustOfAgent } id \ ms \ > \ t) \Rightarrow \text{considerForAnnotation } id \ ms \ t = \text{true} \wedge \\
 \quad (\text{trustOfAgent } id \ ms \leq \ t) \Rightarrow \text{considerForAnnotation } id \ ms \ t = \text{false}
 \end{array}$$

AnnotatedContribution

$$\begin{array}{l}
 \text{symbol} : \text{ActSym} \\
 \text{terms} : \text{seq Term} \\
 \text{agents} : \mathbb{P} \text{AgentID}
 \end{array}$$

AnnotateContribution

$$\begin{array}{l}
 \text{annotateContribution} : \text{Contribution} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \\
 \rightarrow \text{AnnotatedContribution}
 \end{array}$$

$$\begin{array}{l}
 \forall c : \text{Contribution}; ms : \mathbb{P} \text{AgentModel}; n : \mathbb{Z}; t : \mathbb{R}; \\
 \quad ac : \text{AnnotatedContribution} \bullet c.\text{symbol} = ac.\text{symbol} \wedge \\
 \quad c.\text{terms} = ac.\text{terms} \wedge ac.\text{agents} = \{id : \text{AgentID} \mid \\
 \quad \quad id \in \text{ran}(\{i : \mathbb{Z} \mid i \leq n\} \mid \text{orderedCapableAgents } c \ ms) \wedge \\
 \quad \quad \text{considerForAnnotation } id \ ms \ t = \text{true} \bullet id\}
 \end{array}$$

A.27.2 Joint Action Annotation

A joint action comprises a set of individual contributions, and so an annotated joint action, *AnnotatedJointAction*, is simply defined to be a set of annotated contributions. To annotate a joint action, the agent must determine which sets of agents are capable of performing it, and then select the most trusted, as formalised below.

AnnotatedJointAction

$$\text{contributions} : \mathbb{P} \text{AnnotatedContribution}$$

$$\# \text{contributions} \geq 2$$

ValidityCheckJointAction

$validAssignments : \mathbb{P}(\mathbb{P}(Contribution \times AgentID))$
 $\rightarrow \mathbb{P}(\mathbb{P}(Contribution \times AgentID))$
 $isValid : \mathbb{P}(Contribution \times AgentID) \rightarrow bool$

$\forall annotations : \mathbb{P}(\mathbb{P}(Contribution \times AgentID)) \bullet$
 $validAssignments\ annotations =$
 $\{a : \mathbb{P}(Contribution \times AgentID) \mid a \in annotations \wedge$
 $isValid\ a = true \bullet a\}$
 $\forall annotation : \mathbb{P}(Contribution \times AgentID) \bullet$
 $(\forall cid : (Contribution \times AgentID) \mid cid \in annotation \bullet$
 $(\forall cid' : (Contribution \times AgentID) \mid cid' \in annotation \wedge$
 $cid \neq cid' \bullet second\ cid \neq second\ cid'$
 $\Rightarrow isValid\ annotation = true))$
 $\forall annotation : \mathbb{P}(Contribution \times AgentID) \bullet$
 $(\forall cid : (Contribution \times AgentID) \mid cid \in annotation \bullet$
 $(\exists cid' : (Contribution \times AgentID) \mid cid' \in annotation \wedge$
 $cid \neq cid' \bullet second\ cid = second\ cid'$
 $\Rightarrow isValid\ annotation = false))$

ExtractInfoJointAction

$extractAgents : \mathbb{P}(Contribution \times AgentID) \rightarrow \mathbb{P}\ AgentID$
 $extractContributions : \mathbb{P}(Contribution \times AgentID) \rightarrow \mathbb{P}\ Contribution$
 $agentsOfContributionA : \mathbb{P}(Contribution \times AgentID) \rightarrow Contribution$
 $\rightarrow \mathbb{P}\ AgentID$
 $agentsOfContributionAs : \mathbb{P}(\mathbb{P}(Contribution \times AgentID)) \rightarrow Contribution$
 $\rightarrow \mathbb{P}\ AgentID$

$\forall annotation : (\mathbb{P}(Contribution \times AgentID)) \bullet extractAgents\ annotation =$
 $\{a : (Contribution \times AgentID) \mid a \in annotation \bullet second\ a\}$
 $\forall annotation : (\mathbb{P}(Contribution \times AgentID)) \bullet$
 $extractContributions\ annotation =$
 $\{a : (Contribution \times AgentID) \mid a \in annotation \bullet first\ a\}$
 $\forall annotation : \mathbb{P}(Contribution \times AgentID); c : Contribution \bullet$
 $agentsOfContributionA\ annotation\ c =$
 $\{cid : (Contribution \times AgentID) \mid cid \in annotation \wedge$
 $first\ cid = c \bullet second\ cid\}$
 $\forall annotations : \mathbb{P}(\mathbb{P}(Contribution \times AgentID)); c : Contribution \bullet$
 $agentsOfContributionAs\ annotations\ c =$
 $\bigcup\{annotation : \mathbb{P}(Contribution \times AgentID) \mid$
 $annotation \in annotations \bullet$
 $agentsOfContributionA\ annotation\ c\}$

GenerateAnnotationsJointAction

\exists *ExtractInfoJointAction*

$orderedAnnotations : \mathbb{P}(\mathbb{P}(Contribution \times AgentID)) \rightarrow \mathbb{P} AgentModel$
 $\rightarrow seq(\mathbb{P}(Contribution \times AgentID))$
 $allPossibleAnnotations : seq Contribution \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R}$
 $\rightarrow \mathbb{P}(\mathbb{P}(Contribution \times AgentID))$
 $combine : \mathbb{P}(\mathbb{P}(Contribution \times AgentID)) \rightarrow \mathbb{P}(Contribution \times AgentID)$
 $\rightarrow \mathbb{P}(\mathbb{P}(Contribution \times AgentID))$
 $contribSeq : JointAction \rightarrow seq Contribution$

$\forall unordered : \mathbb{P}(\mathbb{P}(Contribution \times AgentID)); ms : \mathbb{P} AgentModel;$

$ordered : seq(\mathbb{P}(Contribution \times AgentID)) \bullet$

$orderedAnnotations unordered ms = ordered \wedge$

$ran ordered = unordered \wedge$

$(\forall n : \mathbb{Z} \mid n \geq 2 \wedge n \leq \#ordered \bullet$

$trustOfAgentSet (extractAgents (orderedn)) ms$

$> trustOfAgentSet (extractAgents$

$(ordered(n - 1))) ms$

$\forall s : seq Contribution; ms : \mathbb{P} AgentModel; t : \mathbb{R} \bullet$

$(s = \langle \rangle \wedge allPossibleAnnotations s ms t =) \vee$

$(s \neq \langle \rangle) \wedge allPossibleAnnotations s ms t = combine$

$(allPossibleAnnotations (tails) ms t)$

$\{c : Contribution; ag : AgentID \mid c = head s \wedge$

$ag \in capableAgents c ms \wedge$

$considerForAnnotation ag ms t = true \bullet (c, ag)\}$

$\forall s1 : \mathbb{P}(\mathbb{P}(Contribution \times AgentID)); s2 : \mathbb{P}(Contribution \times AgentID) \bullet$

$combine s1 s2 = \{working : \mathbb{P}(Contribution \times AgentID);$

$new : (Contribution \times AgentID) \mid working \in s1 \wedge new \in s2 \bullet$

$(working \cup \{new\})\}$

$\forall ja : JointAction \bullet \#(contribSeq ja) =$

$\#ja.contributions \wedge ran(contribSeq ja) = ja.contributions$

AnnotateJointActionAuxiliary

\exists *ValidityCheckJointAction*

\exists *GenerateAnnotationsJointAction*

\exists *ExtractInfoJointAction*

AnnotateJointAction

\exists *AnnotateJointActionAuxiliary*

annotateJointAction : *JointAction* \rightarrow \mathbb{P} *AgentModel* \rightarrow $\mathbb{Z} \rightarrow \mathbb{R}$
 \rightarrow *AnnotatedJointAction*

annotate : $\text{seq}(\mathbb{P}(\text{Contribution} \times \text{AgentID})) \rightarrow \text{AnnotatedJointAction}$

$\forall ja : \text{JointAction}; ms : \mathbb{P} \text{AgentModel}; n : \mathbb{Z}; t : \mathbb{R} \bullet$

annotateJointAction *ja ms n t* = *annotate* ($\{i : \mathbb{Z} \mid i \leq n\}$ |
orderedAnnotations (*validAssignments* (*allPossibleAnnotations*
(contribSeq ja) ms t)) *ms*)

$\forall s : \text{seq}(\mathbb{P}(\text{Contribution} \times \text{AgentID})); aja : \text{AnnotatedJointAction} \mid$

annotate s = *aja* \bullet $\forall c : \text{Contribution} \bullet c \in \text{extractContributions } (s \ 1)$

$\Leftrightarrow (\exists_1 ac : \text{AnnotatedContribution} \bullet ac \in \text{aja.contributions} \wedge$
 $c.\text{symbol} = ac.\text{symbol} \wedge c.\text{terms} = ac.\text{terms} \wedge$
 $ac.\text{agents} = \text{agentsOfContributionsAs } (\text{ran } s) \ c)$

A.27.3 Concurrent Action Annotation

In a similar manner, we can specify that an annotated concurrent action comprises a set of annotated contributions and joint actions. Annotation of a concurrent action involves annotating each of its components, and including the result in the annotated concurrent action.

