— *Thanks* —

The best part about my nine years of university education was not anything I might have learned but all the fantastic people I met along the way. Thanks to the many friends that have made those last nine years so much fun: Morgan, Anth, John and all the guys from St. Bernards; Tim, Matt, Goody, Dave and others from Melbourne University; everyone from The Keg, especially Luke, Zero, Narelle and Rachel; Josh, Sunny and others from RMIT, Erik, Tim, Asmus, Jean-Paul, Thorsten, Cameron, Tuuli, Pirrita, Rupert, Kritter, the Helsingborg Saints and many others from Linköping; and Will, Mike, Jeff, Bob, Monica, Andrew, George, David, Jay, Ranjit, Hyuckchul and all the L.A. Crows in L.A. I have been extremely lucky to have worked with advisors with the enthusiasm, dedication and patience to make studying both challenging and enjoyable. Thanks a lot to: Lawrence, Lin and Vic from RMIT, Milind at USC and especially Anders, Dimiter, Simin and Nancy at Linköping. My family have not always been excited about my travels but have never failed to support and encourage everything I do – knowing that I had that support and encouragement is more important than they probably realize. Finally, a very special thanks to Fiesta who is simply a legend.

CONTENTS

1. Introduction 1
1.1 Autonomy 2
1.2 Adjustable Autonomy 3
1.3 Research Question 5
1.4 Contributions 5
1.5 Outline of the Thesis 6
1.6 An Aside 7

2. Agent Services for Adjustable Autonomy 9
2.1 What is Adjustable Autonomy? 9
2.1.1 What Exactly is an Agent? 10
2.1.2 Autonomy 13
2.1.3 Conceptual Model of the AA Task 18
2.2 Motivation 22
2.3 Research Question 23
2.3.1 Scope of this Work 24
2.4 An Ideal Adjustably Autonomous Agent 26
2.5 Summary 29

3. State of the Art 31
3.1 Applications of Agents 31
3.2 Motivating Adjustable Autonomy 32
3.2.1 When is AA Useful? 32
3.3 Adjustable Autonomy in the Literature 37
3.3.1 Reasoned Autonomy 39
3.3.2 Directed AA 43
3.4 Related Ideas 52
3.4.1 Mixed Initiative Systems 52
3.4.2 Tele-operation 53
3.4.3 Interactive Theatre 55
3.4.4 Human Factors 56

3.5 Summary 58

4. Guidelines for Building Agents for Adjustable Autonomy Systems 59
4.1 Using Guidelines for Capturing Design Experience 59
4.1.1 Using the Guidelines 60
4.1.2 Guideline Philosophy 61
4.1.3 Guideline Evaluation 62
4.2 Building Agents For Adjustable Autonomy Systems 62
4.3 $\mathcal{I}$ Services 63
4.4 AA Actuation Services 67
4.5 Summary 74

5. Adjustable Autonomy for Interactive Simulations 77
5.1 What are Interactive Simulations? 78
5.1.1 Evaluation Simulation Environments 83
5.2 AA Requirements 87
5.2.1 What should AA Information Do? 89
5.2.2 Summary of AA Information Service Requirements . . 90
5.2.3 What should the AA Actuation Do? 90
5.2.4 Implementation Issues 92
5.3 Overview of a Solution 93
5.4 A Note About Agents and Actors 95
5.5 EASE 96
5.5.1 Social Conventions 98
5.5.2 Managing Conflicting Goals 102
5.6 Actor Services and Prototype Interfaces 110
5.6.1 Information Services 112
5.6.2 Actuation Services 122
5.7 Example 132
5.7.1 The Scenario 132
5.7.2 Actor Specification 134
5.7.3 Avoiding Aircraft 137
5.7.4 Suspending Agents 140
5.7.5 Changing Generic Pool 145
5.7.6 Changing Constants 147
5.7.7 User Taking Over High Level Decision Making 152
5.7.8 Detail Traces 154
5.8 Limitations 159
5.9 Other Behaviour/Agent Based Actor Architectures 160
5.10 Summary 162

6. Adjustable Autonomy for Personal Assistants 163
6.1 Electric Elves 165
6.2 AA in Human Collaboration 171
6.2.1 Issues 173
6.3 Conceptual Design of the E-Elves 177
6.4 Implementation Details 190
6.4.1 Model of Organisation 190
6.4.2 Friday Decision Making 194
6.4.3 Partitioning 195
6.4.4 Example – Delaying Meetings 196
6.4.5 Team Decision Making 199
6.4.6 Safe Learning 203
6.5 Using the E-Elves 208
6.5.1 General Observations 209
6.5.2 Individual AA 210
6.5.3 Team AA Evaluation 211
6.6 Personal Assistants 213
6.7 Summary 214

7. Evaluation 217
7.1 Method 217
7.1.1 Scope of Evaluation 219
7.2 Explicit Information Guideline 221
7.2.1 EASE and the Explicit Information Guideline 221
7.2.2 E-Elves and the Explicit Information Guideline 222
7.2.3 Violation of the Explicit Information Guideline 223
7.3 Design Information Guideline 224
7.3.1 EASE and the Design Information Guideline 224
7.3.2 E-Elves and the Design Information Guideline 225
7.3.3 Violation of the Design Information Guideline 226
7.4 Software Engineering Guideline 227
7.4.1 EASE and the Software Engineering Guideline 227
7.4.2 E-Elves and the Software Engineering Guideline . . . 228
7.4.3 Violation of the Software Engineering Guideline 229
7.5 Deterministic Execution Guideline 229
7.5.1 EASE and the Deterministic Execution Guideline . . . 229
7.5.2 E-Elves and the Deterministic Execution Guideline . . 230
7.5.3 Violation of the Deterministic Execution Guideline . . 231
7.6 Explicit Behaviour Guideline 232
7.6.1 EASE and the Explicit Behaviour Guideline 232

7.6.2 E-Elves and the Explicit Behaviour Guideline 233
7.6.3 Violation of the Explicit Behaviour Guideline 234
7.7 Building Blocks Guideline 234
7.7.1 EASE and the Building Blocks Guideline 235
7.7.2 E-Elves and the Building Blocks Guideline 235
7.7.3 Violation of the Building Blocks Guideline 236
7.8 No Extra Mechanisms Guideline 236
7.9 Design Expecting Failures Guideline 237
7.9.1 EASE Features 237
7.9.2 E-Elves and the Design Expecting Failures Guideline . 239
7.9.3 Violation of the Design Expecting Failures Guideline . 239
7.10 Evaluation Summary 240
7.11 Miscellaneous Agent Design Issues 244
7.11.1 The Guidelines are Flexible 244
7.11.2 Choosing between Directed and Reasoned AA 245
7.11.3 Good Software Engineering is Essential – Unfortunately 245
7.11.4 Ironically, Behaviour-based Systems are Very Appropriate 246
7.11.5 Teamwork is a Key 247
7.11.6 Using AA During Development 248
7.12 Summary 249

8. Conclusions and Future Work 251
8.1 Summary 251
8.1.1 Conceptual Model of AA 251
8.1.2 Guidelines 252
8.1.3 Implementations 252
8.1.4 Evaluation 253
8.2 Summary of Contributions 253
8.3 Future Work 253
8.3.1 Tradeoffs 254
8.3.2 Other Domains 254
8.3.3 Underlying Properties 255
8.3.4 AA Implementations 255
8.3.5 EASE and E-Elves Extensions 256
8.4 Concluding Remarks 256

LIST OF FIGURES

2.1 The relationships between goals, goal hierarchies, top-level goals and system goals. 12

2.2 The diagram on the left shows the conventional way of thinking about human-agent interaction – interaction is either via the environment or through some external interface. In the AA system on the right the agent and human are part of the same system that interacts with the environment as a single entity. 16

2.3 The conceptual relationship between AA and a system. The arrows show the paths data travels in the system. 19

2.4 This figure shows how the different conceptual parts of the AA are related to the agent services for a single agent. Dotted lines show examples of how $\mathcal{I}$ services might extract information from different parts of an agent's reasoning and $\mathcal{A}$ services might change different parts of that reasoning. 20

2.5 Directed AA involves a human user doing the reasoning about the distribution of autonomy. 21

2.6 Reasoned AA involves the agent doing the reasoning about the distribution of autonomy. 22

3.1 Diagram relating the different factors affecting function allocation. Balanced work keeps the different components in balance (Bye et al. 1999). 57

5.1 Standard life cycle of an actor. Each time the domain expert requires a change in behaviour the simulation is stopped and an actor expert consulted to make the change. 80

5.2 Actor life-cycle for an EASE actor using AA to change actor behaviour online. Notice that the cycle is much simpler than the one shown in Figure 5.1. 81

5.3 Snapshot of the RoboCup simulation monitor. The larger half light and half dark circles represent the 11 players of each team. The lighter side of the player shows the direction it is facing. In the bottom left hand corner is a record of the commands the automatic referee has sent to the players. In the bottom right hand corner is a record of the commands sent by the players to the server. 84

5.4 Snapshot of the TACSI air-combat simulation. Solid lines show the path of the different aircraft as they engage over the East coast of Sweden. The three aircraft in the bottom right corner are engaging the three in the top left part of the screen. 86

5.5 The conceptual configuration of actor, user and AA for interactive simulations. $\mathcal{I}$ and $\mathcal{A}$ are implemented in software while a human does the $\mathcal{R}$. Some type of human-computer interface needs to connect the software parts to the human reasoning. 88

5.6 This diagram shows the flow of information from the actor, via $\mathcal{I}$ services and $\mathcal{I}$ to a user. When the user receives the information it is in a format they can understand. Across the bottom of the diagram is an example of the types of information representations that might be used at each step. 89

5.7 This diagram shows the flow of commands from the user, via $\mathcal{A}$ and $\mathcal{A}$ services to implementation in the actor. Along the bottom of the figure is an example of how a command from the user might get translated into changes in the actor. . . . 93

5.8 Relationship between an actor, agents and a user. 95

5.9 A high level view of the EASE run-time architecture 96

5.10 Tool for specifying a fixed contract. The manager being specified will make contracts with a safety agent, a conserve fuel agent and a patrol mission agent. The patrol mission agent contract has two parameters, "Rules of Engagement" and "Minimum Height" which are instantiated with the values "Aggressive" and "Safe Altitude" (which are named constants or more complex expressions). 100

5.11 The end-user tool for specifying a dynamic contract. In this case an agent is required with the capability "Safety". 101

5.12 The tool for specification of a list contract. 102

5.13 A snapshot of the condition specification sub-system for end-user programming of complex agent functions, showing a user editing one *cell*, a comparison between other cells, in the center of the window. At the bottom of the window is information about where the cell is used and the cells it uses. On the right of the window are the cells that have already been created. 104

5.14 An example (simple) environmental priority function for an obstacle avoidance agent. The agent's environmental priority is higher if the obstacle aircraft is close and/or the aircraft is heading towards the obstacle. 106

5.15 The end-user state machine specification tool, showing a state machine for a simple patrol agent. Circles represent states, while arrows represent transitions. The state in the top left corner, i.e., take-off, with an extra line is the start state. Success transitions are colored green while normal transitions are annotated with the condition for their traversal. 109

5.16 Information extracted from the agent organisation and generic pool. The surrounding box represents the extent of the EASE actor software. The arrows out of the box show what information is extracted and the parts of the software where that information originates. 110

5.17 Information extracted from a single EASE agent. The large circle represents an agent, the arrows show which information is extracted from where. 111

5.18 How the different AA actuation services affect an actor. The box represents the extent of the EASE software and the arrows show where $\mathcal{A}$ changes occur. 112

5.19 The Boss visualisation of the agent organisation. The state of each agent is shown in parentheses after its name. Agents with a small circle to their left are managers which can be clicked on to show contracted agents. 116

5.20 Implications Visualisation tool for investigating the implications of different actions on the agent organisation. The buttons along the bottom of the window allow the user to specify different changes to the organisation. 120

5.21 The Calculation Trace window shows the details of a calculation one of the agents is performing. Each line shows the result of computing the value of one cell. The names used to describe the cells are used to name the intermediate results. The bottom line in the window shows that the satisfaction calculated was 3.0. 122
5.22 The tool for modifying agents in the generic pool. Unchecking the "Available" checkbox would mean the "Goal Kicker" agent could not be contracted. The list on the left shows all the agents in the generic pool. The slider in the middle allows the intrinsic priority of "Goal Kicker" to be changed. 128
5.23 The tool for changing the value of named constants while an actor is running. The name is on the left, the current value in the middle and a button for making the changes on the right. 131
5.24 A snapshot of the TACSI visualisation at the start of the scenario. 133
5.25 A snapshot of the "Start control" tool from which the actor and other tools are started. 134
5.26 The Boss showing the starting agent organisation of the actor. 135
5.27 The user has selected the *Patrol Mission* agent to be added to the agent organisation. 136
5.28 A snapshot of The Boss after the *Patrol Mission* agent has been contracted. 137
5.29 A snapshot of the TACSI window shortly before the EASE controlled aircraft (labeled "1", at the top) takes evasive actions. 138
5.30 A snapshot of The Boss, with hierarchies fully expanded, after an aircraft avoidance agent has been contracted to avoid a detected aircraft. 139
5.31 A snapshot of The Boss, with hierarchies fully collapsed, after an aircraft avoidance agent has been contracted to avoid a detected aircraft. 139
5.32 A snapshot of the Negotiation Viewer shortly after an aircraft was successfully avoided. The graph shows about two minutes worth of negotiation. 141
5.33 The scenario shortly after an aircraft was successfully avoided. The path of the EASE controlled aircraft (on left) is shown with a trail behind it. Notice the small curve near where the paths of the two aircraft cross showing the slight turn the aircraft made to avoid the obstacle. 142

5.34 A snapshot of The Boss with the *Smooth Manager* agent suspended. 143
5.35 A trace of the aircraft's path during the patrol. The turn at the bottom (one long turn) was made while the *Smooth Manager* was active and the tighter turn at the top was made while the *Smooth Manager* was paused. 144
5.36 The starting state of the generic pool, showing the ten available agents. 145
5.37 The state of the generic pool after a second file of agent specifications has been opened. The generic pool now contains 13 agents. 146
5.38 Making the agent *Go to (x,y)* unavailable to be contracted in the generic pool. 146
5.39 A snapshot of The Boss with the *Go To (x,y) Fast* manager and its contractees. 147
5.40 A trace of the calculations the *Smooth Manager* is doing to determine whether a state transition should be made. 148
5.41 A snapshot of the tool for viewing and changing the values of named constants. 149
5.42 A snapshot of The Boss after the *Smooth Manager* makes a state transition. (Notice that the user has also replaced the *Patrol Mission* agent with a *Circle* agent.) 150
5.43 A trace of the path of the aircraft when the location of one of its waypoints is dynamically changed. The "turn" on the right is actually where the waypoint location was changed to be much further north, rather than north-east. 151
5.44 A snapshot of The Boss after the *Patrol Mission* agent is stopped. 152
5.45 Creating a *Fly Heading* agent and instantiating the "Required Heading" parameter. 153
5.46 Condition Specification tool for specifying calculations to be performed by the actor. 154
5.47 A snapshot of The Boss after the user adds the *Fly Heading* agent. 155
5.48 The route taken by the aircraft after the user stepped in and started making high level decisions. Notice the turn to the east near the left of the screen. 156
5.49 A trace of a *Go to (x,y)* engineer's reasoning for its satisfaction with the selected actor action. 157

5.50 A trace of some of the changes in the agent organisation and the reason for those changes. 158
5.51 A trace of a *Hard Deck* engineer's reasoning for its priority. . 159

6.1 A conceptual visualization of the relationships between humans and agents, human teams and agent teams. 163
6.2 Basic architecture of the E-Elves. Agent proxies, supporting their users activities, communicate via *broadcast nets.* 165
6.3 Dialog for delaying meetings. $\mathcal{R}$ has decided that the human should decide what action should be taken. Friday is asking the user, via a dialog on their workstation, what meeting delay action should be taken (if any). 166
6.4 Friday, the user's agent proxy, reporting that is has ordered dinner from California Pizza Kitchen. 166
6.5 A webpage showing that Ranjit Nair is not currently at ISI (as of 14:28 on 03/09/01). 168
6.6 The E-Elves auction tool. The top left corner shows the meeting involved in the auction. To the right of the meeting name is the role being auctioned. Underneath the meeting name is the team that will attend the meeting. Empty spaces in the bid information are where information has yet to arrive. . . 170
6.7 Palm VII connected to GPS device. 171
6.8 Conceptual design of AA for the Friday proxy agent. This figure shows more detail of one of the agents as shown in Figure 6.1. 178
6.9 Conceptual design of AA for agent team to human team interaction. 182
6.10 Part of the environment state space for an episode where a user should attend a meeting. Final states are shaded. Arrows between states show *some* of the transitions that might occur. 184
6.11 Part of the state space for the case where the user is asked for input. In the shaded state the user was asked for input. State transitions are labeled with the reply from the user. . 186
6.12 Part of the state space with some of the transitions that might occur if Friday acts. Arcs, the meaning of which is shown in the legend, show a selection of possible outcomes of different actions Friday might take. 187
6.13 A graph of the number of states in the autonomy policy where a human is asked for input versus the cost of asking. 193

6.14 A graph of the number of states in the autonomy policy where a human is asked for input versus the probability of the user responding to the request. 194
6.15 A high level view of the conceptual information flow for team decisions. The text next to the arrows shows the information passed. Numbers indicate the ordering of messages. 201
6.16 A Venn diagram showing the relationships between the policies allowed by pre- and post- learning constraints and the set of allowable policies. 206
6.17 Diagram showing the effect of the post-learning check on the policies produced by a learned set of parameters. The dark circle represents the original policy. The solid line shows the change due to learning and the dotted line shows the changes due to the post-learning check. 207
6.18 A graph of the number of meetings monitored by Friday for each user. 210
6.19 A graph of the number of meeting delays that were done autonomously vs. the number of delays initiated by the user. 211
6.20 Number of daily coordination messages exchanged by proxies over three-month period. The y-axis shows the number of messages exchanged between proxies and the x-axis shows the date. 212

LIST OF TABLES

2.1 Summary of definitions presented so far. 16

3.1 A listing of the AA systems that are discussed below with a brief description of their functionality. 37
3.2 A selection of $\mathcal{I}$ features and the underlying agent features that make them possible. 49
3.3 A selection of $\mathcal{A}$ features and the underlying agent features that make them possible. 50
3.4 Summary of $\mathcal{R}$ for the surveyed systems. 51

4.1 Summary of the agent features intended to be encouraged by each guideline and why those features are useful to AA. . . . 75

5.1 Selection of EASE actor features and the $\mathcal{I}$ service they provide. 94
5.2 Selection of EASE actor features and the $\mathcal{A}$ service they provide. 95
5.3 The three different contract types a manager can make and their basic functionality. 99
5.4 Some examples of how overall priority is a function of environmental, organisational and intrinsic priority for a priority function in the obstacle avoidance agent. 107
5.5 Summary of the different aspects of an agent and their effect on overall agent behaviour. 108
5.6 The table shows the conditions under which an engineer and manager will take success or failure transitions. 109
5.7 Summary of some of the useful EASE actor features resulting from adherence to each guideline and the subsequent AA facilities. 114
5.8 The relationships between information provided by the system, the property of EASE that makes that information available and the end user tool that leverages that information. . . 115
5.9 The table maps the different actuation services offered by an EASE actor to the resulting changes in behaviour. 123

5.10 An example of how exchanging one agent with another with the same capability can affect a RoboCup actor's behaviour. The first column lists the events that triggered changes. The second column lists the state of some striker agent responsible for trying to make the player kick goals. The third column lists the contracts the striker agent has. The final column lists part of the contents of the generic pool. 130

6.1 Summary of the E-Elves features resulting from adherence to each of the guidelines and the features of the AA that utilized the features. 189
6.2 Hand-coded probabilities of all possible different lengths of time user will be late to a meeting, given their location five minutes before the meeting. 191
6.3 Example hand-coded costs and rewards for the meeting scenario. 192
6.4 Part of the combined decision making and autonomy policy for meeting delaying. There are 736 states in the complete policy. 200
6.5 Part of the auction closing policy. The "Difference" column shows the difference between the quality of the best bid and the second best bid. The total auction policy has over 1300 states. 204
6.6 Results of the team auction for the presenter role at the group's weekly meetings, during a three month period. 213

7.1 Summary of the features in the two implementations led to by each of the guidelines. 218
7.2 Summary of the EASE actor features led to by each guideline and the AA facilities they support. 241
7.3 Summary of the ELVES actor features led to by each guideline and the AA facilities they support. 242
7.4 Summary of the examples of violations of the guidelines and the problems the violations cause. 243

1. INTRODUCTION

Artificial Intelligence (AI) technology is rapidly starting to impact our everyday activities, both at work and in our personal lives. AI systems that have the ability to affect an environment they share with users, change the user's experience of that environment – for better or worse. For example, agents in a future home might adjust the temperature of a house to suit the time of day (Lesser et al. 1999). Given their increasing impact on our lives, it is important to carefully examine how users interact with intelligent systems. We need to aim to maximise the utility of intelligent systems while minimizing any harmful effects and making absolutely sure that humans stay in control of their own environments.