ACComponent ::= *AContrib* $\langle\langle$ *AnnotatedContribution* $\rangle\rangle$
 \mid *AJA* $\langle\langle$ \mathbb{P} *AnnotatedContribution* $\rangle\rangle$

AnnotatedConcurrentAction

contributions : \mathbb{P} *ACComponent*

$\#$ *contributions* ≥ 2

AnnotateConcurrentAction

\exists *AnnotateContribution*

\exists *AnnotateJointAction*

annotateConcurrentAction : *ConcurrentAction* \rightarrow \mathbb{P} *AgentModel* \rightarrow \mathbb{Z} \rightarrow \mathbb{R}
 \rightarrow *AnnotatedConcurrentAction*

annotateCAcomponent : *CAcomponent* \rightarrow \mathbb{P} *AgentModel* \rightarrow \mathbb{Z} \rightarrow \mathbb{R}
 \rightarrow *ACAcomponent*

\forall *cac* : *CAcomponent*; *ms* : \mathbb{P} *AgentModel*; *n* : \mathbb{Z} ; *t* : \mathbb{R} ;

acac : *ACAcomponent* \bullet *annotateCAcomponent cac ms n t* = *acac*

$\Leftrightarrow (\exists c$: *Contribution* | *Contrib*(*c*) = *cac* \bullet

acac = *AContrib*(*annotateContribution c ms n t*) \vee

$(\exists cs$: \mathbb{P} *Contribution*; *ja* : *JointAction* | *JA*(*cs*) = *cac* \bullet

ja.contributions = *cs* \wedge *acac* =

AJA((*annotateJointAction ja ms n t*).*contributions*))

\forall *ca* : *ConcurrentAction*; *ms* : \mathbb{P} *AgentModel*; *n* : \mathbb{Z} ; *t* : \mathbb{R} ;

aca : *AnnotatedConcurrentAction* \bullet

annotateConcurrentAction ca ms n t = *aca* \wedge

aca.contributions = {*cac* : *CAcomponent* |

cac \in *ca.contributions* \bullet

annotateCAcomponent cac ms n t}

A.27.4 Annotated Plans

The notion of an annotated plan is formalised below in the schema *AnnotatedPlan*, in which all contributions are annotated with a *set* of agents. Each contribution is annotated with a set, rather than the individual agent that will execute it since, at this stage, the annotation represents the agents to request assistance from.

APlanStep ::= *AIndividual* $\langle\langle$ *AnnotatedContribution* $\rangle\rangle$
| *AJoint* $\langle\langle$ \mathbb{P} *AnnotatedContribution* $\rangle\rangle$
| *AConcurrent* $\langle\langle$ \mathbb{P} *ACAcomponent* $\rangle\rangle$
| *ASubgoal* $\langle\langle$ *Goal* $\rangle\rangle$

AnnotatedPlan

achieves : *Goal*

preconditions : \mathbb{P} *Literal*

body : seq *APlanStep*

AnnotatePlan

\exists *AnnotateContribution*

\exists *AnnotateJointAction*

\exists *AnnotateConcurrentAction*

annotatePlan : *Plan* \rightarrow \mathbb{P} *AgentModel* \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \rightarrow *AnnotatedPlan* \bullet

annotateStep : *PlanStep* \rightarrow \mathbb{P} *AgentModel* \rightarrow \mathbb{Z} \rightarrow \mathbb{R} \rightarrow *APlanStep* \bullet

$\forall p : \text{Plan}; ms : \mathbb{P} \text{AgentModel}; n : \mathbb{Z}; t : \mathbb{R}; ap : \text{AnnotatedPlan} \bullet$

annotatePlan *p ms n t* = *ap* \Leftrightarrow *p.achieves* = *ap.achieves* \wedge

p.preconditions = *ap.preconditions* \wedge

$(\forall n : \mathbb{Z} \mid n \leq \#p.body \bullet ap.body\ n =$

annotateStep (*p.body n*) *ms n t*)

$\forall ps : \text{PlanStep}; ms : \mathbb{P} \text{AgentModel}; n : \mathbb{Z}; t : \mathbb{R}; aps : \text{APlanStep} \bullet$

annotateStep *ps ms n t* = *aps*

$\Leftrightarrow (\exists c : \text{Contribution} \bullet \text{Individual}(c) = ps \wedge aps =$

AIndividual(*annotateContribution* *c ms n t*) \vee

$(\exists cs : \mathbb{P} \text{Contribution}; ja : \text{JointAction} \mid ja.contributions = cs \bullet$

Joint(*cs*) = *ps* \wedge *aps* = *AJoint*(

annotateJointAction *ja ms n t*).*contributions*) \vee

$(\exists cac : \mathbb{P} \text{CAcomponent}; ca : \text{ConcurrentAction} \mid$

ca.contributions = *cac* \bullet *Concurrent*(*cac*) = *ps* \wedge *aps* =

AConcurrent(*annotateConcurrentAction*

ca ms n t).*contributions*) \vee

$(\exists g : \text{Goal} \bullet \text{Subgoal}(g) = ps \wedge aps = \text{ASubgoal}(g))$

A.28 Soliciting Commitment to Cooperate

After annotating its plan, an agent must request assistance from the annotated agents, and there are four possible types of request, each giving different degrees of information to the participants.

RequestActions

to : *AgentID*

from : *AgentID*

contributions : \mathbb{P} *Contribution*

RequestGoalActions

to : *AgentID*
from : *AgentID*
goal : *Goal*
contributions : \mathbb{P} *Contribution*

RequestPartiallyAnnotatedPlan

to : *AgentID*
from : *AgentID*
plan : *AnnotatedPlan*

RequestAnnotatedPlan

to : *AgentID*
from : *AgentID*
plan : *AnnotatedPlan*

Request ::= *ActionRequest*(*RequestActions*)
| *GoalActionRequest*(*RequestGoalActions*)
| *PartiallyAnnotatedPlanRequest*(*RequestPartiallyAnnotatedPlan*)
| *AnnotatedPlanRequest*(*RequestAnnotatedPlan*)

As discussed in Section 7.6 an agent's request for assistance should include only the actions for which it needs help if it believes that the goal is of zero or negative motivational value to the provider. This is specified in the schema *UseActionRequest* below, which takes an annotated plan and set of agent models, and returns true if the goal is believed to be of zero or negative motivational value to one or more of the agents being requested, and the agent's request should be of this form.

AnnotatedPlanAuxiliary

$believedMV : Goal \rightarrow AgentID \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R}$

$extractAgentsACacomponent : ACacomponent$

$\rightarrow \mathbb{P}(Contribution \times AgentID)$

$extractAgentsStep : APlanStep \rightarrow \mathbb{P}(Contribution \times AgentID)$

$extractAgentsPlan : AnnotatedPlan \rightarrow \mathbb{P}(Contribution \times AgentID)$

$allAgents : AnnotatedPlan \rightarrow \mathbb{P} AgentID$

$agentContributions : AgentID \rightarrow \mathbb{P}(Contribution \times AgentID)$

$\rightarrow \mathbb{P} Contribution$

$\forall acac : ACacomponent \bullet$

$(\exists ac : AnnotatedContribution \mid AContrib(ac) = acac \bullet$

$extractAgentsACacomponent acac =$

$\{id : AgentID; c : Contribution \mid id \in ac.agents$

$\wedge c.symbol = ac.symbol \wedge c.terms = ac.terms \bullet (c, id)\}) \vee$

$(\exists acs : \mathbb{P} AnnotatedContribution \mid AJA(acs) = acac \bullet$

$extractAgentsACacomponent acac =$

$\{ac : AnnotatedContribution; id : AgentID; c : Contribution \mid$

$ac \in acs \wedge id \in ac.agents \wedge c.symbol = ac.symbol \wedge$

$c.terms = ac.terms \bullet (c, id)\})$

$\forall aps : APlanStep \bullet$

$(\exists ac : AnnotatedContribution \mid AIndividual(ac) = aps \bullet$

$extractAgentsStep aps =$

$\{id : AgentID; c : Contribution \mid id \in ac.agents$

$\wedge c.symbol = ac.symbol \wedge c.terms = ac.terms \bullet (c, id)\}) \vee$

$(\exists acs : \mathbb{P} AnnotatedContribution \mid AJoint(acs) = aps \bullet$

$extractAgentsStep aps =$

$\{ac : AnnotatedContribution; id : AgentID; c : Contribution \mid$

$ac \in acs \wedge id \in ac.agents \wedge c.symbol = ac.symbol \wedge$

$c.terms = ac.terms \bullet (c, id)\}) \vee$

$(\exists acacs : \mathbb{P} ACacomponent \mid AConcurrent(acacs) = aps \bullet$

$extractAgentsStep aps =$

$\bigcup \{acac : ACacomponent \mid acac \in acacs \bullet$

$extractAgentsACacomponent acac\})$

$\forall ap : AnnotatedPlan \bullet extractAgentsPlan ap =$

$\bigcup \{aps : APlanStep \mid aps \in \text{ran } ap.body \bullet extractAgentsStep aps\}$

$\forall id : AgentID; cids : \mathbb{P}(Contribution \times AgentID) \bullet$

$agentContributions id cids = \{c : Contribution \mid (c, id) \in cids \bullet c\}$

$\forall ap : AnnotatedPlan \bullet allAgents ap =$

$\{id : AgentID; cid : (Contribution \times AgentID) \mid$

$cid \in extractAgentsPlan ap \wedge id = \text{second } cid \bullet id\}$

UseActionRequest

\exists *AnnotatedPlanAuxiliary*

useActionRequest : *AnnotatedPlan* \rightarrow \mathbb{P} *AgentModel* \rightarrow *bool*

\forall *ap* : *AnnotatedPlan*; *ms* : \mathbb{P} *AgentModel* •

useActionRequest ap ms = true

$\Leftrightarrow (\exists id : AgentID \mid id \in allAgents\ ap \bullet$
believedMV ap.achieves id ms $\leq 0) \wedge$

useActionRequest ap ms = false

$\Leftrightarrow (\forall id : AgentID \mid id \in allAgents\ ap \bullet$
believedMV ap.achieves id ms $> 0)$

A.29 Nominal Commitment

The process of forming a cooperative intention requires agents to form a nominal commitment to informing others if they agree to cooperate, only later change their decision. Before requesting assistance a nominal commitment is formed with respect to the agents whose assistance is sought.

NominalCommitment

plan : *AnnotatedPlan*

agents : \mathbb{P} *AgentID*

FormNominalCommitment

Δ Agent

\exists AnnotatedPlanAuxiliary

newPlan : AnnotatedPlan

committedTo : AnnotatedPlan \rightarrow bool

$\forall ap : \text{AnnotatedPlan} \bullet \exists c : \text{NominalCommitment} \mid$

$c \in \text{nominalCommitments} \bullet c.\text{plan.achieves} = ap.\text{achieves}$

$\Leftrightarrow \text{committedTo } ap = \text{true}$

$\forall ap : \text{AnnotatedPlan} \bullet \forall c : \text{NominalCommitment} \mid$

$c \in \text{nominalCommitments} \bullet c.\text{plan.achieves} \neq ap.\text{achieves}$

$\Leftrightarrow \text{committedTo } ap = \text{false}$

committedTo newPlan = false

$\Rightarrow (\exists c : \text{NominalCommitment} \bullet c.\text{plan} = \text{newPlan} \wedge$

$c.\text{agents} = \text{allAgents newPlan} \wedge$

$\text{nominalCommitments}' = \text{nominalCommitments} \cup \{c\})$

committedTo newPlan = true

$\Rightarrow (\exists c, c' : \text{NominalCommitment} \mid c \in \text{nominalCommitments} \bullet$

$c.\text{plan.achieves} = \text{newPlan.achieves} \wedge$

$c \notin \text{nominalCommitments}' \wedge c' \in \text{nominalCommitments}' \wedge$

$c'.\text{plan} = \text{newPlan} \wedge c'.\text{agents} = \text{allAgents newPlan})$

A.30 Committing to Cooperate

On receiving a request for assistance an agent decides whether or not to cooperate, based on the trust ascribed to the requester, which determines the perceived risk of interacting with it and the motivational value that would be attained (by the providing agent) in cooperating.

We formalise these factors below.

consideredTrusted : AgentID \rightarrow \mathbb{P} AgentModel \rightarrow \mathbb{R} \rightarrow bool

$\forall \text{requester} : \text{AgentID}; ms : \mathbb{P} \text{AgentModel}; t : \mathbb{R} \bullet$

trustOfAgent requester ms $> t$

$\Rightarrow \text{consideredTrusted requester ms } t = \text{true} \wedge$

trustOfAgent requester ms $\leq t$

$\Rightarrow \text{consideredTrusted requester ms } t = \text{false}$

ConsiderFurtherContribution

\exists Agent

$considerFurtherContribution : Contribution \rightarrow AgentID \rightarrow \mathbb{R} \rightarrow bool$

$\forall c : Contribution; requester : AgentID; t : \mathbb{R} \bullet$

$considerFurtherContribution c requester t = true$

$\Leftrightarrow consideredTrusted requester (extractAllModels beliefs) t = true \wedge (\exists m : Motivation \mid m \in motivations \bullet mvContribution m c > 0) \wedge$

$considerFurtherContribution c requester t = false$

$\Leftrightarrow consideredTrusted requester (extractAllModels beliefs) t = false \vee (\forall m : Motivation \mid m \in motivations \bullet mvContribution m c \leq 0)$

ConsiderFurtherContributionGoal

\exists Agent

$considerFurtherContributionGoal : Contribution \rightarrow Goal \rightarrow AgentID \rightarrow \mathbb{R} \rightarrow bool$

$\forall c : Contribution; g : Goal; requester : AgentID; t : \mathbb{R} \bullet$

$considerFurtherContributionGoal c g requester t = true$

$\Leftrightarrow consideredTrusted requester (extractAllModels beliefs) t = true \wedge (\exists m : Motivation \mid m \in motivations \bullet mvContribution m c > 0) \wedge (\exists m : Motivation \mid m \in motivations \bullet mitigation m g > 0) \wedge$

$considerFurtherContributionGoal c g requester t = false$

$\Leftrightarrow consideredTrusted requester (extractAllModels beliefs) t = false \vee (\forall m : Motivation \mid m \in motivations \bullet mvContribution m c \leq 0) \wedge (\forall m : Motivation \mid m \in motivations \bullet mitigation m g \leq 0)$

ConsiderFurtherPlan

$\exists Agent$

$considerFurtherPlan : Plan \rightarrow AgentID \rightarrow \mathbb{R} \rightarrow bool$

$\forall p : Plan; requester : AgentID; t : \mathbb{R} \bullet$

$considerFurtherPlan p requester t = true \Leftrightarrow$

$consideredTrusted requester (extractAllModels beliefs) t =$
 $true \wedge (\exists m : Motivation \mid m \in motivations \bullet$
 $mvPlan m p > 0) \wedge$

$considerFurtherPlan p requester t = false \Leftrightarrow$

$consideredTrusted requester (extractAllModels beliefs) t =$
 $false \vee (\forall m : Motivation \mid m \in motivations \bullet$
 $mvPlan m p \leq 0)$

A.31 Generating Full Commitment to Cooperation

Each of the requested agents makes an appropriate response, either accepting or declining to cooperate. Then, if sufficient agents accept, a cooperative intention can be formed.

$Response == AgentID \times Contribution \times bool$

CheckResponses

$\exists UseActionRequest$

$\exists ExtractInfoJointAction$

$checkResponses : AnnotatedPlan \rightarrow \mathbb{P} Response \rightarrow bool$

$\forall ap : AnnotatedPlan; rs : \mathbb{P} Response; cid : \mathbb{P}(Contribution \times AgentID) \mid$

$cid = extractAgentsPlan ap \bullet checkResponses ap rs = true$

$\Leftrightarrow (\forall c : Contribution \mid c \in (extractContributions cid) \bullet$
 $(\exists id : AgentID \bullet (id, c, true) \in rs)) \wedge$

$checkResponses ap rs = false$

$\Leftrightarrow (\forall c : Contribution \mid c \in (extractContributions cid) \bullet$
 $(\forall id : AgentID \bullet (id, c, true) \notin rs))$

If sufficient agents accept, the initiating agent selects the particular agents to cooperate with (since the plan may have been redundantly annotated) and sends a confirmation, and a cooperative intention is formed.

Confirmation

to : AgentID

from : AgentID

contributions : \mathbb{P} Contribution

ConstructConfirmationsAuxiliary

$processACComponent : ACAComponent \rightarrow \mathbb{P} AgentID \rightarrow \mathbb{P} AgentModel$
 $\rightarrow ACAComponent$

$processStep : APlanStep \rightarrow \mathbb{P} AgentID \rightarrow \mathbb{P} AgentModel \rightarrow APlanStep$

$orderedAgents : \mathbb{P} AgentID \rightarrow \mathbb{P} AgentModel \rightarrow seq AgentID$

$filterAccepted : \mathbb{P} AgentID \rightarrow \mathbb{P} AgentID \rightarrow \mathbb{P} AgentID$

$\forall requested, accepted : \mathbb{P} AgentID \bullet filterAccepted requested accepted =$
 $\{id : AgentID \mid id \in requested \wedge id \in accepted\}$

$\forall as : \mathbb{P} AgentID; ms : \mathbb{P} AgentModel; as' : seq AgentID \bullet$

$orderedAgents as ms = as' \Leftrightarrow ran as' = as \wedge$

$(\forall n : \mathbb{Z} \mid n \geq 2 \wedge n \leq \#as \bullet$

$trustOfAgent (as'n) ms > trustOfAgent (as'(n-1)) ms)$

$\forall acac : ACAComponent; accepted : \mathbb{P} AgentID; ms : \mathbb{P} AgentModel \bullet$

$(\exists ac, ac' : AnnotatedContribution \mid AContrib(ac) = acac \bullet$

$processACComponent acac accepted ms = AContrib(ac')$

$\Leftrightarrow ac.symbol = ac'.symbol \wedge ac.terms = ac'.terms$

$\wedge ac'.agents =$

$\{orderedAgents (filterAccepted ac.agents accepted) ms 1\}) \vee$

$(\exists acs : \mathbb{P} AnnotatedContribution \mid AJA(ac) = acac \bullet$

$processACComponent acac accepted ms =$

$AJA(\{ac, ac' : AnnotatedContribution \mid ac \in acs \wedge$

$ac.symbol = ac'.symbol \wedge ac.terms = ac'.terms \wedge$

$ac'.agents = \{orderedAgents$

$(filterAccepted ac.agents accepted) ms 1\} \bullet ac')\})$

$\forall aps : APlanStep; accepted : \mathbb{P} AgentID; ms : \mathbb{P} AgentModel \bullet$