Many recent, exciting developments in AI have centered around the concept of an *agent*. An agent is an autonomous entity that senses its environment and acts intelligently and pro-actively towards its goals (Wooldridge & Jennings 1994, Bradshaw 1997). An important characteristic of an agent is that it has the ability to take actions that affect its environment (Russell & Norvig 1995). Some agents sense and act in purely software environments (e.g., an operating system monitoring agent (Song et al. 1996)), while others have a physical embodiment and inhabit a physical environment (e.g., a museum tour guide robot (Burgard et al. 1998)). Because an agent can act, it can be assigned tasks that it can potentially do more quickly, efficiently, cheaply or safely than a human can. Thus, humans are freed from menial, dangerous and/or boring tasks. Despite the long list of successful agent applications, it is unlikely we have seen more than a sample of the ways intelligent agents can change the way we live our lives (Pell et al. 1998, Rybski et al. 2000). In the future, autonomous agents will take on more complex tasks and act more intelligently and more decisively (Jennings & Wooldridge 1998).

Despite the obvious benefits of agent technology (Jennings 1999), we, as humans, should not rush blindly into handing over control of our environment to an army of intelligent agents – unless we are sure we can get back control when required. Because an agent can *act* it has the potential to cause harm. As with any intelligent entity in a complex environment, an agent will

occasionally make mistakes and unintentionally cause harm. As a society we are unwilling to accept any harm caused by malfunctioning technology (Asimov 1950). Whether the harm be actual physical harm, e.g., physical injury to humans or damage to equipment, or non-physical harm, e.g., financial cost, inconvenience, loss of control or mis-trust, it is unacceptable. However, it is precisely the ability to act that makes the concept of an agent so compelling, but that ability opens up the possibility for harm. Hence, a key challenge for AI researchers is to find ways of leveraging the potential of agent technology without paying a high price for its shortfalls.

1.1 Autonomy

A central concept when discussing agents is that of *autonomy* (Wooldridge & Jennings 1995). Intuitively, autonomy is independence from constraints when selecting and taking actions. More formally, we define the autonomy of an agent by its responsibilities, its authorization to act and its intrinsic capabilities. The more *responsibility* an agent has the more autonomous it is. Likewise, the more an agent is *allowed* to do and is *capable* of doing, the more autonomous it is. (A more detailed discussion of autonomy is given in Section 2.1.2.)

As AI technology has developed, agents have been given correspondingly more autonomy, i.e., more capable agents have been given more authority and more responsibility. In most cases, once an agent's autonomy is determined by a system designer the agent is left to fulfill its responsibilities according to its specification, within the bounds of its authority and capabilities. In complex environments, an agent will be faced with a vast range of situations, in each of which it must act to the net benefit of the users. It is unlikely that we can build any agent for a complex environment that has appropriate reasoning mechanisms, sensors and actuators for it to act appropriately in *all* the situations it faces (Shneiderman 1998, Lanier 2001, Brann et al. 1996). There will be some situations in which the agent will make unacceptable decisions and, hence, take unacceptable actions – potentially causing harm. Some situations the agent finds itself in may be so unlikely that the designer can reasonably choose to ignore them, hence if the situation *does* arise the agent may not act properly. Other situations might commonly occur but to build software or robotic hardware to properly handle the situation may be too expensive or time-consuming to justify. Yet other situations will be unacceptably handled by an agent due to "bugs" in its software. Importantly, for the same reasons that an agent handles a situation badly, it may not detect that it has encountered a situation it is

not capable of dealing with properly.

The more autonomy an agent has, over more complex and critical tasks, the more potentially harmful the consequences when its behaviour is unacceptable. However, because the agent is autonomous and because it may not detect when it is acting inappropriately, in some cases "killing" the agent may be the only way of preventing the incorrect actions from being taken. Completely stopping an agent means another entity needs to take over the agent's responsibilities, something that may be unreasonable in a complex physical system (e.g., a spacecraft (Bernard et al. 1999)). Even if it were possible for a human to take over, requiring substantial human input whenever some small thing goes wrong, reduces the benefits of having autonomy in the first place (Brann et al. 1996). A more desirable scenario is to allow an outside entity to take responsibility for the *parts* of the agent's behaviour that are inappropriate while allowing correctly functioning parts to continue autonomously.

Thus, agent developers are faced with a challenging dilemma. For some applications, autonomous agents can be very useful in very many situations, most of the time. But in a small number of situations, a small percentage of the time, agent behaviour will be unacceptable and potentially have harmful consequences. Hence, agent based system developers need to employ techniques that allow agents to be safely deployed even though there may be some situations where the agent will act inappropriately.

1.2 Adjustable Autonomy

One stage of the system design process is to allocate functionality and responsibility to the users, software and hardware that make up the system (Bye et al. 1999). In the case of an intelligent agent-based system, this process includes assigning autonomy, e.g., responsibility, to the various entities in the system. For example, in an intelligent aircraft control system, the human pilot might be given responsibility for choosing a destination and an agent given responsibility for guiding the aircraft to that destination. The allocation of responsibility and authority, i.e., the allocation of autonomy, is fixed for the lifetime of the system. This implies that there is limited flexibility in the overall system to react to unusual circumstances.

Adjustable Autonomy (AA) is a recent idea meaning to *dynamically adjust the level of autonomy of an entity depending on the situation* (Musliner & Krebsbach 1999). Instead of the responsibility and authority of individual entities being fixed at design time, they can be changed to best configure the overall system's autonomy distribution for the current situation (Bar-

ber & Martin 1999*a*). The idea is to dynamically assign autonomy to best leverage the constituent entities' strengths and avoid their weaknesses in the current situation. Thus, *an Adjustable Autonomy system is an intelligent system where the autonomy is distributed dynamically amoung the system's entities to optimize overall system performance.* For example, if a human pilot notices that a collision is imminent with a flock of rare ducks that the autopilot cannot detect, the pilot might like to slightly alter the aircraft's course without taking over all the details of the aircraft's functioning. The pilot might adjust the aircraft's heading slightly without taking over control of altitude, speed or radar settings.

Flexible assignment of autonomy means a system can deal with a wider range of situations more effectively (Kortenkamp et al. 2000). Specifically, a human user can assume responsibility for aspects of an agent's task the agent is performing inadequately, by reducing the agent's autonomy. Thus, AA allows the intelligence and autonomy of agents to be exploited without the users being saddled with the agent's inadequate decision making when situations occur the agents cannot handle (or humans could handle better).

The basic tasks necessary to achieve AA are to collect information relevant to autonomy decision making, deciding how autonomy should be distributed and distribute the autonomy to optimize performance. The reasoning process which decides on suitable autonomy configurations can be performed either by special purpose software or by a human user. The agents in the system need to provide two critical pieces of functionality for the AA. Firstly, they must provide the information about their internal functioning. Secondly, they must be able to smoothly change their autonomy when such a change is required. How well agents provide this functionality affects how well the AA can work.

Chapter 5 presents an application of AA in the domain of simulation environments. In that system autonomy is dynamically assigned either to an intelligent software actor or a human domain expert depending on whether the simulated intelligent entity is behaving as required. Chapter 6 presents an application of AA in the domain of agent based human collaboration support software. For that application the ability to dynamically take autonomy away from personal assistant agents means the agents are not forced into taking risky decisions when they are unsure or the costs of incorrect action are high.

1.3 Research Question

This thesis look in particular at the *services* that agents must provide so that AA reasoning can be implemented with a minimum of effort. The AA services are the interface to the agent that provide the foundation for an AA implementation. AA services provide two key pieces of functionality. Firstly, they provide the *information* used for AA reasoning. Secondly, AA services handle requests for changes in the agent's autonomy.

The limitations on the services agents provide, limit what can be achieved by any AA implementation. Whatever information about an agent the agent's services do not provide cannot be used in the AA reasoning. Likewise, decisions reached by the AA must be able to be realized by an agent or it is useless reaching such decisions. Moreover, the effort required to build effective AA is directly related to how easy it is to extract information from, and enforce autonomy changes in, the system's agents. Hence, the potential of the AA is inextricably dependent on the services provided by the system's agents. The focus of this thesis is on the *relationships* between the design of an agent, the AA services the agent provides and the consequent bounds on implementations of AA.

To date most AA has been implemented post-hoc, on top of already existing agent-based systems (Bonasso 1999). The agents were often designed without regard to the AA that was subsequently developed. An unfortunate consequence of that process is that the AA can be more difficult to develop than it needs to be or even, in the worse case, fail to meet its requirements at all. In other cases, agents might need to be redesigned to facilitate the AA, e.g., (Ferguson & Allen 1998). One of the reasons for the sub-optimal development process is that the relationships between agent design and AA are not well understood, hence it is difficult for implications of agent design decisions on AA to be taken into account when designing an agent. This work provides information on the impact of agent design decisions on the potential of AA, which will allow designers to better take AA into account when designing agents. Hence, this work addresses an important issue for the field.

1.4 Contributions

This work makes three specific contributions to the AA field.

Agent Design Guidelines for AA. The central contribution of this work, presented in Chapter 4, is a set of guidelines advising how agents should be

designed so that AA can be implemented effectively and straightforwardly. The guidelines capture our experiences on the impacts of agent design decisions on AA in a form that allows others to leverage the knowledge in their own designs. In particular, the guidelines allow agent designers to understand, early in the design process, the impacts of their design decisions on the ease with which AA can be built.

Prototype Implementations of AA. Secondly, this work contributes descriptions and *evaluations* of two *implemented* AA systems. Detailed descriptions of the agent designs, AA services and AA for the two systems contribute case studies on the impact of agent design decisions on AA implementations.

Agent Services and AA. Finally, this work contributes a study of the *relationships* between an agent's design, the AA services supported by that design and the limitations and simplicity of the AA implementation due to the properties of those AA services.

1.5 Outline of the Thesis

The body of the thesis is organised as follows:

Chapter 2 introduces the important concepts used in this thesis and sets out the specific research question being addressed.

Chapter 3 surveys AA and related literature, discussing this work in relation to other work in the area.

Chapter 4 presents specific guidelines for the design of agents that will be part of AA systems (Scerri & Reed 2001). The guidelines make it possible to make informed tradeoffs when designing agents for AA systems. Agents implemented according to the guidelines are described and evaluated in Chapters 5 and 6.

Chapter 5 presents EASE, a tool for developing intelligent actors for interactive simulation environments (Scerri & Reed 2000*a*,*b*). A human domain expert uses AA to dynamically change the behaviour of actors in a simulation by assuming parts of the decision making responsibility.

Chapter 6 presents the E-Elves, an AA system for streamlining daily activities in human organisations (Scerri, Pynadath & Tambe 2001, Pynadath et al. 2001). The E-Elves uses AA to manage the interactions between human users and their personal assistant agents as well as between human teams and their counterpart agent teams. In particular, the AA reasoning, implemented by software, balances the potential costs and benefits of autonomy by switching decision making responsibility dynamically between users and agents.

Chapter 7 provides an evaluation of the guidelines. The evaluation consists of showing features of the agents in the two applications that are specifically the result of either following or violating each of the guidelines. The evaluation shows that when the guidelines are followed AA is more straightforwardly implemented, while when they are violated problems ensue.

Chapter 8 describes interesting future lines of work and summarises the conclusions and contributions of the thesis.

1.6 An Aside

It is fitting that in the year 2001, a thesis is written discussing techniques for ensuring humans stay in control of intelligent, autonomous software. In 1968, Stanley Kubrick and Arthur C. Clarke made the classic science fiction movie, *2001:A Space Odyessy*. In that movie, set in the year 2001, an intelligent software assistant (i.e., an agent), named HAL, supporting a group of astronauts on a long distance space flight becomes neurotic and begins to "fear" the astronauts it is supposed to be helping. Eventually, HAL kills an astronaut to protect itself. Though many of the movie's predictions for 2001 have not been borne out, AI researchers continue to make significant progress towards building intelligent agents exhibiting the exciting, benevolent properties HAL was supposed to have. Though some people believe that nightmare scenarios like those predicted in 2001 may one day become reality (Joy 2000), most people are more optimistic. AA is one emerging technology that we believe will allow the human race to harness the benefits of 'HAL-like' systems without fear.

2. AGENT SERVICES FOR ADJUSTABLE AUTONOMY

An Adjustable Autonomy (AA) system has the capability to dynamically change the autonomy of the intelligent entities of which it is composed. An AA capability is useful in a wide range of domains for a range of different purposes. We look at some of those uses in Chapter 3. The interest generated by the wide variety of uses has led to various mechanisms being built for, and a wealth of knowledge accumulated about, AA. However, little has been reported about the *services* that need to be provided by an agent in order for effective AA to be straightforwardly developed. Agent services provide information required for AA decision making and the mechanisms by which changes in autonomy are realized.

The aim of this work is to investigate how the features an agent has affect the ease with which AA can be implemented and capture our knowledge of those effects in a way that allows others to leverage that knowledge to build agents suited for AA systems.

This chapter defines AA and describes the research question being addressed by the thesis. First, some important underlying concepts, like *agents* (Section 2.1.1), *autonomy* (Section 2.1.2) and *level of autonomy* (Section 2.1.2) are defined. Building on those definitions, Section 2.1.2 gives a mathematical description of AA. With the definitions presented we turn our attention to introducing and motivating the rest of the thesis. Section 2.3 presents the precise research question being addressed and the scope of the work. We conclude the chapter with an example of an "ideal AA system" to give an intuitive idea of some of the complexity of building AA and the features agents for AA systems need to have.

2.1 What is Adjustable Autonomy?

In this section the concept of *Adjustable Autonomy* is formally defined. We start by giving formal descriptions of an *agent*, *agent autonomy* and *level of autonomy* and then use those definitions to define AA. The aim of the section is to provide a formal description of AA that can be used throughout the rest of the thesis.

2.1.1 What Exactly is an Agent?

The word *agent* is commonly used to describe a piece of intelligent software – but what exactly is an agent? A wide range of definitions have appeared in the research literature from very broad, e.g., "something that perceives and acts" (Russell & Norvig 1995), to much stricter definitions like "essentially conscious entities that have feelings, perceptions, and emotions just like humans" (Huhns & Singh 1997). For the purposes of this work we will use the following definition:

Definition 2.1: An agent is a software artifact that senses, reasons intelligently about how to achieve its goals and acts, all within some environment.

The "senses" aspect of an agent means it can reason based on the current (perceived) state of the world (as opposed to an "autistic" agent (Huhns & Singh 1997)). The goal directed aspect of the definition differentiates agents from routine computations and "slaves" to other entities (Luck & d'Inverno 1995). The intelligence aspect implies flexibility in the courses of action an agent can take, making it an interesting thing to interact with and providing for a richness of interaction experiences (Dorais & Kortenkamp 2001). The precise level of intelligence, nature of the reasoning mechanisms, etc. are not, however, in themselves relevant. The action aspect of the agent definition separates agents from tools that might sense an environment and reason about it but are passive. The ability to act is important because it means the agent can actually achieve its objectives, perhaps without human intervention, in its environment, while an inability to act restricts an entity to the role of (intelligent) advisor.

Despite the informal definition of an agent being sufficient to describe the intuition behind the idea, in order to define AA formally we need a more precise definition of the properties of an agent so we can define its autonomy. For simplicity, we choose to only model the characteristics of an agent that are important for our definitions of AA. It turns out that these abstract characteristics are common to all the intelligent entities in a system. Hence, we use the same definition for all intelligent entities, be they agents or humans. We define an intelligent entity, e, conceptually by what it can do and what it is doing, i.e., :

Definition 2.2:

$$e \equiv (G, A), G \subseteq A$$

where G is the set of goals for which the entity e has decision making control and A is the set of goals that the entity can potentially achieve. For

all $g \in G$ it is the entity's responsibility to make decisions about the achievement of that goal. An entity's decision making process will result either in atomic actions being taken or sub-goals requiring further refinement being produced. Sub-goals produced during decision making may be the responsibility of the decision making entity that produced them or the responsibility may be given to another entity in the system. Hence, goal hierarchies may transcend single entities (see Figure 2.1) but stay within system boundaries (Musliner & Krebsbach 1999).