$(\exists ac, ac' : AnnotatedContribution \mid AIndividual(ac) = aps \bullet$

$processStep aps accepted ms = AIndividual(ac')$

$\Leftrightarrow ac.symbol = ac'.symbol \wedge ac.terms = ac'.terms$

$\wedge ac'.agents =$

$\{orderedAgents (filterAccepted ac.agents accepted) ms 1\}) \vee$

$(\exists acs : \mathbb{P} AnnotatedContribution \mid AJoint(ac) = aps \bullet$

$processStep aps accepted ms = AJoint($

$\{ac, ac' : AnnotatedContribution \mid ac \in acs \wedge$

$ac.symbol = ac'.symbol \wedge ac.terms = ac'.terms \wedge$

$ac'.agents = \{orderedAgents ($

$filterAccepted ac.agents accepted) ms 1\} \bullet ac')\}) \vee$

$(\exists acacs : \mathbb{P} ACAComponent \mid AConcurrent(acacs) = aps \bullet$

$processStep aps accepted ms = AConcurrent($

$\{acac : ACAComponent \mid acac \in acacs \bullet$

$processACComponent acac accepted ms\})\})$

FormFinalPlan

\exists ConstructConfirmationsAuxiliary

\exists UseActionRequest

responses : \mathbb{P} Response

accepts : \mathbb{P} AgentID

formFinalPlan : AnnotatedPlan \rightarrow \mathbb{P} AgentID \rightarrow \mathbb{P} AgentModel
 \rightarrow AnnotatedPlan

accepts = $\{r : \text{Response} \mid r \in \text{responses} \wedge \text{Third}(r) = \text{true} \bullet \text{First}(r)\}$

$\forall ap, ap' : \text{AnnotatedPlan}; ms : \mathbb{P} \text{AgentModel} \bullet$

formFinalPlan ap accepts ms = ap'

$\Leftrightarrow (\forall n : \mathbb{Z} \mid n \leq \#ap.\text{body} \bullet ap'.\text{body } n =$
processStep (ap.body n) accepts ms)

ConstructConfirmations

\exists ConstructConfirmationsAuxiliary

\exists UseActionRequest

responses : \mathbb{P} Response

accepts : \mathbb{P} AgentID

constructConfirmation : AgentID \rightarrow AgentID \rightarrow AnnotatedPlan
 \rightarrow Confirmation

constructConfirmations : AgentID \rightarrow AnnotatedPlan \rightarrow \mathbb{P} Confirmation

accepts = $\{r : \text{Response} \mid r \in \text{responses} \wedge \text{Third}(r) = \text{true} \bullet \text{First}(r)\}$

$\forall \text{from}, \text{to} : \text{AgentID}; ap : \text{AnnotatedPlan}; c : \text{Confirmation} \bullet$

constructConfirmation from to ap = c $\Leftrightarrow c.\text{from} = \text{from} \wedge$

c.to = to $\wedge c.\text{contributions} =$

agentContributions to (extractAgentsPlan ap)

$\forall \text{from} : \text{AgentID}; ap : \text{AnnotatedPlan} \bullet \text{constructConfirmations from ap} =$

$\{id : \text{AgentID} \mid id \in \text{allAgents } ap \bullet \text{constructConfirmation from } id \text{ } ap\}$

A.32 Strategy Choice

When an agent adopts a plan containing an action for which cooperation is sought, it can solicit assistance and initiate the formation of the required cooperative intention as soon as the plan is selected using an *immediate commitment strategy*, or later at execution time, using a *delayed commitment strategy*. The choice between these strategies is based on the degree of dynamism in the environment, and the overall trust of others, as formalised below.

UseDCS

$$\text{useDCS} : \mathbb{P} \text{AgentID} \rightarrow \mathbb{P} \text{AgentModel} \rightarrow \text{Plan} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R} \\ \rightarrow \mathbb{R} \rightarrow \text{bool}$$
$$\forall \text{capable} : \mathbb{P} \text{AgentID}; \text{ms} : \mathbb{P} \text{AgentModel}; \\ \text{p} : \text{Plan}; \text{dynamism}, \text{dT}, \text{rT}, \text{w}_s, \text{w}_c : \mathbb{R} \bullet \\ \text{useDCS capable ms p dynamism dT rT w}_s \text{w}_c = \text{true} \\ \Leftrightarrow (\text{quality p ms w}_s \text{w}_c) > \text{rT} \vee \text{dynamism} > \text{dT} \wedge \\ \text{useDCS capable ms p dynamism dT rT w}_s \text{w}_c = \text{false} \\ \Leftrightarrow (\text{quality p ms w}_s \text{w}_c) \leq \text{rT} \vee \text{dynamism} \leq \text{dT}$$

A.33 Cooperative Plan Elaboration

Once a cooperative intention has been formed, the agents involved can execute it, by performing the contributions to which they are annotated. When a subgoal is reached it must be elaborated, and in general the subgoal is elaborated by the agent that initiated cooperation. The situations in which the initiator might prefer plan selection to be performed by an agent other than itself are formalised below.

DelegateElaboration

$$\exists \text{Agent}$$
$$\text{delegateElaboration} : \text{Goal} \rightarrow \text{bool}$$
$$\forall \text{subgoal} : \text{Goal} \bullet \text{delegateElaboration subgoal} = \text{true} \Leftrightarrow \\ (\text{planSetForGoal subgoal beliefs planLibrary}) = \emptyset \vee \\ (\forall \text{p} : \text{Plan}; \text{m} : \text{Motivation} \mid \text{m} \in \text{motivations} \wedge \\ \text{p} \in (\text{planSetForGoal subgoal beliefs planLibrary}) \bullet \\ \text{mvPlan m p} \leq 0) \\ \forall \text{subgoal} : \text{Goal} \bullet \text{delegateElaboration subgoal} = \text{false} \Leftrightarrow \\ (\text{planSetForGoal subgoal beliefs planLibrary}) \neq \emptyset \wedge \\ (\exists \text{p} : \text{Plan}; \text{m} : \text{Motivation} \mid \text{m} \in \text{motivations} \wedge \\ \text{p} \in (\text{planSetForGoal subgoal beliefs planLibrary}) \bullet \\ \text{mvPlan m p} \geq 0)$$

A.34 Updating Trust of Others

At the end of a cooperative interaction, each agent involved updates its trust of the others. If the cooperative interaction was successful, and the goal achieved, then the trust an agent associates with the others involved is likely to increase. Conversely, if the goal was not achieved then the cooperative intention was unsuccessful, and trust is likely to decrease.

$$\textit{increaseTrust} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}$$

$$\textit{decreaseTrust} : \mathbb{R} \rightarrow \mathbb{R} \rightarrow \mathbb{R}$$

$$\forall \textit{trust}, \textit{trust}', \textit{trustIncrease} : \mathbb{R} \bullet \textit{increaseTrust} \textit{trust} \textit{trustIncrease} = \textit{trust}' \Rightarrow \textit{trust}' > \textit{trust}$$

$$\forall \textit{trust}, \textit{trust}', \textit{trustDecrease} : \mathbb{R} \bullet \textit{decreaseTrust} \textit{trust} \textit{trustDecrease} = \textit{trust}' \Rightarrow \textit{trust}' < \textit{trust}$$

Appendix B

Implementation of the SENARA Testbed

B.1 Introduction

The appropriateness of any design or model is dependent on the environment in which it is situated. Our architecture and model of cooperation is intended for complex domains, in which agents are situated in a dynamic, unpredictable environment. SENARA agents themselves are complex entities, and the interaction of a group of agents situated in a complex environment is virtually unpredictable, and an external high-level analysis is too complex to be practical. Therefore, to assist the understanding of the work described in this thesis, a SENARA testbed has been developed to provide a platform for empirical investigation. The objective in constructing the testbed is to demonstrate the concepts presented in this work, and allow simple experimentation, rather than to develop a sophisticated finished product.

B.2 Overview of the Testbed

The domain in which our testbed is grounded is the warehouse domain introduced in Chapter 5¹. Agents are situated in a warehouse comprising three areas: a delivery area, a standard storage area, and a long term storage area. Boxes arrive in the delivery area and must be moved to one of the storage areas (or rooms), which for simplicity are arranged linearly as shown earlier in Figure 5.2.

The testbed itself is constructed in Java, and comprises a set of distributed autonomous agents of the form specified in Chapters 3 and 4, and able to perform the tasks required for cooperation in our model. Since we are concerned with software agents, the environment in which they are situated also exists in software only. Agents are able to perceive and act upon the environment using Java's Remote Method Invocation mechanism, and the Internet acts as a communication channel. Each agent is constructed as a stand-alone application having a distinct thread of execution. Indeed, in our investigations each agent is invoked on different machine, ensuring they are asynchronous, and providing an easy mechanism for adjusting the relative speed of an agent's reasoning — we can make an agent relatively faster or slower by simply artificially decreasing or increasing the load of the processor on which it is running. The results that we describe in this appendix have been obtained using a selection of platforms² with agents and the environment running on different, geographically distributed, machines. This demonstrates both the platform independent nature of the implementation, and moreover, the ability of the cooperation framework to cope with the lack of synchronisation, and the communication delays resulting from such a situation.

Interaction with the testbed is via command a prompt interface, presented by each entity in the environment, and the environment representation itself. Through this interface users

¹Although we concentrate here on them warehouse domain, the testbed has also been tested on the rubbish collecting environment used in Rao's discussion of AgentSpeak(L) [81], where agents must collect rubbish that appears in one of a number of lanes of traffic, whilst avoiding collisions.