The relationship between a goal and a sub-goal is captured by the predicate:

$$parent\_goal(p, g) \equiv \text{ g is a sub-goal of p}$$

Each entity may have decision making responsibility for goals for which it does not have responsibility for the parent goal, i.e., $g \in G$, but $parent\_goal(p, g)$, $p \notin G$. Those g for which the entity does not have responsibility for the parent goal are the *top level goals* of the entity. A predicate to determine if a goal is a top level goal can be defined as:

Definition 2.3:

$$is\_top\_level\_goal(g) \equiv \forall p \in G, \neg parent\_goal(p, g)$$

and, hence, the set of top level goals, T, for an entity is :

Definition 2.4:

$$T = \{g : g \in G, is\_top\_level\_goal(g)\}$$

Finally, a function that returns a set of top level goals given a set of goals, X, can be likewise defined :

Definition 2.5:

$$top\_level\_goals(X) = \{g : g \in X, is\_top\_level\_goal(g)\}$$

For some goals there will be no parent goal at all, in any of the system's entities. These are the *system goals.*

The A component of Definition 2.2 is the set of goals which e has the *ability* to achieve. The ability of an entity to achieve a specific goal is a static property of that entity, the goal and the system configuration, indicating

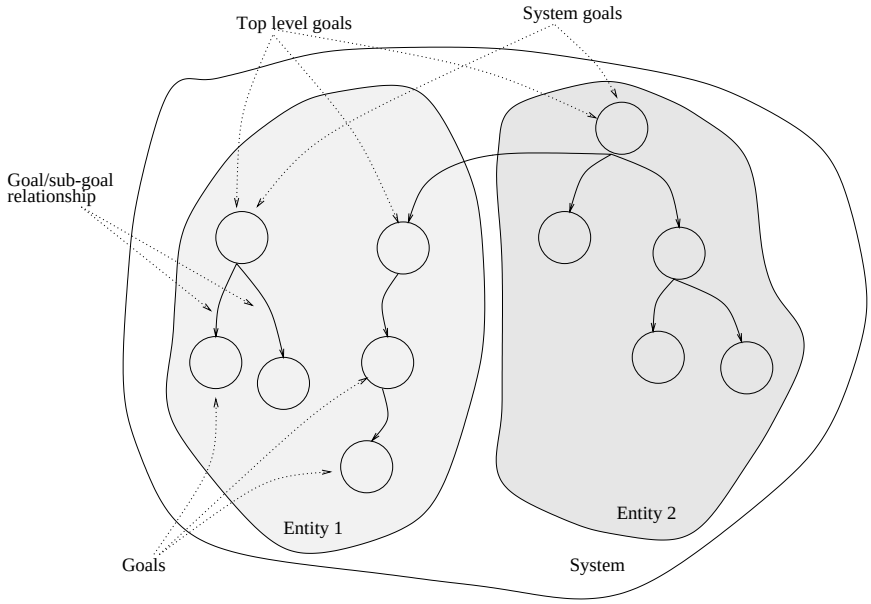

Fig. 2.1: *The relationships between goals, goal hierarchies, top-level goals and system goals.*

that the agent has the know-how, sensing, actuation capabilities, communication channels and so on, to achieve the goal. The ability to achieve a goal assumes that all system resources, including other entities will be available to "help out" with the task. For example, a carpenter is capable of building a house if there is an electrician available to do the wiring. The set A does not reflect potential goal conflicts, environmental circumstances, etc. that might prevent the goal from being achieved at a particular point in time. Hence, A is like an "upper bound" on exactly what the entity can achieve – if a goal is not in A it can never be achieved, if it is in A it may be possible for the entity to achieve the goal if it tries to. So, if at any point in time an entity in a system does not have the ability to achieve a goal it will never have that ability.

Clearly, a rational entity should not assume goals it cannot possibly achieve (Hexmoor & Kortenkamp 2000), i.e., :

$$\forall g \in G, g \in A$$

We will use the following conventions. p and g will refer to arbitrary goals. e and a refer to an arbitrary entity and agent respectively. G, A and C will be subscripted with the name of the entity they refer to, e.g., the set A for an entity e will be A_e. If no subscript is used we are referring to the set in general. Finally, we refer to an entire, arbitrary AA system by S.

2.1.2 Autonomy

Given the above description of an entity what does it mean for that entity to have *autonomy*? In informal terms autonomy means "independence from supervision" but a more formal definition is harder to agree on (Shoham 1998, Huhns & Singh 1997). A fairly diverse range of definitions have appeared in the literature, we briefly present only a sample here (a more detailed look at the range of definitions of autonomy can be found in Chapter 3). Barber & Martin (1999*a*) define agent autonomy as "an agent's active use of its capabilities to pursue its goals, without intervention by any other agent in the decision making processes used to determine how those goals should be pursued". Russell & Norvig (1995, pg. 35) use a far stronger definition – "a system is autonomous to the extent that its behaviour is determined by its experience", a view shared by Steels (1994). Huhns & Singh (1997) view autonomy as a function of the constraints the environment imposes on the agent's behaviour.

The above definitions hint that autonomy is not a binary property, i.e., an entity need not be completely autonomous or completely non-autonomous,

but may have some intermediate level of autonomy (Blumberg 1997*b*). For this reason it is useful to define an entity's *level of autonomy*.

Level of Autonomy

Similar to Barber & Martin (1999*a*) and others we consider the *responsibility* and *ability* of an agent to perform a task to be an important aspect of its autonomy. We also believe that *authority* must be a critical aspect of any understanding of agent autonomy (Musliner & Krebsbach 1999). Formally, we define an entity, e's, level of autonomy, Λ_e, as :

Definition 2.6: $\Lambda_e = (T, C, A)$

where T is the set of top level goals for the entity, i.e., $top\_level\_goals(G_e)$, the C is the set of goals the entity e does *not* have the authority to take on and A is the set of goals the entity can achieve, as defined above. Any goal that is in the set C may not be pursued by the entity, i.e., $\forall g \in C \implies g \notin G$. The goals in C are the ones that the entity does *not* have *permission* to pursue. Clearly, $A - C$ is the set of goals the entity *can* pursue as it has both the authority and ability to do so.

In words Definition 2.6 means:

Definition 2.7: An entity's level of autonomy is defined by the top level goals it has decision making responsibility for and the goals it has the authority to assume and the ability to achieve.

The definition implies that the more authority, responsibility and ability the entity has the more autonomy it has. Conversely, the less authority, responsibility or ability an agent has the less autonomy it has. Notice, however, this is only a partial ordering, e.g., we cannot meaningfully compare the relative autonomy of two entities with the same T but different C and A, but we can compare the relative autonomy of entities with equal C and A and where the set T of one entity is a strict subset of the set T of the other.

The autonomy of an entire AA system, S, is the composition of the autonomy of the constituent entities. Composition of autonomy is defined in the following way :

Definition 2.8:

$$\Lambda_n \circ \Lambda_m = (top\_level\_goals(T_n \cup T_m), C_n \cap C_m, A_n \cup A_m)$$

So, for a system of n constituent parts :

Definition 2.9:

$$\Lambda_S = \Lambda_1 \circ \Lambda_2 \circ \ldots \circ \Lambda_n = (top\_level\_goals(\bigcup_{i=1..n} T_i), \bigcap_{i=1..n} C_i, \bigcup_{i=1..n} A_i)$$

The top level goals of the system are those goals for which there is not a parent goal anywhere in the system. These are precisely the goals for which the *system* has decision making responsibility.

The constraints on the overall system are only those constraints that are constraints on all of the constituent parts, i.e., if any entity has the authority to perform g then the system has the authority to do g. Further, we assume that if the system has the authority to do g any entity in the system *could* be given the authority to do g (of course, it should also have the ability). However, often certain entities will not have authority although the system does. This is analogous to an organisation which has, say, the ability to withdraw money from its bank accounts – anyone in the organisation *could* be given the authority to withdraw money but only some *will* be given the authority.

The set of abilities of the system, A_s, is the union of the abilities of the system's constituents, i.e., anything any entity in the system can do the system is able to do. Notice that there may be some things that entities can do *only* because certain other entities with complimentary abilities are also in the system. This means that for some entity, e, $A_e = f(S)$, i.e., the abilities of an entity are a function of the system it is a part of. For example, an entity may have the ability to lift a heavy object only because it knows there is another (strong!) entity in the system that could help.

Adjustable Autonomy – A Formal Definition

Table 2.1 summarises the definitions presented so far. Building on the formal definitions of agents and autonomy it is possible to make a formal definition of AA. The special case we are predominately concerned with is that of a single human user interacting with a single intelligent agent, though other configurations consisting of only humans or agents or many of each are also possible. It is further assumed that all parties in the system work together towards some goals in some environment. Finally, we assume that the computational costs of reasoning about autonomy are negligible compared to the "normal" computation of the system, hence we do not concern ourselves

Concept	Definition
set of goals for which e has decision making control	G_e
set of goals e has the ability to achieve	A_e
set of goals e may not pursue	C_e
entity	$e = (G, A), G \subseteq A$
top level goal	$is\_top\_level\_goal(g) =$ $\forall p \in G, \neg parent\_goal(p, g)$
top level goals of an entity	T
entity autonomy	$\Lambda_e = (T, C, A)$
system autonomy	$\Lambda_S = \Lambda_1 \circ \Lambda_2 \circ \ldots \circ \Lambda_n =$ $(top\_level\_goals(\bigcup_{i=1..n} T_i),$ $\bigcap_{i=1..n} C_i, \bigcup_{i=1..n} A_i)$

Tab. 2.1: *Summary of definitions presented so far.*

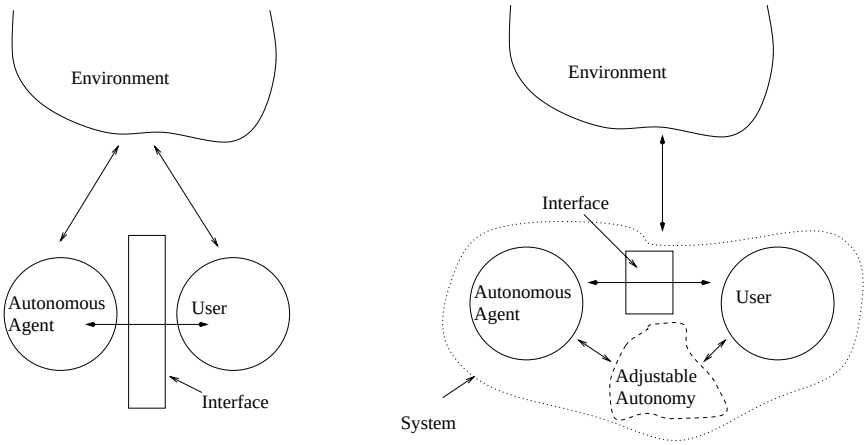

Fig. 2.2: *The diagram on the left shows the conventional way of thinking about human-agent interaction – interaction is either via the environment or through some external interface. In the AA system on the right the agent and human are part of the same system that interacts with the environment as a single entity.*

with "meta-meta-reasoning" about whether to do AA reasoning at all or how much effort to spend on it.

Figure 2.2 shows the conceptual AA view of the agent and human together interacting with the environment rather than working as individuals with some interface inbetween them. The AA component transfers autonomy between the parties as required. To the outside world the system is a black box where the internal workings are irrelevant. The system has autonomy which is can distribute among its constituents without affecting its externally observed autonomy.

For a closed system, i.e., one where no entities are added or removed over time, the set of top level goals will not change over the life of the system, i.e., we assume that the system does not spontaneously spawn new top level goals for itself (Barber, Goel & Martin 2000). As the environment changes so will the breakdown of goals into sub-goals, the priorities of the goals and hence the priority of the entities responsible for the goals, but the top level goals will not change, i.e., T_s is constant. For example, a pilot's top level goal might be to complete a mission without crashing. While the particular tasks being undertaken to reach the top level goals will change over time the top level system goals will not.

To say a system (or entity) can change whether it has the authority to do something is illogical, by definition. To not have authority to do something is meaningless if you can give yourself authority when desired. This is effectively saying you cannot do g unless you want to do g in which case you can give yourself permission to do it. Hence, the authority constraints on the closed system will not change over time, i.e., C_S is constant.

Finally, by definition, the abilities of the system do not change over time, i.e. A_S is constant. Hence, because neither the authority, top level goals nor the abilities of the system change *the autonomy level of the system is fixed for the lifetime of the system. However, in an AA system, the distribution of the decision making responsibility and authority constraints may change over time.* So :

$$\Lambda_s(t_0) = \Lambda_1(t) \circ \Lambda_2(t) \circ \ldots \circ \Lambda_n(t) \tag{2.1}$$

To summarise what has been said so far, an AA system can dynamically change which entities are responsible for the achievement of which goals by transferring decision making responsibility. Furthermore, an AA system can dynamically change the authority constituent entities have to take on particular goals. However, despite the relative autonomy of entities within the system changing, the autonomy of the overall system is constant.

A key problem to be addressed when building AA is to determine an appropriate distribution of autonomy and provide mechanisms to realize the autonomy changes.

The distribution of autonomy should change according to the current situation and sub-goals, reconfiguring so as to best organise the system resources to achieve the system's goals.

2.1.3 Conceptual Model of the AA Task

The task of distributing autonomy, i.e., AA, can be conceptually broken into three components:

- Information ($\mathcal{I}$) : Collection of the information relevant to the AA decision making.
- Reasoning ($\mathcal{R}$) : Reasoning about what autonomy changes should be made.
- Actuation ($\mathcal{A}$) : Realization of the decisions made by $\mathcal{R}$.

The conceptual AA model and its relationship to an AA system is shown in Figure 2.3. $\mathcal{I}$ provides information on prevailing environmental conditions, current system state and potential (referred to as *context* in (Hexmoor 2000*a*)) that are relevant to $\mathcal{R}$. $\mathcal{R}$ determines what changes in the autonomy distribution might lead to better system performance with respect to the system's goals. $\mathcal{R}$ might be done either by a human or by software or a combination of both. Finally, $\mathcal{A}$ provides the mechanisms for implementing the decisions of $\mathcal{R}$, i.e., it provides the mechanisms for realizing changes in authority or transfer of responsibility.

$\mathcal{I}$ and $\mathcal{A}$ tightly constrain the design and potential of $\mathcal{R}$. Any information not supplied by $\mathcal{I}$ cannot be used in determination of appropriate autonomy configurations. Likewise, any change that cannot be realized by $\mathcal{A}$ should not be considered by $\mathcal{R}$. In turn, $\mathcal{I}$ and $\mathcal{A}$ are both tightly constrained by the services provided to them by the entities in the system. Any information the entities cannot provide cannot be supplied by $\mathcal{I}$ to $\mathcal{R}$. Similarly, any autonomy change the entities cannot accept could not be implemented by $\mathcal{A}$. Hence *the services provided by the entities are critically important to the building of an AA system.* Figure 2.4 shows how the agent AA services constrain the realisation of AA. In this work, we are obviously limited to designing the services that software entities provide as the services of the human entities are fixed.

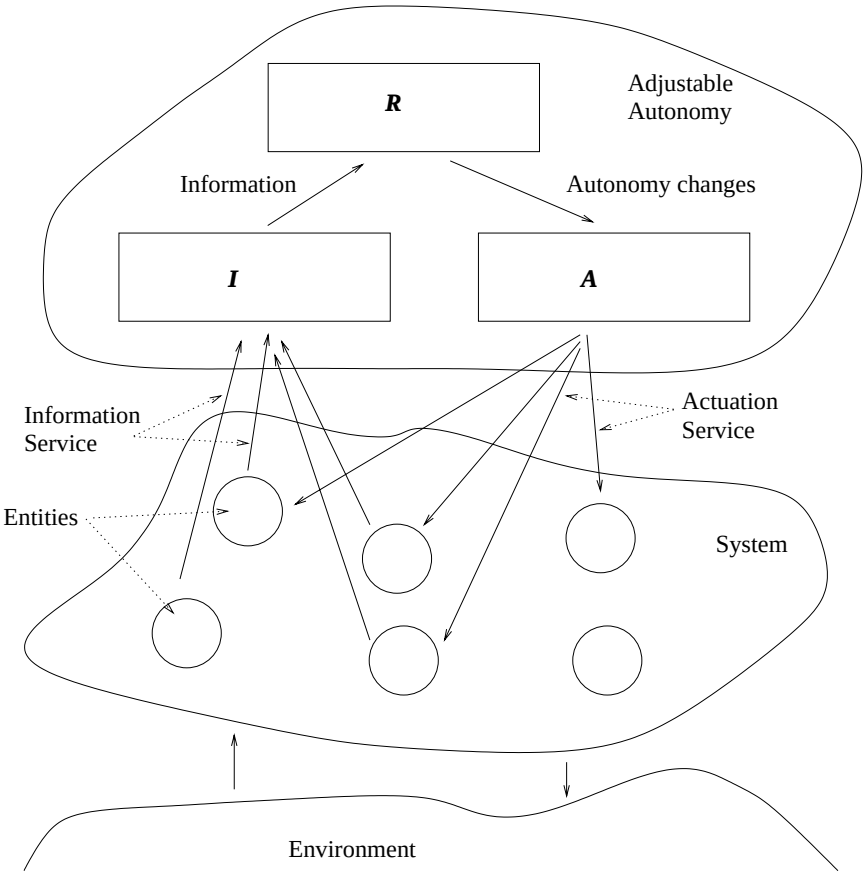

Fig. 2.3: *The conceptual relationship between AA and a system. The arrows show the paths data travels in the system.*

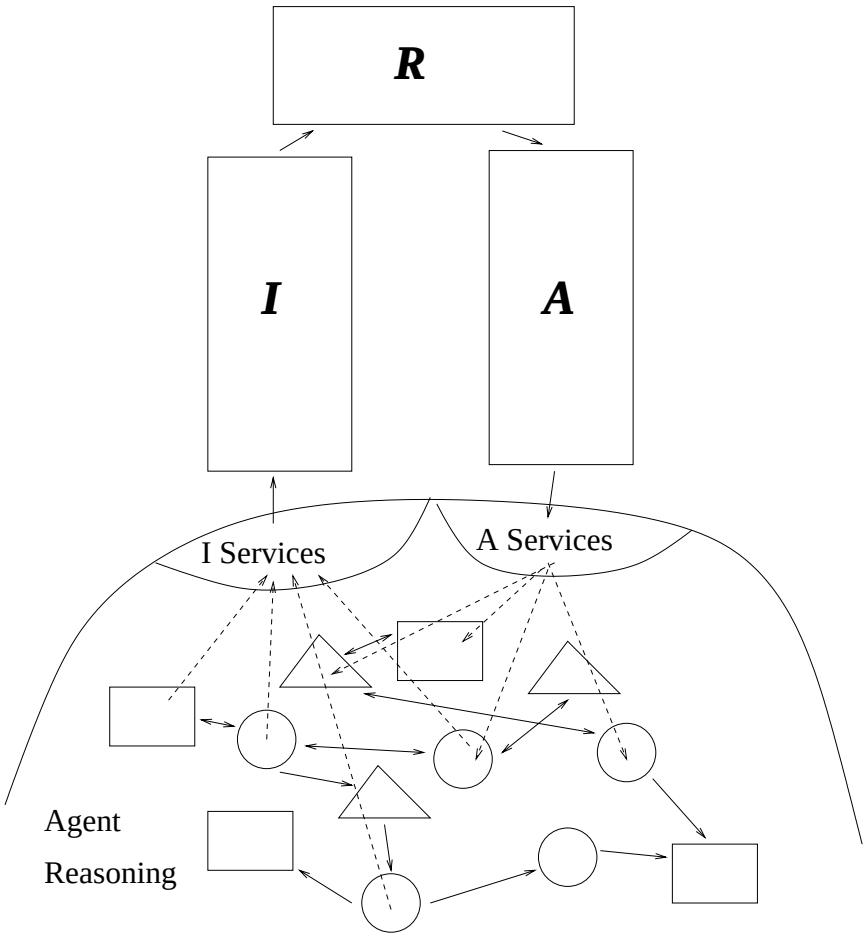

Fig. 2.4: *This figure shows how the different conceptual parts of the AA are related to the agent services for a single agent. Dotted lines show examples of how $\mathcal{I}$ services might extract information from different parts of an agent's reasoning and $\mathcal{A}$ services might change different parts of that reasoning.*

Consider an analogy between AA and a management consultant in an organisation. The consultant's job consists of collecting information, making decisions about organisational changes and implementing those changes. No matter how good the consultant is at his/her job they are reliant on employees in the organisation to inform them (either implicitly or explicitly) of the current running, goals, etc. of the organisation in order to make "good" decisions. If the employees supply limited, insufficient, incorrect or misleading information, the consultant's job is significantly harder and their results are likely to be disappointing. Once the consultant makes a decision it needs to be implemented. No matter how good a decision is, if it is not accepted and appropriately implemented by the employees, the decision is worthless. Implementation of AA works in the same way – if the entities do not supply appropriate information ($\mathcal{I}$) and properly implement decisions ($\mathcal{A}$), the best $\mathcal{R}$ is useless.