²In particular we have tested the implementation using various combinations of Linux, Solaris, and Windows 98/2000 machines.

can invoke a variety of functions that change the state of an agent, such as changing the intensity of its motivations, or its trust of others, or even forcing it to drop a particular intention. The user can also manually invoke the agent's reasoning cycle, and thus have extra control over the relative speed of agent's reasoning. The environment allows the user to add and remove objects, change the power level of an agent, and intercept and remove messages between agents. By intercepting a message (and removing it) the user can simulate a broken communication link between the agents, allowing the model to be tested in situations where communications are unreliable. In addition to being run interactively, experiments in the testbed can also be automated, with the user specifying the number of cycles, and delay between them, on start up. An initial situation can be specified for the environment, along with specifying a set of predefined *paths* that define when certain events occur in the environment. For example, a path may specify that a box appears in the delivery area at a particular point in time, and another appears a certain number of cycles later. The interaction windows for a group of four agents, and the environment can be seen in Figure B.1, which shows the beginning of a potential interaction between `agent1` and `agent4`, where `agent1` has requested assistance from `agent4` in storing a large box.

Output from the testbed is in the form of a trace of the mental state and actions (including any commitments) of the agents in the environment, along with a trace of the changes in environment itself. Since the agents and environment are distributed, each entity produces its own trace, and this requires some analysis to determine the order of events and state changes in the testbed, and to understand the circumstances that led to a particular situation. To aid this analysis a local time stamp is printed on the output of each iteration of an agent's control cycle, and before beginning an experiment we synchronise the local times on the machines involved.

The warehouse scenario is a complex and dynamic domain — communications are not guaranteed (since users can intercept messages) and execution is asynchronous and distributed. Thus, it offers a approximation of the issues faced in a real-world multi-agent domain. However, a number of simplifications are made, since we are concerned with

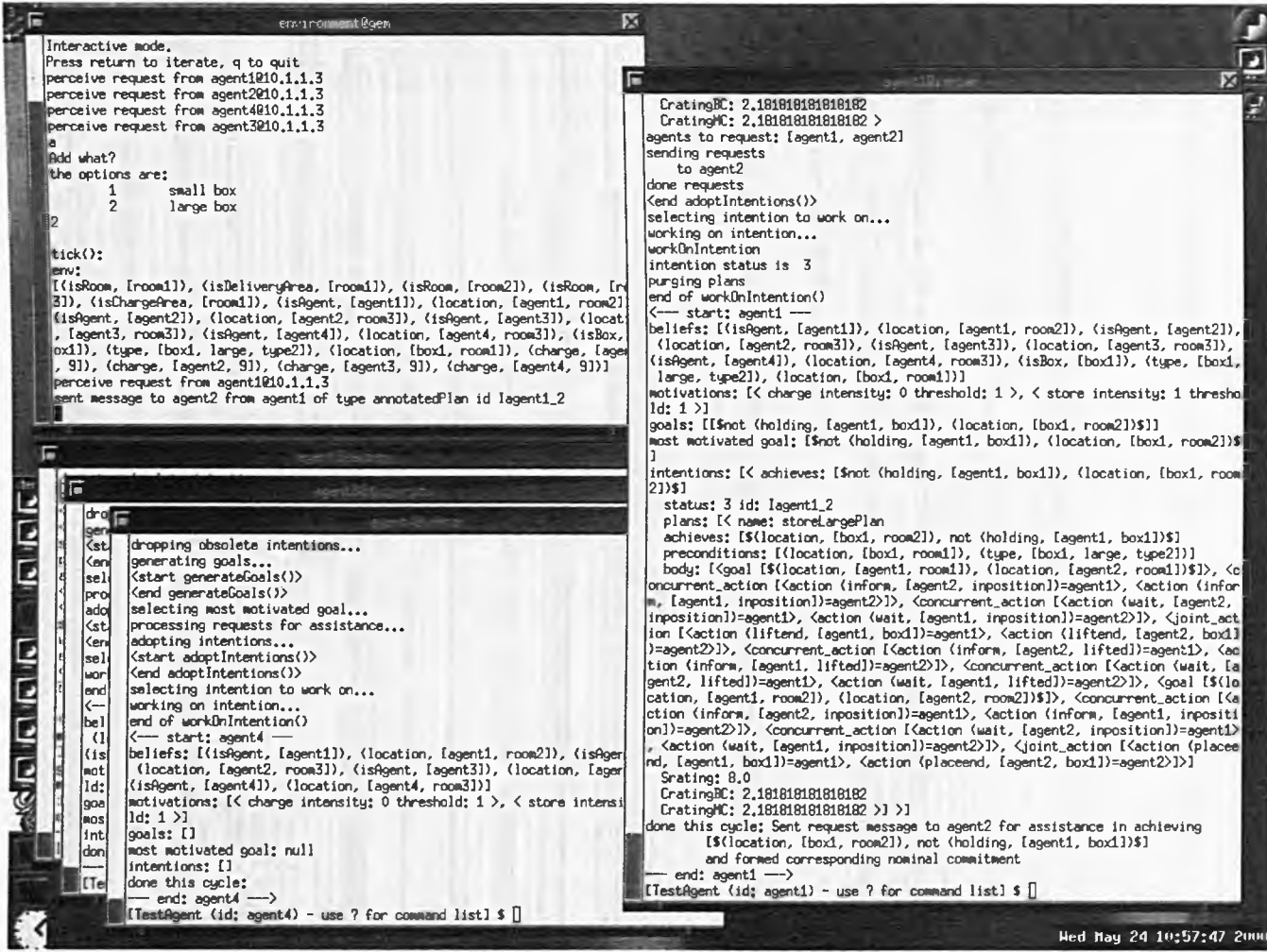


Figure B.1: The interface to the SENARA testbed

```

Interactive mode.
Press return to iterate, q to quit
perceive request from agent1@10.1.1.3
perceive request from agent2@10.1.1.3
perceive request from agent4@10.1.1.3
perceive request from agent3@10.1.1.3
a
Add what?
the options are:
  1      small box
  2      large box
2
tick():
env:
[(isRoom, [room1]), (isDeliveryArea, [room1]), (isRoom, [room2]), (isRoom, [r
3]), (isChargeArea, [room1]), (isAgent, [agent1]), (location, [agent1, room2]),
(isAgent, [agent2]), (location, [agent2, room3]), (isAgent, [agent3]), (locat
, [agent3, room3]), (isAgent, [agent4]), (location, [agent4, room3]), (isBox,
ox1]), (type, [box1, large, type2]), (location, [box1, room1]), (charge, [age
, 9]), (charge, [agent2, 9]), (charge, [agent3, 9]), (charge, [agent4, 9])]
perceive request from agent1@10.1.1.3
sent message to agent2 from agent1 of type annotatedPlan id [agent1_2

```

```

dro
gen
<st
<en
sel
<end
pro
ado
<st
<en
sel
war
end
<
bel
(1
(is
mot
ld:
goe
mos
int
don
[Te
[TestAgent (id: agent4) - use ? for command list] $ ]

```

```

CratingC: 2.181818181818182
CratingMC: 2.181818181818182 >
agents to request: [agent1, agent2]
sending requests
  to agent2
done requests
<end adoptIntentions()>
selecting intention to work on...
working on intention...
workOnIntention
intention status is 3
purging plans
end of workOnIntention()
<← start: agent1 →
beliefs: [(isAgent, [agent1]), (location, [agent1, room2]), (isAgent, [agent2]),
(location, [agent2, room3]), (isAgent, [agent3]), (location, [agent3, room3]),
(isAgent, [agent4]), (location, [agent4, room3]), (isBox, [box1]), (type, [box1,
large, type2]), (location, [box1, room1])]
motivations: [(charge intensity: 0 threshold: 1), < store intensity: 1 thresho
ld: 1 >]
goals: [(!$not (holding, [agent1, box1]), (location, [box1, room2])$)]
most motivated goal: [$not (holding, [agent1, box1]), (location, [box1, room2])$
]
intentions: [(achieves: [$not (holding, [agent1, box1]), (location, [box1, room
2])$)]
status: 3 id: [agent1_2]
plans: [(name: storeLargePlan
achieves: [(location, [box1, room2]), not (holding, [agent1, box1])$]
preconditions: [(location, [box1, room1]), (type, [box1, large, type2])$]
body: [(goal [(location, [agent1, room1]), (location, [agent2, room1])$]), <
concurrent_action [(action (inform, [agent2, inposition])=agent1), <action (infor
m, [agent1, inposition])=agent2]), <concurrent_action [(action (wait, [agent2,
inposition])=agent1), <action (wait, [agent1, inposition])=agent2]), <joint_act
ion [(action (liftend, [agent1, box1])=agent1), <action (liftend, [agent2, box1]
)=agent2]), <concurrent_action [(action (inform, [agent2, lifted])=agent1), <ac
tion (inform, [agent1, lifted])=agent2]), <concurrent_action [(action (wait, [a
gent2, lifted])=agent1), <action (wait, [agent1, lifted])=agent2]), <goal [(lo
cation, [agent1, room2]), (location, [agent2, room2])$]), <concurrent_action [(a
ction (inform, [agent2, inposition])=agent1), <action (inform, [agent1, inpositi
on])=agent2]), <concurrent_action [(action (wait, [agent2, inposition])=agent1)
, <action (wait, [agent1, inposition])=agent2]), <joint_action [(action (placee
nd, [agent1, box1])=agent1), <action (placeend, [agent2, box1])=agent2])$]
Crating: 8.0
CratingRC: 2.181818181818182
CratingMC: 2.181818181818182 >>]
done this cycle: Sent request message to agent2 for assistance in achieving
[(location, [box1, room2]), not (holding, [agent1, box1])$]
and formed corresponding nominal commitment
— end: agent1 →
[TestAgent (id: agent1) - use ? for command list] $ ]

```

demonstrating and increasing our understanding of the model, and not with developing a product. In particular we make the following simplifying assumptions.