Two AA Configurations Implementations of AA appearing in the literature follow two basic configurations. The first configuration has $\mathcal{R}$ performed by a human (or humans) in the system – we call this *Directed AA* (see Figure 2.5). The second configuration has $\mathcal{R}$ implemented by software – we call this *Reasoned AA* (see Figure 2.6).

Definition 2.10: Directed AA is when $\mathcal{R}$ is performed by a human.

Definition 2.11: Reasoned AA is when $\mathcal{R}$ is done by software.

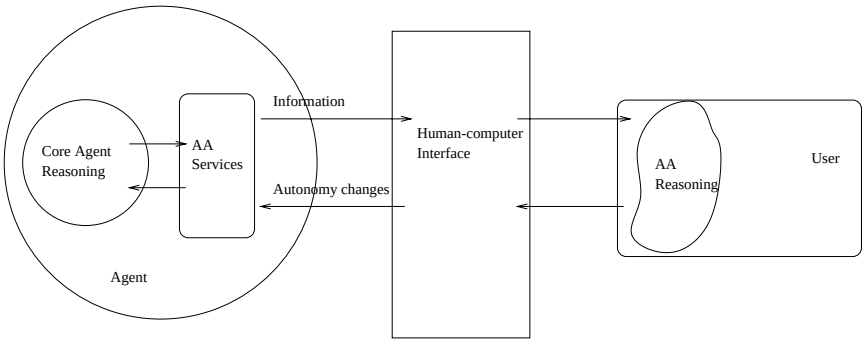

Fig. 2.5: *Directed AA involves a human user doing the reasoning about the distribution of autonomy.*

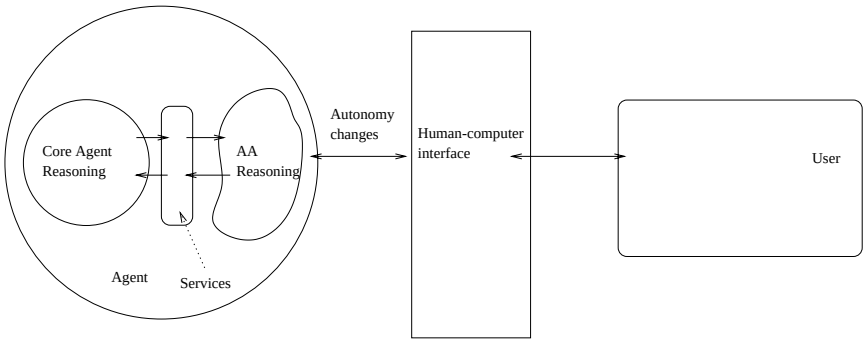

Fig. 2.6: *Reasoned AA involves the agent doing the reasoning about the distribution of autonomy.*

2.2 Motivation

AA is a technology rather than a particular capability or algorithm. The technology provides for dynamic flexibility in the relative autonomy of the entities in a system over time. As a technology, AA is not a particular solution to a particular problem, rather it is another "tool" system developers can use to improve system utility and quality. In some ways the relationship between AA and intelligent systems is the same as between Graphical User Interfaces (GUIs) and conventional software. GUIs increase the usability of many systems – even allowing some things to be built that would not be possible without a GUI. AA is similar – it increases the usability of many intelligent systems and opens up application possibilities that would not be possible otherwise. Furthermore, like for GUIs, it is desirable to have a fundamental understanding of AA that is independent of the particular application but can be used to guide the development of a range of systems (Shneiderman 1998).

The potential usefulness of AA technology has been well motivated in the literature, e.g., (Dorais et al. 1998, Musliner & Krebsbach 1999). Many authors have motivated AA by pointing out particular systems or applications that would seem to benefit from the interaction style AA offers (Zhang 1999, Miller 1999, Schreckenhost 1999). Some have explored uses of AA for improving multi-agent system performance and found it to be useful (Barber, Goel & Martin 2000). Others have reported particular applications that show effective use of AA (Bindiganavale et al. 2000, Brann et al. 1996). A recurring theme in much of the work is that the artificial intelligence tech-

nology required for a particular application is mature enough for it to be beneficial in a real world setting, however a human (or humans) is required in the loop. The reasons *why* the human is required in the loop vary. Sometimes the human is needed for mainly psychological (Rajan et al. 2000) or legal reasons, i.e., humans are unwilling to give up control for non-technical reasons. Sometimes a human is required because moral decisions need to be made. Sometimes systems need humans for their superior reasoning, sensing or manipulation abilities.

Irrespective of the reason why the human is required in the loop the humans and agents need to be able to work together effectively – AA is a technology for facilitating that effective interaction. In that role, AA can be considered as a bridging technology that allows agents (and artificial intelligence) to be used in domains where they are not currently acceptable or fully capable of acting autonomously.

Given that AA is a useful technology the key challenge becomes finding ways to effectively include it in complex systems. Any knowledge available when making design decisions in complex systems can only help to increase the chance the design process ends with an effective system. A good understanding of the implications of early design decisions inevitably leads to better systems later on. Wooldridge (2000) has looked at this problem for agents in general by analyzing exactly what it is that makes agents complex. Unfortunately, knowledge of the relationships between early system design decisions and effective AA is not readily available. Making the problem even more difficult is that AA is often patched, post hoc, onto an already very complex system. This means that the system has been designed without AA in mind. Such a process has the potential to make AA development unnecessarily difficult and/or the results unnecessarily disappointing (Dorais & Kortenkamp 2001). The insights provided by this work into the effects of agent design decisions on the AA developed later can help make the development of AA a more solid engineering practice. In particular, it allows AA to be considered from the earliest stages of system development.

2.3 Research Question

This work looks primarily at the agent services that enable effective AA to be developed for systems involving a single agent and a single human. In particular we are aiming to answer the following questions:

- How should agents be designed so that $\mathcal{I}$ can collect all the appropriate information for $\mathcal{R}$ in a straightforward manner?

- How should agents be designed so that the decisions of $\mathcal{R}$ can be effectively implemented? That is, what sorts of autonomy changes should an agent allow?
- What is the relationship between the services provided by agents and the implementation of the $\mathcal{I}$ and $\mathcal{A}$ components?

The remainder of this thesis presents our analysis of the services that need to be provided by the agent to support effective AA. A full understanding of AA also requires understanding how the information collection, reasoning and actuation should actually be implemented. In the case of Directed AA this additionally involves the design of human-computer interfaces. Though prototype versions of these systems are presented, detailed design of the interfaces is left as future work.

2.3.1 Scope of this Work

AA technology potentially applies to the interactions between any intelligent entities. However, this thesis specifically looks at AA between a single agent and a single human user. Further, we consider the user to have strictly more authority than the agent, i.e., $C_h \subseteq C_a$, the constraints on the agent are always at least as tight as those on the human. That is, there is nothing the agent may decide which the human may not. This means that the human is strictly in charge over all aspects of the system. However, this says nothing about the relative abilities of the parties. For example, the agent may be the only party capable of actually firing an aircraft's missiles but only the human can make the decision that a missile should be fired. With the current state of the art in AI and current social values a human must always be ultimately in charge, hence our assumption is reasonable (for now).

Next we assume that the agent will never intentionally disobey or reject a decision made by $\mathcal{R}$. This does not preclude the agent from questioning or asking for confirmation of a directive (e.g., a decision by $\mathcal{R}$ might prompt an agent to provide more information that it believes might lead to reconsideration of the decision) though we do not explicitly consider that possibility here. Hence, the agent is a benevolent part of the system, though certainly not completely reliable – if agents were in some way perfect AA might not be needed. For some applications or when agent technology develops further it may be useful to reassess this assumption because there are some situations where artificial intelligence is more reliable than human decision making and hence agent decisions might be "preferred" to human ones.

Even restricting our focus to AA systems with a single agent and human leaves open a wide range of research problems. Important and relevant issues range from very "soft" subjects on the human end of the interface, like *What mental models does/should a user adopt for effective AA?* to very "hard" subjects at the agent end of the interface, like *How can we maintain safety critical guarantees given AA?* We have chosen to look primarily at the way agents should be designed so that effective AA can be easily built – any complete AA theory would need to consider many other aspects as well.

It has been suggested that AA interfaces might allow a user to control an agent by manipulating its sensors, reasoning or actuators (Dorais & Kortenkamp 2001). We have chosen to look only at mechanisms which manipulate an agent's reasoning.

Although we have not focused on potential design strategies for the current human-agent interface, we have, by necessity, made some assumptions about that interface. These assumptions are explicitly stated where they are relevant throughout this work but are not necessarily uniform across application areas. For example, in the interactive simulation system (Chapter 5) we have assumed there is a high bandwidth, reliable connection between the agent and the user, so that as much data as required can be exchanged. We *do not* make such an assumption for the human collaboration system (Chapter 6).

Finally, we do not consider the cost of the computation done by $\mathcal{R}$. In some domains the cost of that computation might outweigh the benefits of using it and hence the resources allocated to AA might need to be carefully evaluated. We restrict our attention to applications where either the AA is so critical as to justify all resources allocated to it or the resource requirements are negligible in the context of the system's computational requirements.

Group Decision Making The formal framework above and this work focuses on decisions that are the exclusive responsibility of a single entity (either human or agent), although the decision making for sub-goals might be the responsibility of other entities. Some systems use group decision making where a decision is arrived at via negotiation between a group of entities (de Carvalho Gomes et al. 1998). Formalizing AA for group decision making can be more subtle, e.g., (Barber, Martin & Mckay 2000).

Group decision making units can be incorporated into the above framework by considering the group making the decision to be an entity in its own right. The group has certain abilities, responsibilities, authority constraints, etc. that can be modeled in the framework presented here. Any AA *internal*

to the group can be modeled separately using another framework.

2.4 An Ideal Adjustably Autonomous Agent

In order to make concrete the rather abstract idea of AA presented in the preceding sections, this section presents an example of AA between two humans. The example aims to highlight how AA should work, the types of functionality the agents need to have to support that AA and to highlight some of the complexity of AA so the reader can better understand the problem of building agents for AA systems. The following example illustrates just one possible AA configuration. In the example a boss, named Cameron, interacts with an intelligent butler, named Drew. We use the subscripts c and d when referring to Cameron and Drew respectively.

Together Drew and Cameron form a system that has some system goals, such as maintaining a sufficient level of caffeine in Cameron's bloodstream. Drew, the butler, has varying autonomy with respect to the sub-goals that need to be pursued. For example, Drew may be responsible for deciding when coffee should be made, what of type coffee to make or only the low levels details of making a specific type of coffee when requested, e.g., turning on and off the kettle. Assume at the beginning Drew only makes coffee when asked. We can capture Drew's initial autonomy as:

$$\begin{aligned}
&\Lambda_d = (T_d, C_d, A_d), where \\
&T_d = \emptyset, \\
&C_d = \{\text{choose_coffee, decide_when_coffee_should_be_made}, \dots\} \\
&A_d = \{\text{decide_when_to_make_coffee, choose_coffee, make_coffee}, \dots\}
\end{aligned} \tag{2.2}$$

and Cameron's autonomy as:

$$\begin{aligned}
&\Lambda_c = (T_c, C_c, A_c), where \\
&T_c = \{\text{maintain_reasonable_caffeine_level}\}, \\
&C_c = \emptyset, \\
&A_c = \{\text{decide_when_to_make_coffee, choose_coffee, make_coffee}, \dots\}
\end{aligned} \tag{2.3}$$

Drew does not have the authority to assume any of the goals in C_d, so with the above autonomy level Drew can do very little autonomously. However, at any point in time Cameron can change Drew's autonomy level. We view this conceptually as the AA system changing the autonomy level, it

just happens that in this case Cameron is doing $\mathcal{R}$. Some autonomy changes may be slight, e.g., "Use the cups on the bench when making coffee" which limits the latitude Drew has when making coffee. i.e.,

$$C_d = \{\forall g \in A_d \ \wedge \ (g \ \neg\text{involves\_cups\_on\_bench} \\ \vee \ g \ \text{does\_not\_involve\_making\_coffee}), \dots\}$$

Other changes in autonomy may be more significant and at quite a different level of abstraction, e.g., "Make me a Brazilian coffee now, please". That is,

$$T_d = \{\text{make\_coffee(Brazilian)}\}$$

Notice that Cameron's top level goals do not change because Cameron still has responsibility for maintaining reasonable caffeine levels. It is only the sub-goal of this responsibility that has been assigned to Drew.

Drew, like any intelligent human, can accept such goals at all levels of abstraction at any time – but some discretion should be used. For example, clearly, one cannot be requested to do something beyond one's abilities, i.e., Drew will not accept a goal g if $g \notin A_d$. If such a request is made Drew will intelligently, politely decline the request, possibly suggesting other courses of action. For example, a request to build a time machine might be declined. A goal may also be rejected because it is dangerous, illegal, contrary to social norms or the like. Within the framework presented here such constraints would be captured by having those goals in the set C_d (i.e., $g \in C_d$). For example, a goal to punch the cook in the nose would (hopefully) be declined even though it may be in A_d.

Rather than outright declining a request from Cameron, Drew might check that Cameron understands the implications of the requested actions. For example, checking whether Cameron realizes that it will be impossible for Drew to answer the phones while getting lunch from the sandwich shop. So, even if $g \in A_d, g \notin C_d$ (i.e., Drew can do something and is allowed to do it), but potential difficulties are detected with adding g to G_d, some dialog might be entered into.

The intricacies of the instantiation of AA in a butler do not finish with accepting or rejecting a request. Drew must also reason intelligently about how to incorporate a request with other ongoing behaviour. Say Drew was taking a phone-call when Cameron requested coffee be made. It would not be expected that Drew immediately drop the handset and put coffee on. As well as "standard" planning issues like how to achieve both tasks and

which to prioritize, Drew needs to ensure that switching between tasks is done smoothly – for example a caller should not be left hanging, without explanation while the coffee is made. The details of this re-planning and task switching should not be of concern to Cameron.

In extreme cases an intelligent human butler might find it best to consciously override requests from a boss. For example, Cameron may request not to be disturbed under any circumstances. But when the building catches fire or Australian football legend Greg Williams rings, Drew will (hopefully) ignore the earlier request and disturb Cameron – and Cameron will, no doubt, appreciate that the request was overridden. For this to be allowed with the above autonomy definitions, the goal for handling the exception must not be in C_d, i.e., Drew must have explicit authority to handle the exceptional case by overriding a previous request. If the goal was in C_d then the system would have to explicitly change Drew's autonomy by removing the goal from C_d before Drew could act. Notice that, with our formulation, the combined Cameron-Drew system always had the authority for Drew to interrupt Cameron but Cameron might choose to (temporarily) impose the authority constraint on Drew.

Sometimes the flow of initiative will not be from Cameron to Drew but the other way around. Tasking problems or information might come to the attention of the butler that would be better with the boss. For example, Drew may realize that a letter that should be in California by Friday will not arrive in time by ordinary mail. But whether the slight delay is worth the cost of a courier might be a decision which Drew cannot easily make. In such a case Drew might decide to relinquish autonomy to Cameron for that decision, i.e., decision making responsibility for the goal to get the letter sent is moved from the Drew's set G_d to the Cameron's set G_d. That is:

Originally:
$G_d = \{\text{get\_letter\_to\_California\_by\_Friday}, \dots\}$
$G_c = \{\dots\}$
After autonomy change:
$G_d = \{\dots\}$
$G_c = \{\text{get\_letter\_to\_California\_by\_Friday}, \dots\}$
After Cameron's decision:
$G_d = \{\text{send\_by\_courier}, \dots\}$
$G_c = \{\text{get\_letter\_to\_California\_by\_Friday}, \dots\}$

2.5 Summary

In this chapter we have defined AA and the underlying concepts of autonomy and level of autonomy. We defined an entity's autonomy in terms of that entity's abilities, authority and decision making responsibility. We also introduced the research problem being addressed in this thesis, which focuses on the relationships between an agent's design and implementations of AA for a system including that agent.

3. STATE OF THE ART

In this Chapter we present a review of the Adjustable Autonomy (AA) literature with the aim of putting into perspective the work in this thesis. After briefly looking at the types of applications for which AA is appropriate we look at the work that has been done in the area. The review of AA literature is divided into three sections: definitions of AA; a sample of applications using Reasoned AA; and a sample of applications using Directed AA. In particular we look at how the published work fits the conceptual model presented here and how it relates to the work in this thesis.

After reviewing AA literature we look at related fields, in particular a brief review of some relevant human factors work, a few mixed initiative systems, teleoperations research and interactive theatre applications.

The work in this thesis aims to make AA easier to build rather than showing its usefulness, thus we rely on the published literature, some of which is presented below, to motivate the utility of AA.