- Rooms in the warehouse are laid out linearly, requiring movement along one axis only.
- Agents are able to perceive the whole environment.
- Agents do not enter or leave the system while it is in operation
- Communication is via a simplified agent communication language, containing only performatives for *requesting* assistance, *declining* a request, *accepting* a request, and *informing* an agent of something.
- No errors or losses occur in communication, unless explicitly caused by user interaction with the environment.

Although this environment is simple, agents in it must still cooperate with others that are potentially unreliable to achieve certain goals. They must select the best plan for a given situation based on both the cost and risk associated with plans. Cooperative plans must be annotated with agents to request assistance from, and a cooperative intention must be formed before cooperation can actually occur. Therefore, it represents a balance between being rich enough to demonstrate the model, and simple enough to understand the interactions that occur and the reasons for them.

The remaining sections of this appendix can be divided into two categories. Firstly, in the following section we give the details of the plan library given to agents in the testbed, and then in Section B.4 we discuss how agents can order the execution of their actions. Secondly, in the final section of this appendix we give a brief walk-through of the testbed in operation in a specific situation, demonstrating the mechanisms and processes described in the earlier chapters of this thesis. For ease of understanding we take a simple example of two agents forming a cooperation intention and jointly performing the actions involved in moving a box from the delivery area to a storage area.

```

name: moveRightPlan
  achieves: [$(location, [_agent, _y])$]
  preconditions: [(location, [_agent, _x]), (isRightOf, [_y, _x])]
  body: [<action (move, [_agent, right])=_agent>,
        <goal [$(location, [_agent, _y])$]>]
name: moveLeftPlan
  achieves: [$(location, [_agent, _y])$]
  preconditions: [(location, [_agent, _x]), (isLeftOf, [_y, _x])]
  body: [<action (move, [_agent, left])=_agent>,
        <goal [$(location, [_agent, _y])$]>]
name: stayPutPlan
  achieves: [$(location, [_agent, _x])$]
  preconditions: [(location, [_agent, _x])]
  body: []
name: rechargePlan
  achieves: [$(location, [_agent, _y]), (charge, [_agent, 10])$]
  preconditions: []
  body: [<goal [$(location, [_agent, _y])$]>,
        <action (recharge, [_agent])=_agent>]

```

Table B.1: Plans for moving and recharging in the warehouse domain

B.3 Plan Library

We introduced the Warehouse domain in Section 5.5, however, so as not to complicate our earlier discussions we did not give details of the plans given to agents. Thus, before considering a sample interaction we briefly describe the set of plans given to agents in the testbed.

Agents are able to perform certain actions in the warehouse, in particular they are able to move around, pick up and put down boxes, and check that boxes are stored correctly. There are three types of lifting action: `pickup` which operates on small boxes, `liftend` which lifts one end of a large box, and `pickupBIG` which operates on large boxes. All agents are capable of performing these actions, with the exception of the `pickupBIG` action, which can only be performed by a specific agent, `agent3`.

For an agent to be able to achieve a goal, it requires a plan specifying the actions that

are needed for its achievement. In SENARA, an agent is given a library of plans from which it selects the most appropriate for a particular goal, as described in Chapter 3, rather than planning from first principles. Agents in the warehouse scenario must be able to move around their environment, store boxes that are delivered, check boxes are correctly stored, move boxes to the waste disposal area, and recharge their power levels when required, and we provide agents with a plan library to achieve this.

We begin by defining three plans for moving around their environment, i.e. achieving the goal of being in a particular location; agents are given a plan for moving right, moving left, and staying in their current location. The latter is required so that an agent has a plan to achieve the goal of being in specific room when it is already there. These plans, along with a plan defining how an agent can recharge its power, are shown in Table B.1, in which x denotes a variable and g denotes a goal.

Along with the ability to move around in the warehouse, agents must be able to store boxes that are delivered. Now, there are two sizes of box, small and large, and small boxes can be stored by individuals, while large boxes must be stored by a specific agent, or through cooperation — agents need appropriate plans to represent this. The plans associated with storing boxes are shown in Table B.2, which contains plans for an individual storing a small box, two agents storing a large box together, and an agent storing a large box alone. Table B.2 also contains a plan for checking that a box is correctly stored — an agent must move to the location of the box, and check it has not been stored beyond its expiry time. We do not need to define additional plans for moving boxes to the waste disposal area, since the storage plans can be used (with the target room set appropriately). These plans, taken in conjunction with those described earlier, comprise the agent's plan library.

```

name: storeSmallPlan
  achieves: [$(location, [_box, _room]), not (holding, [_agent, _box])$]
  preconditions: [(location, [_box, _loc]), (type, [_box, _size, _type])]
  body: [<goal [$(location, [_agent, _loc])$]>,
        <action (pickup, [_agent, _box])=_agent>,
        <goal [$(location, [_agent, _room])$]>,
        <action (putdown, [_agent, _box])=_agent>]
name: storeLargePlan
  achieves: [$(location, [_box, _room]), not (holding, [_agent, _box])$]
  preconditions: [(location, [_box, _loc]), (type, [_box, _size, type])]
  body: [
    <goal
      [$(location, [_agent1, _loc]),
       (location, [_agent2, _loc])$]>,
    <joint_action
      [<action (liftend, [_agent1, _box])=_agent1>,
       <action (liftend, [_agent2, _box])=_agent2>]>,
    <goal
      [$(location, [_agent1, _room]),
       (location, [_agent2, _room])$]>,
    <joint_action
      [<action (placeend, [_agent1, _box])=_agent1>,
       <action (placeend, [_agent2, _box])=_agent2>]>]
name: storeLargePlanCheap
  achieves: [$(location, [_box, _room]), not (holding, [_agent, _box])$]
  preconditions: [(location, [_box, _loc]), (type, [_box, _size, _type])]
  body: [<goal [$(location, [_agent, _loc])$]>,
        <action (pickupBIG, [_agent, _box])=_agent>,
        <goal [$(location, [_agent, _room])$]>,
        <action (putdownBIG, [_agent, _box])=_agent>]
name: checkPlan
  achieves: [$(checked, [_agent, _box])$]
  preconditions: [(location, [_box, _loc])]
  body: [<goal [$(location, [_agent, _loc])$]>,
        <action (check, [_agent, _box])=_agent>]

```

Table B.2: Plans for storing and checking boxes in the warehouse domains

B.4 Synchronising and Ordering Action Execution

In executing a cooperative intention agents must ensure that their actions are performed in the correct order, and that the performance of contributions comprising any joint actions are synchronised. Recall from Chapter 7 that we adopt Kinny *et al.*'s solution to ordering and synchronising actions. For completeness we give more details of this approach in this section.

B.4.1 Synchronisation

Solutions to the problem of synchronisation of the contributions comprising a joint action tend to be domain specific. For example, in the Warehouse Domain all the possible joint actions are related to lifting boxes. This requires agents to be in the same area of the warehouse, meaning that synchronisation of contributions can be achieved through simply observing the behaviour of the other agents of the joint actions. However, this method is not applicable where agents are not in the same location as each other. Other types of joint action exist where agents do not need to be in the same place. For example, performing an update operation on a distributed database can be considered to be a joint action requiring several local updates (the contributions) simultaneously. In this case agents must use some other method for achieving synchronisation, through communication. However, the details of how this is best achieved will depend on the nature of their communication. If agents are relatively close in geographic terms, then they may be able to synchronise using the Internet. However, if they are distributed across different continents then the communication delay is likely to be too great, and unpredictable³, to achieve synchronisation over the Internet, and another method must be found.

No general solution that works for all domains exists for synchronisation of joint actions. Solutions can be achieved through communication *if* agents have access to a global

³The delay will be unpredictable since it will depend how messages are routed

clock, know when messages are sent, know the communication delay, and communication is reliable [46]. For example, agents might agree to perform their contributions at a specific time. However, the details of such approaches are beyond the scope of this thesis.

B.4.2 Action Ordering

Similarly to the problem of synchronising joint actions, the most efficient mechanism for constraining the order of actions is dependent of the domain concerned. For example, if it is possible for agents to perceive the actions of others then, before performing its part of a plan, an agent can simply observe whether the preceding action is complete. However, in general, it is not possible for agents to perceive the actions of others, and indeed, even if it were possible, there is the additional problem of ensuring the action observed is part of the plan concerned, and not independent of it⁴.

Unlike the problem of synchronisation, however, it is possible to offer a general solution to the problem of ordering actions based on communication. Although this solution is certainly not the most efficient in all domains, it is straightforward, and provides a base from which to develop a more efficient solution for a given application domain. The problem of action ordering arises from the need for an agent to know that the execution of the previous contribution is complete before beginning the execution of the following action. Kinny *et al.* offer a simple solution in their model of Planned Team Activity [55], which we adopt in this thesis.

The solution is to require that the agent executing a given action informs the agent of the following action when execution is successfully performed. Correspondingly, the agent of the following action must not perform its contribution until it is informed that the previous action has been completed. In the case of joint and concurrent actions we extend this solution such that the *set* of agents performing joint or concurrent actions must

⁴This is a problem since although the action observed might be the same, if it is not part of the same plan the preceding actions in the plan will not have been performed.

inform the agent(s) of the following action when their contributions are complete. Similarly, each agent involved in the execution of the following action must wait until it has been informed of completion by *each* of the agents performing a contribution in the previous action. Action ordering can be achieved, therefore, by inserting appropriate communication and waiting actions into the plan prior to its execution. In general, only the initiating agent has knowledge of the complete plan, and therefore the actions required for ordering must be added by the initiator.

Recall from Chapter 7 that a cooperative intention can be established through either an immediate or a delayed commitment strategy⁵. If an ICS was used, then commitments have been obtained for all of the appropriate actions in the current elaboration of the plan. Conversely, if a DCS is used, then each action is treated in turn, with a cooperative intention being established on an action by action basis. This has a bearing on the achievement of action ordering, since the steps in the plan are treated in different ways. Where the ICS is used then action ordering can be achieved as outlined above, with the agents of an action informing the agents of the following action when its execution is complete. If a DCS is used then it is not known at the time of executing an action which agents will perform the following one, unless it is an individual action to be performed by the initiating agent. Therefore, if a DCS is used, all that is required is that the agents performing a cooperative action inform the initiating agent when their contributions are complete. The initiating agent can then bring about execution of the following action, by forming a cooperative intention or by performing it itself.

We now give the algorithm the initiator uses to insert the appropriate actions into its plan. We concentrate on the case where an ICS has been used to establish a cooperative intention. The algorithm is to step through the actions in the plan, and wherever an action is followed by another that is to be performed by different agents, two actions are inserted. The first is an action for the agents of the current action to inform the agents of the following action as soon as it is complete. The second is an action for the agents of the following

⁵ICS and DCS respectively

Inputs:

plan — the plan for which action ordering is needed

Outputs:

plan' — the plan with appropriate ordering action inserted

Algorithm:

```
plan' = plan
for action in plan do
    if not final(act, plan) and not agentsOf(next(action)) =
        agentsOf(action) then
        informAct = inform(agentsOf(next(action)), done(action))
        annotate(informAct, agentsOf(action))
        insertAfter(action, informAct)
        waitAct = informed(agentsOf(action), done(action))
        annotate(waitAct, agentsOf(next(action)))
        insertBefore(next(action), waitAct)
return plan'
```

Figure B.2: Algorithm for inserting ordering actions based on Kinny *et al.*'s work

action to wait until they have been informed of the previous action's completion, as shown in Figure B.2.

B.5 Example Interaction

In this section we describe a simple cooperative interaction between two agents in the warehouse domain. A complete trace of the output from the testbed for even a simple example is rather large (in the order of 3000 lines for 40 cycles of an agent's control cycle), and so for brevity we only include extracts of the trace here.

B.5.1 Plan selection

In order to achieve a particular goal, an agent must select an appropriate plan from its plan library, and commit to its achievement through the formation of an intention. Where there

is only one plan in the library that achieves the goal and has its preconditions satisfied, then plan selection is a trivial, since that plan is selected. However, if there is more than one plan that achieves the goal (and has its preconditions satisfied) then the agent must choose between them. In Chapter 6 we proposed an approach to plan selection in this situation that takes into account both the cost of a plan, using standard planning heuristics, and the risk associated with it with respect to cooperation. The plan selection process consists of two key components: an assessment of this cost and risk, and the choice between plans for a given goal.

B.5.2 Pre-execution Assessment

All plans in an agent's library are assessed to arrive at a *standard rating* based on standard planning heuristics such as length and cost, and a *cooperative rating* based on the estimated risk arising from cooperation. To arrive at these ratings an agent must consider the actions contained in a plan, along with the set of possible elaborations of any subgoals it contains. However, because constructing the entire plan tree at plan selection time is too expensive, agents perform an off-line *pre-execution* assessment of the plan library. As described in Section 6.5 agents start by assessing plans that require no further elaboration, since these can be directly evaluated without considering subplans. These ratings are then fed into other plans in the library as values for subgoals within them, and so on in subsequent levels of elaboration.

We illustrate this process by considering the assessment of the plan library introduced above, from the perspective of a particular agent, `agent1`. Assessment begins with the plan for staying in the current location, since this is the only plan that does not require further elaboration. However, since it does not contain any actions either, its standard and cooperative rating are both zero.

```
start assessPlanLib
  assessing stayPutPlan
    plan's cooperative rating assessed as
```

The plan for staying in the current location may be used in the elaboration of the plans for moving right and left, and so its rating is used in determining their ratings. To obtain the rating for `moveRightPlan`, each of the steps in its body is considered in turn. Firstly, the rating for the step corresponding to the action of moving right is obtained by considering its standard cost and the trust of the agents that might perform it, since the plan, and therefore the action of moving right, may be performed by *another* agent. Secondly, the subgoal of being in a particular location is assessed by considering the ratings of its possible elaborations. Two possible plans (apart from the plan for moving right itself) might be used in its elaboration: `stayPutPlan` and `moveLeftPlan`. The rating for the former of these can be incorporated, but the latter cannot since `moveLeftPlan` and `moveRightPlan` are mutually recursive meaning each might be a subplan of the other. Therefore, their ratings must be scaled for recursion, as described in Chapter 6. The plan for moving left is assessed in a similar manner, giving the following results.

```

assessing moveRightPlan
  current step: <action (move, [_agent, right])=_agent>
  step is action, assessing... value is 1.03
  current step: <goal [$(location, [_agent, _y])$]>
  step is goal, finding BC and MC rating
    poss elaborations: [moveLeftPlan, stayPutPlan]
    considering subplan moveLeftPlan:
      plans are mutually recursive
    considering subplan stayPutPlan
      BC value is 0.0 MC value is 0.0
    scaling for recursion... factor is 3.0
  plan's cooperative rating assessed as
  3.08 (BC) 3.08 (MC)
  plan's quality assessed as
  6.08 (BC) 6.08 (MC)
assessing moveLeftPlan
:
  plan's cooperative rating assessed as
  6.08 (BC) 6.08 (MC)

```

Likewise, these ratings are used in assessing the subsequent level of plans in the library,

which in this case corresponds to all the remaining plans. Firstly, consider the plan for storing small boxes. The body of this plan contains the steps of moving to the location of the box (for which one of the above three plans will be used), picking up the box, moving to the storage location (again using one of the above plans), and finally putting down the box. The assessment of this plan is therefore achieved by considering the ratings of the actions of picking up and putting down a box, along with the ratings of the three plans already assessed. As described in Section 6.5, when considering a set of plans for the elaboration of a subgoal, the best-case and mean-case ratings are determined, and used to arrive at a best-case and mean-case rating for the plan itself. The result of assessing the plan for storing a small box is as follows.

```
assessing storeSmallPlan
  current step: <goal [$(location, [_agent, _loc])$]>
  step is goal, finding BC and MC rating
    poss elaborations: [stayPutPlan, moveRightPlan,
      moveLeftPlan]
    considering subplan stayPutPlan
      BC value is 0.0 MC value is 0.0
    considering subplan moveRightPlan
      BC value is 6.08 MC value is 6.08
    considering subplan moveLeftPlan
      BC value is 6.08 MC value is 6.08
  current step: <action (pickup, [_agent, _box])=_agent>
  step is action, assessing... value is 1.03
  current step: <goal [$(location, [_agent, _room])$]>
  step is goal, finding BC and MC rating
    poss elaborations: [stayPutPlan, moveRightPlan,
      moveLeftPlan]
    considering subplan stayPutPlan
      BC value is 0.0 MC value is 0.0
    considering subplan moveRightPlan
      BC value is 6.08 MC value is 6.08
    considering subplan moveLeftPlan
      BC value is 6.08 MC value is 6.08
  current step: <action (putdown, [_agent, _box])=_agent>
  step is action, assessing... value is 1.03
  plan's cooperative rating assessed as
    2.05 (BC) 10.16 (MC)
  plan's quality assessed as
    10.05 (BC) 18.16 (MC)
```

In this manner the remaining plans are assessed, resulting in the following best-case

and mean-case ratings.

```
assessing storeLargePlan
  plan's quality assessed as
    23.82 (BC) 23.82 (MC)
name: storeLargePlanCheap
  plan's quality assessed as
    13.85 (BC) 21.96 (MC)
assessing rechargePlan
  plan's quality assessed as
    7.0 (BC) 11.05 (MC)
```

Now, since the trust placed in agents changes over time, these ratings may become out of step with the current situation, and so they must be reassessed periodically. For example, if `agent1`'s trust of `agent3` changes from 0.52 to 0.1, i.e. from a high to a low level of trust, then the ratings of risk associated with plans that might involve `agent3` will increase when reassessed, to reflect the increased risk. For example, the actions in the plan for an individual agent storing a large box must be performed by `agent3` because it is the only agent capable of moving such a box. Therefore, if the plan library is reassessed the ratings of this plan will increase, giving the following new rating.

```
name: storeLargePlanCheap
  plan's cooperative rating assessed as
    30.0 (BC) 38.83 (MC)
```

B.5.3 Plan Selection

The ratings determined by an agent's pre-execution assessment of its plan library are used to select the best plan to achieve a goal. In the case where there is only one applicable plan, and that plan can be performed individually, then plan selection is a trivial — the applicable plan is selected. For example, if a small box is delivered to the warehouse, and `agent1` perceives this, it will form the goal of storing the box, as a result of its *tidiness* motivation. In order to form an intention it must select a plan for this goal, and the only applicable plan

is `storeSmallPlan`. Thus, the agent selects this plan, and forms an intention towards its execution.