3.1 Applications of Agents

As underlying technologies have improved, agents have been given more control over more challenging tasks and more complex systems (Chaib-draa 1997). The advances have arguably culminated in the remote agent project where an agent had control, unassisted, of an unmanned NASA space craft for several days (Bernard et al. 1999). The remote agent project was an outstanding success but is far from the only agent success story. New, unmanned aerial vehicles, capable of performing a variety of useful missions, rely on intelligent agent software for their control (Doherty et al. 2000). Closer to earth, agents are being given more responsibility for the day to day running of human organisations (Sen et al. 1997, Tambe, Pynadath & Chauvat 2000). Electronic commerce is another area leveraging, albeit cautiously, the power of intelligent agents (Greenwald & Kephart 1999).

Not all exciting agent applications involve interacting with the real world. Another important class of applications involves intelligent agents acting in virtual environments. Military organisations are using agents to improve

their simulation capabilities, thereby improving testing, training and acquisition processes (Pew & Mavor 1998). Other groups are using intelligent agents to simulate humans providing an exciting opportunity to verify social science theories with repeatable experiments, e.g., economic theories (Luna & Stefannson 2000). Computer game makers are harnessing intelligent agents to make computer games more entertaining and challenging (Bryson 1999*a*). Agents are also featured in the movies, e.g., "Batman Returns" (Reynolds 1995). Notice that despite the huge diversity in domains, many of the same core agent technologies are applicable across a number of domains.

3.2 Motivating Adjustable Autonomy

With the exception of some early thinking about the relationships between humans and machines(Fitts 1962, Chapanis 1965), AA is a relatively new field. Much of the published literature has appeared in a series of AA specific workshops over the last few years (Kortenkamp et al. 1999, Musliner & Krebsbach 1999, Reed 2000, Hexmoor 1999*b*). Other work on AA has appeared in more mainstream areas, though sometimes referred to under other names, such as "directability" (Blumberg & Galyean 1995) and "instruction" (Bindiganavale et al. 2000). Often, the literature either describes the requirements for a particular AA application or presents the results of a particular AA implementation. Some authors have looked at formalizing AA concepts so they could be built into software for automated AA reasoning. However, only a small number of groups have tried to generalize their experiences with AA. Despite the growing number of publications in the area there is yet to be a study of the relationships between agent services and AA capabilities. This work aims to begin to fill that gap.

3.2.1 When is AA Useful?

Building AA is not a goal in itself. Rather AA is a technology that enables deployment of some types of systems or increases a system's utility or usability. It could be said that AA is to agents what GUIs are to human-computer interfaces for conventional computer systems. The advent of GUIs opened up the possibility to build a variety of useful systems that would have been either cumbersome or impossible to use with, for example, a text based interface. Just as GUIs are not appropriate for every application, neither is AA appropriate for every intelligent system. Below we extract some properties of an intelligent system that might indicate that AA will be useful.

The development of effective AA will involve considerable additional effort and cost, even if the implementation is simple (because the guidelines in Chapter 4 were followed). Hence, although AA may have some use in every intelligent system it is only reasonable to develop it if the utility of the AA outweighs the development costs. Hence, it is important to consider what AA brings to the system. For example, for some applications AA actually reduces the time and cost of agent development, because AA can "cover" weaknesses in the agents and/or help with debugging and testing. For other applications AA keeps humans in control or allows personal preferences or allows fast changes or something else. Below we consider some of the specific properties AA adds to a system, which in turn can help the decision of whether to include AA in a system.

The fundamental property that AA brings to a system is the ability to change autonomy dynamically. This immediately implies that a single entity need not have responsibility for a particular task for the lifetime of the system. This very basic property has a wide variety of uses, with different applications taking advantage of the property in different ways.

Overcoming Agent Deficiencies One of the most common uses of the idea that autonomy is not fixed is for allowing human users to assume decision making responsibilities when agent decision making is not acceptable. In particular the human's input is *required* to overcome inadequacies of an agent. The idea is for AA to support a synergistic relationship between the agent and the human by allowing each of them to operate when the task is suited to their strengths. There are a number of reasons why an agent may not exhibit acceptable behaviour. A cross section of the reasons are described in the following list:

- *Sensor, actuator deficiencies*: Sometimes parts of an agent's task may be beyond its sensing or actuation capabilities (Dorais et al. 1998).
- *Moral questions*: Decisions with a moral aspect are expected to be taken by humans (Fox & Das 2000).
- *Cost/Time*: To create an agent that can perform acceptably in *all* situations will require substantially more time and financial resources than just building an agent to cover common situations (Goldman et al. 1997, Zhang 1999).
- *Unexpected situations*: For some applications it is simply not possible to anticipate all the situations that an agent will find itself in, hence

the agent might need human input in some situations.

- *Complexity*: Some tasks may be either too difficult for an agent to handle or too difficult or vague for a designer to specify an executable solution to (Wooldridge 2000).

- *Difficult to define objectives*: In some cases the goal that the system is trying to achieve is hard to formally define. In such cases user input is needed to ensure the system meets its goals (Ferguson & Allen 1998).

Personal Preferences For some applications, e.g., personal assistant agents (Lashkari et al. 1998), it is important that the personal preferences of a user are respected. However, the agent designer might not know those preferences at design time, hence the agent must adapt at run-time. AA is a general mechanism which allows the user to impose their personal preferences on an agent by taking over parts of the decision making when agent decision making is contrary to their preferences.

Entertainment Intelligent entities are rapidly becoming important in the entertainment industry (Maes 1995). Two particular genres are based on advances in agent technology. First, a range of computer games are starting to emerge where the enjoyment of the game comes from interacting with or controlling intelligent entities (e.g., Sims (Smith 2000) and Creatures (Grand 2000)). AA would be a powerful mechanism for giving users more control over the characters, in turn leading to more enjoyment for players.

More recently (and at higher cost) mobile robots for entertainment have begun to enter the market, in particular Sony's AIBO robotic dog (Sony 2001). If experience with simulated dogs is any guide (Blumberg 1997*b*) it is likely that AA type functionality will improve a user's experience with robot dogs.

Development Cycle The original motivation for this work was to make it easier for non-expert programmers to change the behaviour of simulated pilots in a tactical air-combat simulator. We soon realized that getting a non-programmer to sit down and create a specification for a pilot was very difficult. However, if given online control, the non-programming experts might be capable of getting the simulated pilots to behave as they required, i.e., they could get the simulated pilots to do what they wanted without having to interact with the artificial intelligence specialists every time different behaviour was required.

The solution approach used was to have expert programmers use a specification system designed for them to specify "generic" behaviour for missions. The domain experts (i.e., the non-programming experts) would then tailor the behaviour to their particular goals at run-time. This creates an even more efficient development cycle than having the domain expert specify agent behaviour themselves offline. This concept has been demonstrated and some limited testing performed in co-operation with Saab, using their air-combat simulator, TACSI (Saab 1998).

This streamlined development cycle might be applicable to other domains where frequent changes are required to agent behaviour and domain experts have the required knowledge. An obvious example of this is the movie industry where expert graphics artists and artificial intelligence programmers could develop generic characters and directors could tailor their behaviour as required by the script (André et al. 1998).

Training A primary use of the TACSI air-combat simulator is for training human pilots. Sometimes the trainer will adjust the behaviour of the simulated entities in the training scenario to better tailor the scenario to the training needs (Webber & Badler 1993). AA turns out to be ideal for this because this "tailoring" requires ad hoc changes to all levels of the agent's behaviour – precisely what AA can provide.

Multi-agent Systems AA is also important when teams of independent entities work together. Different decision making frameworks will perform differently depending on the situation (Barber, Goel & Martin 2000). Dynamically reconfiguring the autonomy allows the overall system to put decision making responsibility into the hands of those that can best make the decisions.

Applications for AA

The types of applications with properties as described above span a wide variety of domains. It is illustrative to look briefly at some of the applications where AA is already being applied.

For long term manned space missions, particularly a manned mission to Mars, AA is an essential enabling technology (Dorais et al. 1998, Schooley et al. 1993). A great deal of autonomy is required on such a mission to limit the size (and hence cost) of the ground crew required to accomplish the mission (Kortenkamp et al. 2000). More importantly ground assistance may not always be possible due to blackouts, the length of time it takes

messages to get from Earth to Mars and system malfunctions. Autonomous systems would also free up the on board crew for more interesting work by taking over simple, repetitive tasks. However, in order to facilitate online repairs and maintenance as well as allowing opportunistic intervention with systems to improve performance or handle unforeseen circumstances the crew needs to be able to interact with the system at whatever level they deem appropriate.

Another area where AA technology is required is for safety critical intelligent software, such as for controlling nuclear power plants and oil refineries (Musliner & Krebsbach 1999). It is too expensive and difficult to build and properly test intelligent software for complex safety critical applications. If we want to reap the rewards of intelligent systems we need some way of making them safe. AA is one such way.

Our personal experiences have shown that completely autonomous personal assistants currently (and probably always will) make too many mistakes to be allowed to operate completely autonomously (Scerri, Pynadath & Tambe 2001). The reasons why the personal assistants make errors include not knowing the intentions of their user as well as more obvious things like not being able to adequately sense the environment. Hence, useful personal assistant agent systems require AA.

In interactive training environments AA can dramatically decrease the time and effort taken to create appropriate training scenarios (Cremer et al. 1995*a*). Likewise, interactive theatre applications can be more easily developed if the character's behaviour can be changed online. Intelligent homes, where intelligent software controls everyday devices are also likely to be good examples of AA systems (Lesser et al. 1999). In such systems, autonomy reasoning ($\mathcal{R}$) needs to be quite intelligent. For example, the "home" might check with you during the day (if you are at home) before changing the temperature but not during the night, when you are asleep.

AA has been applied to the problem of enabling a human and intelligent agent to build schedules where not all the constraints can be easily expressed in a way that the agent can understand (Galitsky 1999). In that work the level of autonomy is directly proportional to the depth of the agent's reasoning capabilities, pruning the agent's capabilities restricted its autonomy.

A group at Boeing is looking at AA to avoid the prohibitive cost of building full autonomy for manufacturing production cells (Zhang 1999). The AA will allow some of the more difficult automation tasks to be left to humans. AA is already being used to help pilots faced with difficult situations (Rock 1999). That work was inspired by a number of military helicopter accidents that occurred with helicopters flying in close formation

in darkness. The AA will be used to help the pilots maintain an acceptable distance from each other.

3.3 Adjustable Autonomy in the Literature

In this section we review the literature that specifically discusses AA. We look firstly at the types of definitions of AA that have appeared in the literature, then turn our attention to specific AA systems. The AA systems described are listed in Table 3.1.

System	Description
DAA	Multi-agent team, group decision making, naval radar application
Situated Autonomy	Introspective agents, air traffic control application
LookOut	Mixed-initiative using Bayesian reasoning, email filtering application
Robotic Wheelchair	Combines user and autonomous control signals
ALIVE	Tools for run-time manipulation of the behaviour of an intelligent character in an immersive environment
HCSM	Tools for run-time manipulation of the behaviour of entities in driving simulations
AEGIS	AA system based on a reactive planner for control of large, complex systems
JACK	Natural language commands to virtual humans
MASMA	Adjustable user preferences for autonomous behaviour, application to user schedule negotiation
3T	Interaction layer added to three layered controller for complex, autonomous systems

Tab. 3.1: *A listing of the AA systems that are discussed below with a brief description of their functionality.*

Definitions of AA

In a new field it is important that central concepts are clearly defined so that a common language exists in the community. Two basic approaches have been taken to defining AA in the literature. The first is to *formally* define AA, building on formal definitions of autonomy. The second approach is to define the properties that an AA system should have without necessarily formalizing the concept. In this section, we briefly overview some of those definitions.

Formal Definitions of AA. Formal definitions of AA lead to an ability for agents to mechanically, explicitly reason about their own autonomy. Furthermore, formal definitions of autonomy can be used by intelligent $\mathcal{R}$ software for reasoning about appropriate system autonomy configurations. In the Sensible Agents (Barber & Martin 1999*b*) framework, an agent's level of autonomy is modeled by a 4-tuple (R, A, C, I). R denotes planning *R*esponsibility, A denotes the agents and resources an agent has *A*uthority over, C denotes *C*ommitment and I denotes *I*ndependance. Hexmoor (2000*b*) defines situated autonomy using predicate calculus. Falcone & Castelfranchi (1999), Castelfranchi & Falcone (1998) build a formal model of AA based on the concept of *delegation*.

Most formal definitions of AA are tailored to the particular applications the authors are working with and may not be generally applicable. For example, the Sensible Agents definition is strongly tied to group decision making, while Hexmoor's definition is only relevant to introspective agents. A general, formal definition of AA would need to cover a much wider range of AA systems than are covered by the definitions currently appearing in the literature.

Functional Definitions of AA. The second approach to defining AA has been to describe the functionality AA brings to a system. Many authors, especially those who are building AA for existing systems, have taken this approach. In the context of long term manned space missions, Dorais, Bonasso, Kortenkamp, Pell & Schreckenghost (1998) state the following aim: "Our goal is to design a framework for human-centered autonomous systems that enables users to interact with these systems at whatever level of control is most appropriate whenever they so choose, but minimize the necessity for such interaction". Malin & Fleming (1999) offers a very similar definition. In the context of an oil refinery control system AA is referred to as "an intelligent mixed-initiative ... control system" (Musliner & Krebsbach 1999).

For a training application, Bindiganavale et al. (2000) describe AA as the ability for a human to "dynamically refine his or her avatar's behaviour in reaction to simulated stimuli without having to undertake a lengthy offline programming session.".

As with the formal definitions, the property oriented definitions of AA are tailored to the specific applications being developed. However, the essential sentiment of all the definitions is the same, i.e., AA implies flexible, online control at multiple levels of abstraction. This idea is essentially the same as our definition of AA, although we think of the agent and human more as a team than as the master-slave configuration the word "control" implies.

Summary The definitions of AA that appear in the literature capture essentially the same sentiment as is captured in Chapter 2, although our focus is intended to be broader. Generally, definitions of AA have focused on a specific application type, e.g., Barber's focus on agent group autonomy or Hexmoor's focus on introspection. The definition presented here aims to capture the underlying functionality of AA as it applies to a wide range of applications. However, because it covers more applications it is, by necessity, also weaker than many definitions in the literature.

Each of the definitions captures the meaning of AA to the systems developed by the author(s) who presented the definition. None of the definitions has been shown to be more encompassing or more accurate in defining what AA is. However, many of the definitions address similar issues and are different perspectives on the same ideas, rather than conflicting ideas. Despite the common sentiments amoung authors, the wide variety of applications to which AA is applied and differences in the specific functionality that AA brings to particular systems means that a specific, widely agreed definition of AA is still to be achieved.

3.3.1 Reasoned Autonomy

This section contains some short overviews of some of the research being done with $\mathcal{R}$ implemented in software. The survey covers a representative sample of interesting applications and approaches.

Dynamic Adaptive Autonomy

The University of Texas' Dynamic Adaptive Autonomy (DAA) framework allows a group of agents to allocate autonomy amongst themselves in a way

that results in the best group decision making performance (Barber, Martin & Mckay 2000). In this system, $\mathcal{R}$ is performed by a *group* of agents. DAA uses autonomy as an important part of the group decision making process. Experiments using a simulated naval radar frequency sharing problem, show that different decision making arrangements lead to different performance levels depending on the particular task (Barber, Goel & Martin 2000). This is an important result because it clearly shows AA can improve system performance (rather than "only" avoiding agent limitations) by showing no one particular autonomy configuration is right for all situations.

In DAA, autonomy is considered with respect to each goal individually and is described by three attributes, G, D and C where G is the intended goal of the decision making group and C is the set of agents over which the decision making group D has authority. In the decision making framework, D, each agent gets a certain amount of say, represented by the number of votes it has in the decisions of the group. In the decision making framework D, for the goal G, the number of votes an agent has relative to the total number of votes all the agents in the decision making group have gives a numerical value for the agent's autonomy.

Using the formal definitions of autonomy, formal descriptions of different *levels of autonomy* can be accurately described. For example, a *command driven* agent has no votes in the decision making framework which has authority over it and *consensus decision making* is when all agents in the set C are in the decision making framework, D, and have the same number of votes.

One focus of this work has been the protocol by which the agents work out what decision making framework, e.g., consensus, master-slave, locally autonomous, they should adopt given a particular situation (Barber, Martin & Mckay 2000). This is a difficult problem given that each agent has only local knowledge and hence cannot know, for sure, what formation is best for the group. Currently, the system uses a pre-determined mapping from situations to decision making frameworks to determine which autonomy configuration to adopt. The number of votes that an agent has in a decision is the feature of the system which the autonomy actuation component ($\mathcal{A}$) can affect to change the behaviour of the system. At this stage of development the AA information gathering component ($\mathcal{I}$) uses a priori simulations of how different autonomy configurations perform in different circumstances.

Situated Autonomy

Hexmoor (2000*b*) defines *Situated Autonomy* as an agent's *stance* towards a

goal at a particular point in time. That stance is used to guide the agent's actions. A key in this work is that the agent uses *introspection* to determine the current level of autonomy rather than autonomy being an external property of the agent. The agent's own assessment of its own autonomy is used in the decision making process. One of the implications of this view of autonomy is that autonomy and dependence are orthogonal rather than intertwined.

One focus of this work is on *how* an understanding of autonomy affects the agent's decision making at different decision making "frequencies", e.g., reflex actions and careful deliberation (Hexmoor 1999*a*). For example, for reactive actions only the agent's *pre-disposition* for autonomy towards the goal is taken, while for decisions with more time available a detailed assessment is done to optimize the autonomy stance.

Some of the ideas have been applied to an air traffic control problem where agents helping a human air-traffic controller use situated autonomy to guide their decisions for interacting with the air-traffic controller (Hexmoor 2000*a*). In that particular application the level of agent autonomy is a (fixed) function of the priority of the importance of the current task (as assessed by the agent).

$\mathcal{I}$ sends information directly from the agent's perception systems at lower decision making frequencies and uses pre-calculated information at higher frequencies (Hexmoor 2000*b*). $\mathcal{A}$ changes the behaviour of the agent by making the autonomy level an input into the decision making process of action selection for a goal (Hexmoor 2000*a*).

Later work has looked at how autonomy can be measured by comparing an agent's performance with and without its context (Brainov & Hexmoor 2001). The idea being that its autonomy level is related to how much an agent's context adds to its performance.

Focus of Attention

Horvitz, Jacobs & Hovel (1999) have looked at AA-like ideas to reduce annoying interruptions caused by alerts from the variety of programs that might be running on a PC, e.g., notification of new mail, tips on better program usage, print jobs being finished and so on. They are emphasizing the user's *focus of attention* and using that as a primary factor guiding AA decisions. Bayesian theory is used to decide, given a probability distribution over the user's possible focii of attention and potential costs and benefits of action, whether the agent should take some action autonomously. The agent has the further possibility of asking the user for information in order

to reduce its decision making uncertainty. In this case then, $\mathcal{I}$ is presenting information both from the user and from the agent to $\mathcal{R}$. A prototype system, called LookOut, for helping users manage their calendars has been implemented to test the ideas (Horvitz 1999*a*). Future efforts will look at more complex situations with hierarchical arrangements of goals (Horvitz 1999*b*).

The careful determination of the user's focus of attention is an interesting complimentary capability for AA because it provides input to $\mathcal{R}$ from the *user's* perspective without unduly disturbing them. Other AA approaches rely on having only the *agent*'s perspective of a situation, i.e., $\mathcal{R}$, implemented in software, relies on information about the state of agents and, perhaps, user preferences, but does not usually consider detailed information about a user's current activities.

Assistive Technologies

One area where artificial intelligence is having a significant, direct impact on the quality of life is in applications where technology is used to assist people without abilities common to most of the population (Miller 1998). One particular application is where robotics and artificial intelligence are helping mobility impaired people decrease their dependence on human help. Systems such as NavChair incorporate advances in mobile robotics into powered wheelchairs to empower otherwise unable people to move around unstructured environments (Simpson et al. 1998).

Traditional powered wheelchairs have a joystick by which the user controls the direction of the chair. However, a joystick is beyond the capabilities of some users who must rely on head or even eye movements to control the chair. Artificial intelligence techniques that help the user avoid obstacles and otherwise generally assist in the navigation of the chair are extremely welcome. However, such approaches introduce the need for conflict resolution between the automatic system and the user. Current solutions utilize mode switching where the user or system switches between different autonomy modes depending on the current tasks (Miller 1999, Simpson et al. 1998).

An interesting aspect of this application of AA is that sometimes the software will override the user. This is not because the software does not "like" the decisions of the human but because it must consider the possibility that the human was not able to express their decision quickly or accurately enough. This is different from other AA applications where the artificial intelligence state of the art and social standards dictate that humans always

have the final responsibility.

For this type of system, $\mathcal{R}$ is particularly concerned with how the user and human commands are combined. In particular, $\mathcal{R}$ can choose between different *modes* of operation. For example, in *wall following mode* the user's joystick commands are modified to best follow nearby walls (avoiding obstacles etc.) The modes are chosen depending on the environment, i.e., $\mathcal{I}$ supplies environmental information. $\mathcal{A}$ makes changes by altering the algorithm which combines user and agent commands. Experiments have shown the need for more advanced, intelligent interaction between the parties. In this context mode switching can be thought of as a discretized version of AA, i.e., only certain autonomy configurations are allowed instead of the complete spectrum.

3.3.2 Directed AA

The second group of AA applications we look at are those where $\mathcal{R}$ is done by a human. The focus of the development efforts here is not the autonomy reasoning, rather it is focused on providing an interface that users can use for $\mathcal{R}$. That interface must provide both the information required for effective decision making and controls for specifying autonomy changes.

ALIVE

Blumberg & Galyean (1995) were amoung the first authors to note the need for some kind of external control of autonomous characters for a variety of applications. They refer to the ability of an outside party to control an otherwise autonomous creature as *directability*. This work focused on autonomous creatures, the most well known of which is Silas the dog, that could interact with humans in an immersive environment called ALIVE. The directability functionality was used to allow yet other humans to change the behaviour of the autonomous creature to improve the interaction experience of the human immersed in the environment. It was observed that such interaction was needed at all levels of abstraction and at all times (Blumberg 1997*b*, pg. 32).

ALIVE characters, like Silas, use a hierarchical, behaviour based architecture, inspired by ideas from ethology, for action selection. The architecture has many similarities to the EASE architecture described in Chapter 5. A key difference between ALIVE and EASE characters, is that in ALIVE at each level of the hierarchy only one behaviour is *active*, i.e., allowed to act, although others may provide suggestions, while in EASE each "behaviour"

has an influence on the character's actions.

The external control of an ALIVE character is implemented by giving the user access to the internal data structures of the agent, in particular to the factors which influence the relative strength of behaviours or groups of behaviours in the decision making process. Four types of controls, constituting $\mathcal{A}$, are provided. First, the user can change the character's motivation for engaging in a particular behaviour. Second, the user can change the objects that are the focus of a behaviour, which in turn changes the probability they will become active. For example, the *marking* behaviour of a simulated dog can be changed to focus on *user's pants leg* instead of *fire hydrant*. Third, the level of interest a character has in a behaviour can be changed. Finally, the user has the ability to simply make particular behaviours active at each level of the hierarchy. A feature of the approach to directability was that it was not necessary to change the character architecture at all to introduce the directability (Blumberg 1997*b*, pg. 86).

Although Blumberg's approach to directability, or AA, is more powerful and flexible than many other approaches, it is still limited. The user has a range of control over the *existing* behaviour hierarchy but cannot change the hierarchy by adding or removing behaviours. Furthermore, persistent directions, e.g., *never do X*, are difficult to express. Another unsatisfactory aspect of the approach is the way transitions between behaviours are managed. When a behaviour completes, it returns the character to some neutral state from which the next behaviour knows what to do ((Blumberg 1997*b*), pg. 65). If human-like behaviour is required, the technique of returning to a neutral state might be unacceptable as it is not natural. When an autonomy change is made, ad hoc transitions between behaviours will occur. Returning the character to a neutral state when making such a transition may be unacceptable.

Hierarchical Communicating State Machines

Hierarchical Communicating State Machines (HCSMs) are the behaviour control mechanism for actors in a driving simulation developed by the University of Iowa (Cremer et al. 1995*a*). The simulation is used to train drivers and perform experiments on the effects of different factors on driving ability. To be most effective as a training and experimentation tool, specific driving situations need to arise in what otherwise appears to be normal traffic.

HCSMs are concurrent, hierarchical state machines that control the behaviour of the entities in the traffic scenario. Everything in a simulation from cars to traffic lights are controlled by a HCSM. Each HCSM has a *control*

panel interface that allows other HCSMs or humans to alter its behaviour. When the HCSM is developed the specific controls on the control panel are designed and their effects on the state machine "hard-coded". Thus, controls for changing complex, abstract aspects of behaviour like a driver's *aggressiveness* or *reaction time* can be straightforwardly implemented by creating a control that changes a state-machine in whatever way is required to achieve that effect. Users "push buttons" and "turn dials" on the HCSM control panels to change the behaviour of entities and, hence, control a scenario. At run-time the changes that can be made to a HCSM's behaviour are limited to those that have been explicitly designed into the control panel. Special HCSMs called *directors* are embedded in the environment to create experimental driving scenarios by "pushing buttons" and/or "turning dials" on other HCSMs at appropriate times (Alloyer et al. 1997). A graphical user interface allows a human user to view the current state of the HCSM for debugging purposes (Cremer et al. 1995*b*).

AEGIS

Musliner & Krebsbach (1999) at Honeywell are looking at AA as a means of reducing some of enormous cost of "abnormal situations" in the petro-chemical industry. The work has resulted in a system called AEGIS (Abnormal Event Guidance and Information System) that combines human and agent capabilities for rapid reaction to emergencies in a petro-chemical refining plant. AEGIS features a *shared task representation* that both the users and the intelligent system can work with (Goldman et al. 1997). A key hypothesis of the work is that the model needs to have multiple levels of abstraction so the user can interact at the level they see fit.

Both the user and the system can manipulate the shared task model. The model, in turn, dictates the behaviour of the intelligent system. Interestingly, the authors note that given the AA requirements "first principle" planners did not provide a suitable basis for the shared task representation because they require too much detail and do not provide abstraction mechanisms. This observation is in keeping with one of the underlying assumptions of this thesis, i.e., that thought needs to be given to the design of agents in order to effectively support AA.

The important aspect of AEGIS for the purposes of AA is the goal-setting, planning and execution core, called GPE. GPE uses PRS (Ingrand et al. 1992, Georgeff & Lansky 1987) reactive plans for its reasoning. Goals are explicitly represented and hierarchically arranged. Different parts of a goal hierarchy are the *responsibility* of either the system or human user.

Certain goals can only be performed by the system or by the human. There is also an explicit representation of authorization for taking on a particular goal. PRS incrementally develops plans so showing the user implications of their actions or the intentions of the system is difficult, however extensions are being developed to deal with this. Interfaces will be built on top of PRS that allow users to view and manipulate the system plans. One of the key features of this approach to implementing AA is that once appropriate user interfaces to PRS are built a variety of different systems can be controlled without re-implementing the AA components.

JACK

The University of Pennsylvania is working to build *virtual humans* for a variety of domains. The result of the work is a system called JACK. A major focus of their work is to create good animations of human behaviour (Badler 1997). The control architecture of the virtual humans is layered, lower levels look after motion control and animation and PaT-Nets (Parallel Transition Networks) dictate the high-level behaviour of the system.

A key to the user control of the virtual humans are *PARs* (Parameterized Action Representations) that dictate the virtual human's behaviour. As their name suggests, these PARs abstractly describe the actions that the virtual human can perform. Uninstantiated PARs, i.e., UPARs, a sort of generic action, are stored in an "Actionary" and do not affect the character's current behaviour. IPARs are UPARs with parameters instantiated for a particular situation. The IPARs are linked to the PaT-Nets to control the character's behaviour.

A natural language interface allows a (real human) user to interactively command the virtual human(s) (Bindiganavale et al. 2000). The natural language processor take a natural language statement and works out which UPARs should be instantiated, i.e., made into IPARs, to implement the change in behaviour the user desires. Both "standing orders" and immediate actions can be requested via the interface. The natural language processor can also understand qualifiers like "if" and "for all" and create IPARs appropriately.

The natural language approach to getting human input is a very effective way of allowing inexperienced users to control the virtual humans (an approach also taken by some groups to the control of physical robots, e.g., (Perzanowski et al. 1999)). However, the control the user has is limited to the UPARs in the Actionary and the ability of the natural language processor to translate each request into appropriate IPARs. The user relies on the

simulation visualisation to understand the character's behaviour, i.e., there is no $\mathcal{I}$.

MASMA

Cesta, D'Aloisi & Collia (1999) have looked at AA as a way of boosting *trust* between an agent and its human user. The idea is that giving the user more control over their personal assistant agent will allow them to slowly build up trust in the agent, slowly increasing the agent's autonomy as their trust increases. In the system, called MASMA (Multi Agent System for Meeting Automation), autonomy is equated with *initiative*. Each user in a research organisation has a representative agent that negotiates meeting times on their behalf. The user can influence the protocol that the agent uses in the negotiation and the information that their agent makes available to the other agents. The agents have a variety of different generic negotiation mechanisms from which to choose. Mechanisms are chosen to best suit the current user preferences. Over time the user is expected to change a file containing their preferences, increasing the agent's autonomy to negotiate on their behalf. The user also has access to a visualisation of the negotiation which they can use to influence the running of the negotiation directly.

3T

An AA interface to the 3T architecture (Bonasso et al. 1997) has been implemented to solve human-machine interaction problems experienced using the architecture in a number of NASA projects (Brann et al. 1996). The experiences showed that interaction with the system was required all the way from the deliberative layer through to detailed control of actuators. A principle that came out of that work was that the machine should be designed for complete autonomy then the autonomy restrictions relaxed at each level starting at the top (Bonasso 1999). However, we contend that AA should at least be considered from the earliest stages of agent development, even if it is not implemented immediately, to ensure that agents have the features required to make AA implementation straightforward.

At the deliberative, planning level of 3T the user can influence the developed plan while the system ensures that hard constraints are not violated. At the middle level, i.e., the conditional sequencing layer, either the human user or system (usually a robot) can be responsible for the execution of each task. Each task has a pre-defined autonomy level that dictates whether the system should check with the user before starting on the action or just go

ahead and act. At the reactive level the NASA developers were surprised by the need for human intervention but did find it necessary, mainly for testing purposes (Bonasso 1999). The AA at the reactive level is implemented by a *tele-operation* skill that lets the user take over low level control, overriding commands of the system. The AA controls at all layers are encapsulated in what is referred to as the 3T's fourth layer – the interaction layer (Schreckenhost 1999).

An interesting new problem to come out of the addition of AA to 3T for mission critical systems is the need to validate the resulting human-robot system (Malin & Fleming 1999). The complexity and unpredictability that the human brings to the system makes the job of ensuring the intelligent software always works as required significantly more difficult and raises a variety of new problems. It is an interesting paradox that a technology brought in to help an agent handle abnormal situations that were not thoroughly tested increased the difficulty of testing behaviour in "normal" situations.

Summary

Tables 3.2, 3.3 and 3.4 summarise a selection of agent features for the different systems surveyed and the AA facilities those features lead to. The tables show that the diverse agent architectures offer diverse services to AA implementations. Clearly, some of the resulting AA facilities are more powerful or useful than others. Notice that nearly all the architectures, though very diverse, provide substantial $\mathcal{I}$ and $\mathcal{A}$ without any major development effort. Some systems do not have any support for $\mathcal{I}$, meaning that the user or software performing $\mathcal{R}$ has to watch the ongoing situation and is not privy to the internal workings of the agents. The $\mathcal{A}$ features are generally very dependent on an existing underlying architecture hence are limited by the restrictions of that architecture. The systems presented in this thesis, on the other hand, provide extensive information for $\mathcal{I}$ and a range of controls for $\mathcal{A}$.

System	Agent Feature	$\mathcal{I}$ Feature
DAA		Pre-computed configurations
Situated Autonomy		Pre-computed information and perception
LookOut	Can calculate the user's focus of attention Natural language understanding	Focus of attention Importance of email messages
Robotic Wheelchair	Ability to sense its physical environment	Evaluation of environment type
ALIVE	Behaviour based hierarchy	Goal-hierarchy of actor
HCSM	Explicit representation of state machines	State machine visualization
AEGIS	Partial plans Goal hierarchies	Explicit authorization of plan steps, shared plans Abstraction
JACK		User relies on observed agent behaviour
MASMA	Explicit Negotiation	Visualization
3T	Layering	Abstraction

Tab. 3.2: *A selection of $\mathcal{I}$ features and the underlying agent features that make them possible.*

System	Agent Feature	$\mathcal{A}$ Feature
DAA	Voting mechanism for group decision making	Number of votes in decision making
Situated Autonomy	Action selection algorithm considers autonomy level	Direct input into action selection
LookOut	Will not execute without "permission" from AA Dialog with user	Allows, delays or disallows an action Can get additional information from user to reduce uncertainty
Robotic Wheelchair	Different modes for combining software and user input	Chooses between different modes
ALIVE	Explicit factors influencing action selection	Change factors affecting action selection
HCSM	State machine controls	Control panels
AEGIS	Shared task model	User can perform some tasks Mixed-initiative planning
JACK	UPARs	Natural language interpretation instantiates UPARs
MASMA	Different protocols	Preferences file
3T	RAPs, planner	Explicit representation of tasks and plans

Tab. 3.3: *A selection of $\mathcal{A}$ features and the underlying agent features that make them possible.*

System	$\mathcal{R}$ highlights
DAA	Determined by group with established protocol
Situated Autonomy	Determined by available time
LookOut	Bayesian reasoning
Robotic Wheelchair	Mode switched
ALIVE	By human
HCSM	Both by human and by director agents
AEGIS	By humans
JACK	By human
MASMA	By human
3T	By human

Tab. 3.4: *Summary of $\mathcal{R}$ for the surveyed systems.*

3.4 Related Ideas

So far we have looked at work specifically focusing on AA. However, there are other research fields that are closely related to AA which we review here. We believe many of the ideas from these fields might also be relevant to AA because the basic goal of having systems and humans work together is common to all the systems. In particular we look at mixed initiative systems, teleoperated systems, interactive theatre applications and human factors work.

In a mixed initiative system all decision making control is transferred at once, compared to AA where perhaps only some decision making control is transferred. Hence, in mixed initiative systems $G_a = \emptyset \vee G_u = \emptyset$. In a teleoperated system the human user has all the high level decision making control, hence $top\_level\_goals(G_u) = system\_goals$. For mode switching systems like some auto-pilots, Λ_a is constrained to take on a fixed, usually small set of discrete values (each value corresponds to a single mode) hence there is considerably less flexibility than for AA.

3.4.1 Mixed Initiative Systems

Mixed initiative (MI) systems allow control to be traded between the system and a user (Horvitz 1999*a*). Fleming and Cohen distinguish between two types of MI systems, *strong MI* is when users can interrupt the system freely and *weak MI* where the timing of user interaction with the system is restricted (Fleming & Cohen 1999). Strong MI systems have much in common with AA, though are still less general because the possible autonomy configurations are more restricted, i.e., $G_a = \emptyset \vee G_u = \emptyset$. Much work has been done with mixed initiative systems, in the following we review just a small selection of interesting MI systems.

Mixed Initiative Problem Solving

A common use of MI techniques is to combine the skills of a human and intelligent system to solve some (complex) problem (Ferguson et al. 1996). The system takes responsibility for detailed, repetitive or otherwise machine suited task aspects while the human looks after aspects requiring creativity, hard to define objectives or other aspects better suited to human abilities. Mixed initiative *planning* is a special case of mixed-initiative problem solving, where a human and a software planner collaborate to produce a (complex) plan (Veloso et al. 1997, Tate 1997).

TRIPS (Ferguson & Allen 1998) is a good example of a sophisticated MI problem solving assistant. TRIPS collaborates with a user to create complex logistical plans for an imaginary hurricane evacuation scenario. The authors note that substantial changes needed to be made to the underlying, existing planning system to support the advanced collaboration features they introduced. This supports this thesis' contention that careful consideration needs to be given to agent design so as not to hinder the development of AA. In particular, a Problem Solving Manager was introduced to TRIPS which manages additional planning information (i.e., information not strictly required by the software planner).

Dialog

Dialog based systems are a common type of MI system (Haller 1997). During a dialog, complete control over the conversation shifts from speaker to speaker – something that is inherently supported by MI systems. That is, MI systems inherently support shifting complete control between parties, just as occurs in a dialog. Donaldson & Cohen (1997) have developed constraint satisfaction algorithms which allow intelligent agents to intelligently share speaking initiative with a human counterpart. The techniques used there might be applicable to a wider range of AA problems.

3.4.2 Tele-operation

A tele-operated system is one where a human makes all the high level decisions, perhaps based on sensory information received from the system, and sends low-level commands to the system, usually a robot, to be executed(Sheridan 1992). The simplest tele-operated system has no onboard intelligence and simply executes the commands of its controller (Gilbreath et al. 2000). Tele-operated systems are an extension of shared control and traded control systems (Lee 1995). Due to a range of difficulties with "pure" tele-operation, such as time-delays, lack of operator situational awareness and high operator workload (T. Fong & Baur 2000), tele-operation researchers have started including more intelligence in the tele-operated system, relieving some of the responsibility from the user. As more intelligence is built into the system tele-operation systems start to more closely resemble AA systems. Hence, it is useful to look at the work done in the field as the techniques may be applicable to AA, especially Directed AA.