However, where there are a number of applicable plans, and these plans require cooperation to perform, then plan selection is more complex. For example, consider the situation where `agent1` perceives that a large box has arrived in the delivery area, and adopts the goal of moving it to the storage area. There are two applicable plans in this situation, `storeLargePlan` and `storeLargePlanCheap`, the former uses joint actions of two agents lifting the box and moving it, while the latter must be executed by an individual agent with the ability to lift a large box. Any of the other agents can assist for the execution of the former plan (since all agents have the required capabilities), but only `agent3` can assist for the latter, since it is the only agent able to perform the action of lifting a large box individually. The best plan should be selected based on the best-case and mean-case advantage as discussed in Section 6.5. Using the ratings calculated above, the best-case advantage of choosing `storeLargePlan` over `storeLargePlanCheap` is 9.97, and the mean-case advantage is 1.86. Thus, the best-case advantage is greater and so the plan with the lowest best-case rating should be selected, namely `storeLargePlanCheap`⁶. The agent can then begin the procedures required to adopt this plan as an intention.

Alternatively, if the agent had a different level of trust in the others, then a different plan might have been chosen. For example, if `agent3` is little trusted and associated with a trust value of 0.1, instead of being trusted, then the rating for the `storeLargePlanCheap` changes as given above. The ratings for the other plans also change; in particular, the best-case and mean-case ratings for `storeLargePlan` both become 33.53. Here, the risk associated with `storeLargePlanCheap` is significantly increased, and the best-case and mean-case advantages become 3.53 and 5.3 respectively and therefore the mean-case rating should be used to select the best plan. The plan `storeLargePlan` has the lowest mean-case rating, and so is selected.

⁶In this case, `storeLargePlanCheap` would also have been chosen using the mean-case rating, but this is not always the case.

B.5.4 Intention Adoption

Returning to our example interaction, the chosen plan is then used to adopt an appropriate intention, as described in Chapter 7. For individual plans, intention adoption is a trivial matter of committing to a the chosen plan. Where the plan requires cooperation, however, a cooperative intention must be formed (unless a delayed commitment strategy is used). If we take the above example of the agent selecting `storeLargePlan` for its goal of storing a large box, then it must annotate that plan, request assistance, and form a cooperative intention if sufficient agents accept.

To annotate its plan the agent considers each action step in the plan in turn (excluding the synchronisation actions introduced earlier in this section), and annotates it with the appropriate agents, based on its trust of them. Of course, annotating a contribution to itself avoids the risk associated with cooperation, and so is better from a risk perspective, but it does require the agent to act and so will have an associated cost. Thus, an agent must decide whether to annotate itself to a contribution by balancing the risk and cost. After deciding which contributions in the plan to perform itself the agent should go through the remaining steps, annotating them with the most trusted agent (or set of agents in the case of joint and concurrent actions) that have the required capabilities. In our example, suppose that `agent1` annotates itself to one of the contributions in the joint actions of picking up, moving, and putting down the box. The remaining contributions in the joint actions are annotated to the most trusted agent, in this case `agent4`. Finally, the communication actions are annotated by substituting the bindings of agent identifiers to variables that were selected for the other actions.

```
chosenPlan is cooperative
chosenPlan: name: storeLargePlan
current step:
  <goal [$(location, [_agent1, room1]),
  (location, [_agent2, room1])$]>
current step:
  <concurrent_action [<action (inform, [_agent2,
  inposition])=_agent1>,
  <action (inform, [_agent1, inposition])=_agent2>]>
```



```

current step:
  <concurrent_action [<action (wait, [_agent2,
    inposition])=_agent1>,
  <action (wait, [_agent1, inposition])=_agent2>]>
current step:
  <joint_action [<action (liftend, [_agent1, box1])=_agent1>,
  <action (liftend, [_agent2, box1])=_agent2>]>
  <start minAnnotateJointAction()>
    act (liftend, [_agent1, box1])
    assigning self to (liftend, [_agent1, box1])
    (liftend, [agent1, box1]) agent1
    act (liftend, [_agent2, box1])
    assigned []
    capable [agent2, agent3, agent4]
    chosen agent4
    (liftend, [agent4, box1]) agent4
  <end minAnnotateJointAction()>
annotated step:
  <joint_action [<action (liftend, [agent1, box1])=agent1>,
  <action (liftend, [agent4, box1])=agent4>]>
current step:
  <concurrent_action [<action (inform, [_agent2,
    lifted])=_agent1>,
  <action (inform, [_agent1, lifted])=_agent2>]>
current step:
  <concurrent_action [<action (wait, [_agent2,
    lifted])=_agent1>,
  <action (wait, [_agent1, lifted])=_agent2>]>
current step:
  <goal [$(location, [_agent1, room2]),
  (location, [_agent2, room2])$]>
current step:
  <concurrent_action [<action (inform, [_agent2,
    inposition])=_agent1>,
  <action (inform, [_agent1, inposition])=_agent2>]>
current step:
  <concurrent_action [<action (wait, [_agent2,
    inposition])=_agent1>,
  <action (wait, [_agent1, inposition])=_agent2>]>
current step:
  <joint_action [<action (placeend, [_agent1, box1])=_agent1>,
  <action (placeend, [_agent2, box1])=_agent2>]>
  <start minAnnotateJointAction()>
    act (placeend, [_agent1, box1])
    assigning self to (placeend, [_agent1, box1])
    (placeend, [agent1, box1]) agent1
    act (placeend, [_agent2, box1])
    assigned []
    capable [agent2, agent3, agent4]
    chosen agent4
    (placeend, [agent4, box1]) agent4
  <end minAnnotateJointAction()>

```

annotated step:

```
<joint_action [<action (placeend, [agent1, box1])=agent1>,  
<action (placeend, [agent4, box1])=agent4>]>
```

Once the plan is annotated, the agent sends a request for assistance to `agent4` and forms a nominal commitment. For simplicity, and since we are taking a closely coupled view, requests for assistance in the warehouse scenario are based upon communication of the complete annotated plan. Thus, `agent1`'s request message to `agent4` includes the complete plan.

At this stage of execution `agent1` has sent a request, and formed a nominal commitment, and `agent4` must process this request. Now, `agent4` will also have perceived the environment and the box in the delivery area, and so the intensity of its `tidiness` motivation will also be high. The plan contained in the request mitigates this motivation, and is considered to be of motivational value (the motivational effect outweighs the cost of performing the contributions). If `agent1` is trusted by `agent4`, which according to the trust matrix given earlier it is (with a trust value of 0.96), then it accepts the request, and forms a corresponding nominal commitment.

On receiving the acceptance message the initiating agent (`agent1`) can form a full commitment, since `agent4` is the only other agent involved, and send a confirmation message. When `agent4` receives this confirmation, it too can adopt a full commitment, and execution can begin.

B.5.5 Intention Execution

After the establishment of a cooperative intention, the agents concerned can begin their execution. The main issue in execution of a cooperative intention is the correct ordering of actions in the plan, which can be achieved by inserting communication actions. Inform actions are inserted after the joint (or concurrent) actions in the plan and after the subgoals, since their achievement may involve a number of agents, and corresponding wait actions

are inserted after these inform actions. Thus, the body of the `storeLargePlan` becomes as follows.

```
name: storeLargePlan
  body: [
    <goal
      [$(location, [_agent1, _loc]),
        (location, [_agent2, _loc])$]>,
    <concurrent_action
      [<action (inform, [_agent2, inposition])=_agent1>,
        <action (inform, [_agent1, inposition])=_agent2>]>,
    <concurrent_action
      [<action (wait, [_agent2, inposition])=_agent1>,
        <action (wait, [_agent1, inposition])=_agent2>]>,
    <joint_action
      [<action (liftend, [_agent1, _box])=_agent1>,
        <action (liftend, [_agent2, _box])=_agent2>]>,
    <concurrent_action
      [<action (inform, [_agent2, lifted])=_agent1>,
        <action (inform, [_agent1, lifted])=_agent2>]>,
    <concurrent_action
      [<action (wait, [_agent2, lifted])=_agent1>,
        <action (wait, [_agent1, lifted])=_agent2>]>,
    <goal
      [$(location, [_agent1, _room]),
        (location, [_agent2, _room])$]>,
    <concurrent_action
      [<action (inform, [_agent2, inposition])=_agent1>,
        <action (inform, [_agent1, inposition])=_agent2>]>,
    <concurrent_action
      [<action (wait, [_agent2, inposition])=_agent1>,
        <action (wait, [_agent1, inposition])=_agent2>]>,
    <joint_action
      [<action (placeend, [_agent1, _box])=_agent1>,
        <action (placeend, [_agent2, _box])=_agent2>]>]
```

This plan transformation is performed *before* the initiating agent communicates the plan to the potential participants in order to reduce the communication overhead of communicating the plan twice (once without ordering actions to gain commitment, and once with them prior to execution). In our prototypical implementation we are not concerned with investigating synchronisation mechanisms, and so we take a simplistic, but functional, view. Synchronisation is required for each joint action in a plan, and our approach is that agents should simply wait a specified time after receiving the inform messages from the agents of

the previous action before performing their contribution. This mechanism is sufficient if the communication delays are negligible, and in our test environment this is the case.

If both agents perform the contributions contained in the plan, and the environment does not change adversely, then plan execution will be successful, and the agents should increase their trust of each other. Conversely, if cooperation fails for some reason, for example, if the intention ceases to be of motivational value to `agent4`, and it drops its intention, then `agent1` should reduce its trust of `agent4`.