Tele-operation is used for robotic applications where the robot's task is poorly defined, unpredictable or complex and it is infeasible to build all the

required intelligence into the system. For example, terrain exploration tasks are difficult to achieve autonomously because it is difficult to know a priori what interesting features of the terrain deserve closer investigation (Simmons et al. 1994). Another interesting example of a tele-operated system is a command and control architecture for processing plants located in space (Schooley et al. 1993). High level commands are sent to the plant where they are translated into appropriate low-level commands and in turn into simple procedures. Telemetry data sent from the plant is used by the Earth based human operators in their decision making.

Simmons, Krotkov, Hebert & Katragadda (1994) use an arbitrator that takes both high level navigational input from a human user and low level obstacle avoidance input from sensors to produce safe paths for a mobile robot system for terrain exploration. The system allows the user to override the robot's safety advice in exceptional circumstances or relinquish complete control to explore benign terrain, however the system is normally operated in the intermediate mode they refer to as "safe-guarded tele-operation".

T. Fong & Baur (2000) have focused on providing better human-machine interfaces to tele-operated mobile robots. Their work looks at improving both the sensing interface (i.e., $\mathcal{I}$) – providing the human operator with a better understanding of the robot's environment and state and the control interface (i.e., $\mathcal{A}$) – allowing more abstract commands to be sent and executing the commands more intelligently. Rather than thinking of the robot as a slave to the human controller they treat the human and robot as peers, with the humans being taken as an "inprecise, limited source of planning and information" (Fong et al. 1999). Such an approach is similar to Directed AA and the interface technique may be extended to an interesting approach to AA. Lee also takes the view that the robot and user should work together as partners (Lee 1995).

(Sawaragi & Horiguchi 2000) takes quite a different approach to the human-machine interface problem by using a virtual reality (VR) environment as an interface between the parties. The VR environment simplifies the human-machine interface by giving the human operator "embodied cognition" and allowing them to better exploit physical manipulation capabilities when controlling the machine. This type of interaction could be extended as far as directly coupling a human brain to the robot as has been already done with monkeys and robots (Wessberg et al. 2000).

Gilbreath, Ciccimaro & Everett (2000) have addressed some of the limitations of tele-operated systems by increasing the abstraction level of the commands that a user can send to a mobile robot for non-lethal tactical response. Rather than only "joy-stick" type controls for controlling robot

speed and direction, camera direction and non-lethal weapon direction, the user can send higher level commands for, for example, following a wall or tracking a target. The more abstract commands, combined with an advanced interface, reduce the difficulty of the robot operator's task. As the ability to send commands at all levels increases the system becomes more like an AA system. If the user can prescribe changes to many aspects of behaviour at all levels of abstraction then the mechanism could be described as AA.

3.4.3 Interactive Theatre

Interactive theatre is a domain where Directed AA type ideas have been extensively used. In interactive theatre applications simulated, animated characters generally perform a "script" written in more or less formalized natural language. The characters combine built-in personality traits and built-in objectives with the high level instructions in the script to tell interesting stories (Bates 1993). One system has a character performing a multimedia presentation, e.g., a teaching presentation, from a script (André et al. 1998).

The capability of a character to take a script and interpret it at run-time is almost equivalent to it being able to accept arbitrary end-user commands and interpret the commands on the fly. So, if the actor can read a script and accept the commands of the script, incorporating them with its "other" behaviour then it should also be able to accept commands from an end-user and incorporate them with its normal behaviour. In fact, quite often, interactive theatre applications do have provisions for run-time user input, though these are not as extensively used as the scripting capabilities. Generally only $\mathcal{A}$ functionality and not $\mathcal{I}$ functionality is provided (primarily because there is no purpose providing information to a static script).

Hayes-Roth, Brownston & van Gent (1997) have implemented a *Directed Improvisation* system in which characters can interpret end-user directions, while taking into account their particular distinctive styles, adhering to social conventions and meeting other objectives. *Abstract control plans* determine a character's behaviour. Each step of the plans can be instantiated in different ways, depending on the character's personality. Users can specify the control plans interactively at run-time.

Wavish & Connah (1997) translate formalized English scripts into a format that their characters can understand and perform at run-time. Each character is made up of a society of simple, interacting agents, called CDAs – Communicating Deitic Agents. Each CDA is responsible for some aspect

of the character's overall behaviour. A formal script is translated into CDAs via a sort of "compilation" process. The idea of CDAs is similar to the ideas in Chapter 5 though there the intermediate compilation step is not required.

In Improv (Perlin & Goldberg 1996), scripts are again used as the mechanism for "discrete control of the decisions made by the actor's mind". The characters use a behaviour based decision making architecture. The scripts use a variety of specification techniques including parallelism, abstraction and non-determinism, to allow a script-writer to specify a wide range of behaviour. Online, end-user control can be added by including specifications of *user interface elements* in the scripts. At run-time, the interface elements can allow the user to control the characters at any level of abstraction, however the user only has controls that were included in the scripts, hence the handling of unexpected situations is weak.

3.4.4 Human Factors

Whenever humans and intelligent software work together human factors questions arise (Flanagan & Huang 1997). One interesting strand of human factors work has looked at how the total functionality of a system should be divided so that the performance of the human-software system is optimized. *Balanced work* is a notion of balancing technology, humans, and organisation (see Figure 3.1). Furthermore, "balanced work is the result of an adjustment by the working system to the performance demands" (Bye et al. 1999). It is argued that not only should designers look at relative competence and/or technical limitations but that system functionality should be divided so that humans stay in control and maintain their skills (Grote et al. 1995). This consideration is not usually taken into account for AA, where competence is usually the sole rationale in decision making. Bye, Hollnagel & Brendeford (1999) contend that a (relatively small) change in function allocation disturbs the system "equilibrium" potentially causing changes to the equilibrium of the total work situation. The ideas are being applied to a study of automation at a nuclear power plant. The effect on human users is an important issue if an AA system is to continually alter the autonomy distribution of a complex system as $\mathcal{R}$ may have to take into account the effects on humans.

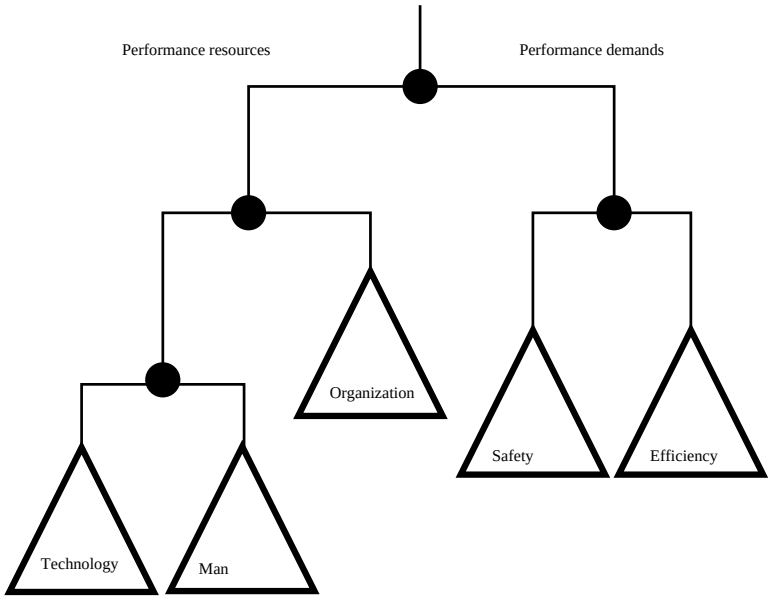

Fig. 3.1: *Diagram relating the different factors affecting function allocation. Balanced work keeps the different components in balance (Bye et al. 1999).*

3.5 Summary

In this chapter we have presented a selection of relevant AA work, as well as some work from related areas. Many of the applications and ideas are exciting and open up new possibilities for interacting with intelligent systems. However, in most cases the AA has been added to an existing system and is thus restricted by the design of the original system.

Despite the number of applications appearing, very few, if any, general principles for developing AA systems have emerged, particularly regarding the way agents should be designed. Generally, developers have added AA controls that are specifically suited to the features of their existing system. The lack of general principles or guidelines for AA systems means that developers have no principled way of going about building AA in a way in which they can be confident of a successful outcome. Thus, it is not surprising that several authors noted that significant changes were required to underlying systems to allow AA to be implemented. This work aims to provide some background knowledge that will help future developers make informed agent design decisions for AA systems.

4. GUIDELINES FOR BUILDING AGENTS FOR ADJUSTABLE AUTONOMY SYSTEMS

In this chapter general guidelines for designing agents to be used in Adjustable Autonomy (AA) systems are described. The guidelines given here should be used during the design process when one of the requirements on the development is that the agents will be used in AA systems. It is our contention that if agents are designed and built according to these guidelines AA will be easier and faster to implement.

4.1 Using Guidelines for Capturing Design Experience

Design patterns are one approach that has emerged in the software engineering community to communicate design experience to others (Gamma et al. 1995). While design patterns for AA would be a desirable way of communicating AA design experience, the breadth of applications for AA is simply too broad for patterns to be useful, although specific patterns for specific AA applications might emerge as the field matures. Instead, we aim to capture AA design expertise via guidelines. The guidelines capture our experiences designing agents for AA systems in a way that allows other designers to leverage that knowledge.

The type of information that we are trying to capture cannot be captured by metrics or other "objective" measures. The type of information we are trying to capture is abstract in nature and is more akin to "rules of thumb" than mathematical proofs or equations. Other authors have also taken the approach of using guidelines to capture design knowledge. Fleming & Cohen (1999) have chosen to use guidelines to capture appropriate practices for building mixed-initiative systems. Shneiderman (1998) provides guidelines for building graphical user interfaces and Joseph (1986) proposed and evaluated guidelines for ski lift design. Often standards documents also provide guidelines for the design and development of some type of system, e.g., (IEEE 1998). Hence, there is ample precedent for this approach.

4.1.1 Using the Guidelines

When designing intelligent agents many competing requirements need to be reconciled. Each requirement poses some constraints on the agent's design. Not all the requirements are related solely to the observable behaviour of the agent. For example, there may be particular verifiability, simplicity or computational requirements on an agent (Scerri & Reed 2000*c*).

Designers, implicitly or explicitly, follow guidelines when attempting to meet some requirement with a design. A guideline is "a statement or other indication of policy or procedure by which to determine a course of action" (Heritage 1996). For example, a (simple) guideline to minimize computational requirements for an agent might suggest avoiding algorithms involving significant amounts of search. By following appropriate guidelines designers have a principled, justifiable reason for believing that their design will meet its requirements. For example, if a design avoids extensive use of search algorithms a designer can argue, with justification (by referring to the above guideline), that once implemented the design will be computationally efficient.

Many of the guidelines a designer follows come from their own experience with previous systems. However, learning to design systems through (good and bad) experiences with previous systems is unlikely to be an effective method for creating complex software, simply because so much experience is required to understand the many implications of a design decision on the resulting complex system. The need for an unreasonable amount of experience is likely to occur for AA designers as the AA systems are very complex and the consequences of agent design decisions on the ease with which AA is developed are not seen until late in development (perhaps even by a different team of designers).

Clearly, when designing a particular agent there are many things to take into account other than making AA easy to implement. Most obviously, the agent must be able to achieve the behaviour required of it. Other design requirements like the ability to formally verify the agent's behaviour will also be made easier or more difficult by the design of the agent. Guidelines, either explicit or implicit, will advise ways to meet other requirements in a straightforward manner. In the usual case it will not be possible to design an agent which makes everything straightforward, i.e., it will not be possible to follow all the guidelines because of other conflicting requirements.

The design of any complex item is a challenging task, requiring knowledge, experience, creativity, etc. When designing a complex system it is important to understand how particular design decisions constrain other

design decisions and the functionality of the final system. Some relationships are clear, e.g. using a heavier component on a spacecraft will mean more thrust is required to get it off the ground. Other relationships are less clear, e.g. when an architect starts designing a building the impact on the layout of the tenth floor from the positioning of lifts may not be clear. The tradeoffs that need to be made, e.g. the extra thrust required for the heavier component on the spacecraft versus the extra functionality the heavier component provides, are often difficult to resolve in a good way. However, if the relationships between a design decision and its effects on other aspects of the design are *understood* at least resolving the tradeoffs can be approached in a principled, informed manner.

The guidelines here serve as a mechanism for making explicit the relationships between design decisions and implications with respect to a single system requirement, namely AA. With the guidelines a designer can attempt to make decisions that lead to agents with features most amenable to implementations of AA. When the designer makes a design decision that violates the guidelines, at least the decision can be an informed one taking into account whether the breaking of the guidelines is an acceptable in terms of the benefits it provides – given that it is known to have some negative consequences. In other cases, where competing design alternatives exist, the guidelines can be used to show which design option is most likely to lead to a straightforward AA implementation.

4.1.2 Guideline Philosophy

An important underlying assumption to the guidelines is that if something is *easier* to do it is strictly better than something that is *hard* to do. In particular, being "easy" is directly proportional to the complexity of the software or the cognitive process in the case of a human, required to perform the task. We assume that the more complex the required software the more difficult the task of building it. The guidelines are designed to make AA as easy to implement as possible, which is equated with being better. The guidelines help in avoiding agent design decisions that make it very difficult to implement AA and advocate design decisions that make AA easy to implement. For example, if a complex mapping is required between the agent's representation of some concept and the AA software's representation of the same concept it is more difficult to build the software for the mapping. Hence, the situation is less desirable, than if a simple mapping or no mapping at all is required to get the agent's representation into a form usable by the AA software.

Notice that there is a subtle, important, necessary difference between having guidelines that lead to "good AA" and having guidelines that lead to "easy to build AA". This work is concerned with the services of the agents that provide the foundation for implementing AA. Clearly, the actual design and implementation of the AA will also be critical to whether the AA is effective. The guidelines given here aim at ensuring the AA services provided by the agent do not handicap the development of that AA and make it as easy as possible to do but do not advise directly on the actual design process of the AA. Hence, these guidelines can only make it easier to build useful AA, but not promise that useful AA will actually result.

Despite the differences in the details of the information requirements for different AA systems, at the agent design stage we should not be concerned with details. Rather, the aim should be to provide a solid base for as wide a range as possible of AA implementations. The guidelines aim to encourage the design of agents that provide a solid base for implementations of AA. The precise nature of that solid base will somewhat depend on whether software or a human is performing $\mathcal{R}$. This, in turn, means that some consideration needs to be given, at agent design time, to whether a human or software will perform $\mathcal{R}$.

4.1.3 Guideline Evaluation

The guidelines capture our experiences designing, implementing and working with AA systems. Two complete, very different, AA systems have been implemented and evaluated as described in Chapters 5 and 6. Both systems have been extensively evaluated and our experiences with the AA carefully analysed. Furthermore, analysis of existing AA literature has been a significant influence on the development of the guidelines. Evaluation of the effectiveness of the guidelines is performed in Chapter 7 by evaluating the impact of following and violating the guidelines on the system designs.

4.2 Building Agents For Adjustable Autonomy Systems

The guidelines are divided into two groups. The first group of guidelines are related to $\mathcal{I}$ services, i.e., those services providing information to $\mathcal{R}$. The second group of guidelines are related to $\mathcal{A}$ services, i.e., those services implementing $\mathcal{R}$ decisions. The $\mathcal{I}$ service guidelines are:

- Explicit Information Guideline
- Software Engineering Guideline

- Design Information Guideline

The $\mathcal{A}$ service guidelines are:

- Deterministic Execution Guideline
- Explicit Behaviour Guideline
- Building Blocks Guideline
- No Extra Mechanisms Guideline
- Design Expecting Failures Guideline

4.3 $\mathcal{I}$ Services

AA Information services provide the raw information which $\mathcal{I}$ will present to $\mathcal{R}$. Whatever information $\mathcal{I}$ services do not provide about the state and intentions of the agent cannot be used in the reasoning. Since $\mathcal{R}$ is obviously limited to reasoning about things it knows about, the $\mathcal{I}$ services limit $\mathcal{R}$ and hence the AA in general. The details of the precise information needed for $\mathcal{R}$ (and hence the precise services required) vary according to the particular application. Further, the form required for the information depends on whether a human or software is doing $\mathcal{R}$.

Agent designs differ in the way that information is represented in the agent and how easily it can be extracted in an understandable (to $\mathcal{R}$) manner. Whether the $\mathcal{I}$ services are simple, e.g., just providing access to particular parts of the agent's reasoning, or very complex, e.g., requiring complex algorithms to extract information from the agent, depends on the design of the agent. The key point is that it will be harder to extract information from some types of agents than from others. AA implementations will be easier and more likely to be successful if it is easy to extract relevant information from the agents in the system. The $\mathcal{I}$ service guidelines presented below aim to guide designers to create agents where as much useful, understandable information as possible can be gathered by simple inspection of the data structures of a running agent. The guidelines try to avoid the need for complex algorithms to extract useful information from a running agent.

We propose three guidelines for designing agents which will lead to easy to build $\mathcal{I}$:

- *Explicit Information Guideline* : Represent the agent's reasoning process and reasoning state explicitly and in a format easily translatable to the format that will be used by $\mathcal{R}$.

- *Software Engineering Guideline* : Follow good software engineering practices in agent design to make it easier to build AA.

- *Design Information Guideline* : Represent information that explains design decisions implicit in the reasoning process, regardless of whether or not it is needed for agent reasoning.

In the following sections we describe and motivate each of the $\mathcal{I}$ service guidelines in detail.

Explicit Information Guideline

The Explicit Information Guideline recommends that all information used, by an agent when reasoning be explicitly represented in that agent. Goals that are explicitly represented in the agent can be easily extracted and clearly show what the agent is trying to achieve. Similarly, an explicitly represented world model can be more easily extracted than an implicitly represented one and gives $\mathcal{R}$ insights into how the agent sees the world.

More subtlty, the mechanism for performing the actual reasoning should be explicit and intelligible (Goel et al. 1996). If the action selection process is a simple, clear, explicit process $\mathcal{I}$ services will more easily be able to extract useful information. A simple process requires less translation and inference to be done to get the reasoning state into a useful format for $\mathcal{R}$ than a complex process containing many implicit assumptions.

To see how the design of an agent affects how easily information can be extracted from it, consider the following two architectures. Agents can be built from simple, reactive behaviours that produce complex, apparently goal-directed behaviour when immersed in a complex environment (Brooks 1991*a*, Kelso 1995). When an observer looks at the run-time data structures of such an agent they will not find the goals it sees the agent pursuing – *those goals are implicitly represented.* Because the goals are implicitly represented either $\mathcal{I}$ has to try to infer those goals or $\mathcal{R}$ has to reason without a clear idea of what the agent is trying to achieve. Conversely, an agent architecture like PRS (Ingrand et al. 1992, Georgeff & Lansky 1987) has intentions and plans explicitly represented. Looking at the data structures of a running PRS agent would easily give an observer a good idea of what the agent is doing without requiring any complex reasoning. We contend that it is easier to build $\mathcal{I}$, and hence AA, for the latter rather than the former agent type.

More than just having information represented explicitly, the Explicit Information Guideline recommends having information represented in a way

that is most *easily translatable* to a form understandable by the entity performing $\mathcal{R}$. In an AA system, a translation between the representation of the agent and that of the $\mathcal{R}$ must be performed (usually by software) – the simpler that translation the better. The data representation used by the $\mathcal{R}$ will largely depend on whether a human or software is doing the reasoning. If software is doing the reasoning, formality will be required, but if a human is doing the reasoning, information needs to be represented in a format a person can understand. Some translations are trivial, e.g., Australian and American engineers need to translate between metric and imperial units when sending each other designs. But, some translations are far more complex, for example, translating a plan into a corresponding set of reactive rules or vice versa may be very complex and information may be lost or mis-interpreted. Hence, an agent design that allows simple translations to be done to the format required by $\mathcal{R}$ should imply that better "quality" information is given to $\mathcal{R}$.

Software Engineering Guideline

The Software Engineering Guideline says that following established software engineering practices when designing agents leads to an easier AA implementation task. The important factor is not the coding practices software engineering advocates, rather it is the important characteristics of software built according to software engineering guidelines. Many software engineering practices are specifically designed to make the resulting system easier to understand and modify (Sommerville 1996). It is especially this ease of understanding that is important when trying to implement AA. Good software engineering will make the process of translating agent information into a format usable by $\mathcal{R}$ easier because the information is already well structured and clear, with important information explicit in the design. Well engineered programs are self-documenting and easy to understand (Booch 1994). Furthermore, well engineered programs are easy to change because modules are well separated and the relationships between them are clear (Eder et al. 1992). Other principles like good naming conventions, use of named constants and a simple design are also important. Each of these aspects provides something that allows an observer to better and more easily understand how the agent works and hence why it is doing what it doing and what it will do next.

In some ways AA can be seen as taking the idea of rapid prototyping software engineering to an extreme – i.e., AA is (re-)engineering software while it runs. Understanding the running of a piece of software is a prerequisite

of knowing how to change it, regardless of whether or not the software is running or not. The same thought process will be used to make a change to a piece of software regardless of whether or not it is running. Given the similarity between software engineering and AA, it is not surprising, therefore, that well software engineered agent specifications are easier to understand and change at run-time and are therefore better than those that are poorly engineered.

It might be argued that this guideline is obvious and superfluous – clearly well software engineered agents are better than badly engineered ones. However, there are two important reasons for including it here anyway. Firstly, good software engineering gives more to AA development than to other software development because we actually utilize the good engineering at *run-time* instead of just at *design-time*, i.e., a good design makes the job of understanding the agent's behaviour at runtime easier. Secondly, often good engineering is traded off against other concerns, such as efficiency, when designing any piece of software. Such tradeoffs are perfectly reasonable provided that they are based on solid reasoning. This guideline points out to designers that the cost of trading off good engineering for some other concern is higher when building agents for AA systems than for some other types of software.

Design Information Guideline

The Design Information Guideline recommends representing as much design information as possible explicitly in an agent, regardless of whether or not that information is required for the agent's reasoning. Information such as assumptions, approximations, limitations, etc. should be represented so that $\mathcal{R}$ can understand characteristics of the agent's behaviour. This type of information is critically important when $\mathcal{R}$ is trying to determine whether changing decision making responsibilities from one agent to another could result in better system performance (Dorais & Kortenkamp 2001). For example, $\mathcal{R}$ may be able to change decision making responsibility for some task from one agent to another when it sees that the currently responsible agent's assumptions are violated by the current situation. Getting relevant design information, if it is not explicitly represented in the agent, can be extremely difficult. So, for $\mathcal{R}$ to come to a good decision about reassigning decision making control if the assumptions inherent in the agent's design were lost in some designer's head may be very difficult. Effectively that requires $\mathcal{I}$ to be able to look at a specification and determine that specification's strengths and weaknesses.

Representing dependencies between different parts of an agent's reasoning is an important part of representing design decisions. All dependencies should be clear and explicit. For example, the design of one behaviour in a behaviour-based architecture might rely on another behaviour and the absence of the other behaviour causes it to fail. If the relationship between the behaviours is not seen at run-time, $\mathcal{R}$ might not fully realize the implications of a change to either behaviour. Hence, changes might be made to one behaviour with unexpected effects on the agent's overall behaviour.

For example, imagine the classic subsumption robot example: a wall following robot (Mataric 1994). In its simplest incarnation such a robot has two independent motors for wheels on either side of the robot and three behaviours controlling its actions. One behaviour, the *forward* behaviour, makes the robot go forward by providing equal current to both motors. A second behaviour, the *left wall* behaviour makes the robot go right by adding an amount of current to the left motor proportional to the detected distance to the left wall. A corresponding behaviour exists for the right wall. A naive user wanting to stop a robot, say, stuck in a corner might try stopping the *forward* behaviour expecting it to stop the robots forward movement. But seeing as though the *left wall* and *right wall* behaviours will still provide current to the motors, if walls are detected to the left or right, the robot will continue moving. The user makes the mistake because the behaviours appear independent though they are really very dependent on each other. Because the dependencies are not explicitly represented it may be difficult to understand those dependencies.

To summarise, the Design Information Guideline advocates including design information regardless of whether or not it is required for agent reasoning. The information required includes capabilities and limitations and dependencies between modules. The design information is useful for $\mathcal{R}$ and is difficult to reconstruct if not represented explicitly.

4.4 AA Actuation Services

In its most concrete form, a change in autonomy means a change in the data structures of an agent. $\mathcal{A}$ services implement autonomy changes decided on by $\mathcal{R}$ by changing the internal data structures of agents. Only those autonomy changes that can be realized by $\mathcal{A}$ services can be decided on by $\mathcal{R}$. Hence, the limitations of the $\mathcal{A}$ services strictly limit $\mathcal{R}$ and the resulting AA. In theory, if an agent allows unlimited access to its internal structure, $\mathcal{A}$ could force it to do anything that the agent is capable of, therefore the $\mathcal{A}$ services would be powerful enough to do anything. So, if any part of

the agent's internal structure can be changed online then the autonomy and behaviour of the agent could be changed arbitrarily online. However, the nature of the agent's representation of its behaviour affects the *ease* with which an outside entity can dynamically alter the agent's behaviour. That is, it will be harder to make appropriate changes to some agents at run-time than it will be to change others.

For example, imagine the difficulty of "re-programming" the ants building an ant hill to build wide flat hills instead of tall and skinny ones. The new plan would need to be mapped to individual ants and simple pheromone based rules designed – possible (assuming we could re-design ants) but far from trivial.

Thus, an important aim when building $\mathcal{A}$ is to make the changes required to the internal structures of the agent to affect a particular autonomy change as simple as possible, therefore making the AA as easy to implement as possible.

Further, the more smoothly the agent incorporates any autonomy changes decided on by $\mathcal{R}$ into its ongoing behaviour, the better the behaviour of the overall system. For example, imagine a team of (human) furniture movers carrying a piano up a staircase. If their foreman yells out from another room that one mover is being re-assigned to another job the reassigned mover will (hopefully) not simply let go of the piano and walk off (leaving his colleagues squashed under a piano at the bottom of the stairs) but will make the piano safe before moving on. Thus, the behaviour of the overall "furniture moving" system would be successful because the reassigned worker switches between tasks in a reasonable manner. The same idea applies to AA where smooth autonomy changes mean better system behaviour. Further, if the agent could exhibit similar "common sense" behaviour of the furniture movers, $\mathcal{R}$'s task would be simpler because changes can be made without (unnecessary) consideration given to transitions of the agent's behaviour.

These guidelines summarise design strategies for building agents whose behaviour can be most flexibly and easily changed online:

- *Deterministic Execution Guideline* : Make the reasoning process of the agent as deterministic, hence predictable, as possible.
- *Explicit Behaviour Guideline* : Represent behaviour as explicitly as possible and in a format that requires the least translation from that used by $\mathcal{R}$.
- *Building Blocks Guideline* : Divide overall behaviour into small pieces that are related to each other in a very semantically clear and simple

way.

- *No Extra Mechanisms Guideline* : $\mathcal{A}$ should not invoke mechanisms other than the normal reasoning mechanisms used by the agents.
- *Design Expecting Failures Guideline* : The agent should be designed so that if any part of it fails at any time its behaviour will degrade gracefully.

In the following we motivate each of the $\mathcal{A}$ service guidelines in more detail.

Deterministic Execution Guideline

The Deterministic Execution Guideline says that the workings of the decision making system should be predictable so that it is possible to work out what the agent will do next (within the bounds of the uncertainty in the agent's environment). $\mathcal{R}$ needs to know what an agent will do in order to decide whether autonomy changes are required. For example, if an agent will make appropriate decisions autonomously there is no need for authority to be withdrawn. Furthermore, if agent decision making is predictable $\mathcal{R}$ can work out what will happen when particular autonomy changes are made, allowing it to make autonomy decisions confidently and decisively (Rajan et al. 2000). For example, if $\mathcal{R}$ "knows" that an agent will behave "sensibly" when authority to pursue some goal is revoked it can confidently revoke that authority (if for some reason revoking that authority is in the best interests of the overall system.)

The Deterministic Execution Guideline somewhat reflects the old show business adage "never work with children or animals". Because the actions of children and animals are not always predictable, directors and actors have a hard time making sure a performance comes out as planned. Conversely, a car is a good example of how deterministic execution makes it easy for an outside entity to have good control. Although most drivers do not understand the details of the workings of the car, the car's actions are very predictable and the actions the driver takes have very predictable effects on the car, so drivers can safely (usually) drive. The point is that predictability, stemming from determinism, leads to better control over an agent.

Explicit Behaviour Guideline

The first step to implementing an autonomy change decided on by $\mathcal{R}$ is to determine what changes to the internal data structures of the agents need

to be made in order to get the desired effect. This guideline builds on the observation that things that are explicit are easier to change. Getting a desired effect requires a translation or mapping from the "language" of the requested change to the "language" of the agent. The more similar these two "languages" are, the easier the mapping. For example, at the extreme, a translation from an explicit natural representation to a connectionist representation where much behaviour is emergent, is likely to be extremely difficult. Concepts that are represented explicitly in one "language" but implicitly in the other, are likely to be problematic for mapping, and hence, implementing AA is likely to be difficult.

A good example of the usefulness of explicit representation of behaviour is the ease with which explicitly represented behaviour can be changed as compared to implicitly represented behaviour in a human society. (The following example ignores the "human aspect" of the rules and focuses just on the basic mechanisms for change.) Laws encode explicitly rules by which human societies must abide. When circumstances change, laws are changed and society easily changes over to the new system. Driving on the left or right hand side of the road is a good example of this, as is a change to the tax system. In both cases society (relatively) easily and quickly changes over to a different set of rules. On the other hand, social conventions are implicit in a society. These conventions change much more slowly and erratically than the explicitly represented rules. For example, it would be fair to say that society's behaviour with respect to, for example, woman's rights or gay rights, has changed much more slowly and with much more difficultly than the tax system. The aspects of behaviour that have changed most slowly are those that cannot be enforced by anti-discrimination laws. Although, there are other reasons for the slow change in behaviour it is at least in part because there is no explicit rule regarding, for example, attitudes that can be easily changed.

Hence, explicitly represented behaviour makes the task of changing the behaviour of an agent easier because it makes it easier to locate and reason about the parts of the behaviour that need to be changed.

Building Blocks Guideline

The Building Blocks Guideline recommends aiming for agent designs where overall behaviour is built up from small pieces combined via some, preferably formal, well understood mechanism. The pieces are like building blocks that can be put together in different ways to achieve different results. It is important that it is clear how the building blocks fit together and what the

effect of a certain configuration of blocks will be. The smaller the building blocks the more flexibly they can be put together leading to greater flexibility in the agent's behaviour. Hence, the smaller the building blocks the more different behaviours can be achieved without having to delve into the details of the building blocks. Furthermore, a building block architecture makes it easier to isolate some aspect of the agent's behaviour because it is more likely to be encapsulated in a single "block" than spread out over the whole architecture.[1]

This guideline assumes that the mechanisms connecting building blocks are easier to work with than those internal to the building blocks. It further assumes that building blocks are easier to configure in different ways than monolithic structures.

Pragmatically, there is a limit to the changes that can be made to behaviour at run-time, e.g., complex equations controlling a non-rigid robot arm (Chang & Chen 1998) might be too complicated to change safely online. The size of the building blocks should be equivalent to the smallest detail of the agent's behaviour that might change at run-time. This means that configurations of building blocks can be changed but the building blocks can be "black boxes".

The Building Blocks Guideline builds on the software engineering principle of *high cohesion and low coupling* which advocates designs where individual software modules have a clear, single purpose and are as loosely connected to other parts of the system as possible (Booch 1994, pg. 137). For software engineering, this principle simplifies the overall design and makes it easier to change individual modules. As it does for software engineering, high cohesion and low coupling make it simpler to put building blocks together. This simplicity is critically important if we want to combine the modules at runtime. Clean, high level, flexible interconnections between behavioural building blocks makes it easier to add and remove blocks.

Architectures in which different aspects of behaviour are tightly coupled, like neural networks, are more complex to change than loosely coupled ones because the effect of a change has potentially more complicated effects in tightly coupled agents. Architectures like neural networks have all the agent's behaviour coalesced into a single network, while other, for example, behaviour based architectures spread the overall behaviour out over a number of smaller modules connected together via a well known mechanism. We contend that the latter architecture is better for AA because the behaviours

[1] Though some aspects of behaviour might be due to interactions between building blocks.

can be switched in and out to produce required overall behaviour more easily than changing the weightings on neural network nodes to have the same effect.

Hence, this guideline advocates using architectures where behaviour is made up from loosely coupled modules, connected via a well defined mechanism because in such architectures it is easier to change an agent's behaviour.

No Extra Mechanisms Guideline

The No Extra Mechanisms Guideline says that no special purpose mechanisms should be built solely to implement behaviour requested by $\mathcal{R}$ (Blumberg 1997*b*). For example, adding the capability for a neural network based agent to use simple rules just to implement $\mathcal{R}$ decisions is inadvisable. The rationale for the inclusion of this guideline is mainly pragmatic, as complexity and expense increase if extra mechanisms are used. If extra mechanisms are used, the overall system becomes more complex hence more difficult to design, build, verify, test, more prone to errors, etc. Furthermore, the AA is also more complex because the architecture of the agent is more complex. In any case, the need for extra reasoning mechanisms may indicate a design flaw in the original agents – why should the AA need to ask something of the agent the agent cannot do with its "normal" reasoning mechanisms? Hence, this guideline could be thought of as a guide to errors in a design rather than pointing out the way something should be done.

Notice that the No Extra Mechanisms Guideline is not in conflict with the Design Information Guideline. The former says not to build new *reasoning* mechanisms while the latter says that more *information* should be built in than is strictly necessary. Hence, although one guideline says "do not do extra" and the other says "more is needed" they are referring to quite different aspects of a design and, as such, are not in conflict. A good example of the distinction is an extension that was required to a PRS based AA system to show what potential plans that will be used to handle possible upcoming situations. No change in the basic reasoning mechanism was required but extensions were needed for the agent to explain what it will do next (Musliner & Krebsbach 1999). The extensions did not add to the complexity of the agent design, which did not change, but were required to make $\mathcal{R}$'s task less uncertain by letting it know what the agent will do next.

Design Expecting Failures Guideline

The Design Expecting Failures Guideline advocates designing an agent to be tolerant to failures occurring in any aspect of its behaviour at any time. The reason for this guideline is that often the effect of AA on an agent is the same as if an aspect of the agent's behaviour had failed unexpectedly. When AA reduces an agent's autonomy (e.g., takes away responsibility or authority) the effect is similar to the effect of the component pursuing the goal failing. In one case the agent suddenly has no authority to do something it did have authority to do while in the other case it suddenly could not achieve something it thought it could. For example, consider an agent with a certain sequence of sub-goals planned to achieve some top-level goal. The effect of the AA suddenly withdrawing an agent's authority to pursue a certain sub-goal on the agent's overall plan is effectively the same as the sub-goal failing to be achieved. Hence, if the agent has mechanisms to deal with its own failure the same mechanisms may well deal with changing autonomy.

As with the Software Engineering Guideline, this guideline should probably be adhered to for all types of agents, even those not likely to be used in AA systems. However, the rationale for including it is the same as that for the Software Engineering Guideline, i.e., the guideline encourages agent features that provide *extra* benefit to AA, hence tradeoffs causing violations of this guideline should be taken even more seriously than they normally would be.

Further, designing for failure builds robustness into the agent that will give more flexibility to the AA at run-time. If the agent is capable of dealing with problems in some of its components then $\mathcal{R}$ and $\mathcal{A}$ can be less "careful" when making autonomy changes, instead relying on the agent to deal with minor details, via its failure mechanisms. That is, an unexpected change to one of its components can be dealt with by the agent as if it were a failure. If the agent has good failure handling then $\mathcal{A}$ needs to "worry" less that some change it makes to the agent will cause an overall failure, clearly making $\mathcal{A}$ easier to implement.

From an anthropomorphic perspective the Design Expecting Failures Guideline goes some way to providing the "common sense" of an agent. In other words, an agent's ability to react "intelligently" to some changing authority or responsibility via appropriate reactions can be seen by an observer as common sense. We contend that the flexible, robust handling of changes to behaviour will result in behaviour that gives the impression of "common sense" and make the functioning of the overall system better in the face of autonomy changes. For example, an agent for controlling an unmanned

aircraft might be designed so that if any part of its mission failed it would return to its home airport. When an AA system removed its authority to enter some airspace rendering the mission impossible the aircraft will return to its airport – a "common-sensical" reaction.

Thus, the Design Expecting Failures Guideline advocates designing an agent as if any part of it could fail at any time because the resulting agent features make the agent and system behaviour more reasonable when autonomy is changed.

4.5 Summary

The guidelines presented above give recommendations for the design of intelligent agents to be used in AA systems. Each guideline focuses on issues that are important to consider when designing agents to ensure that AA will be easy to design and implement. Clearly, not all guidelines can be followed at all times, but at least the guidelines give a designer an opportunity to make informed tradeoffs when designing agents.

Each guideline is designed to encourage the development of specific features of an agent which, we contend, make some aspect of the task of building AA simpler. Table 4.1 summarises the intended agent features for each guideline and why those features are useful for AA.

In the following chapters two systems, developed according to the above principles, are presented and evaluated. An evaluation of the guidelines, using the presented implementations is given in Chapter 7.

Guideline	Agent Feature	AA Usage
	$\mathcal{I}$	
Explicit Information Guideline	Explicit representation of important information	Easy extraction of important information
	Reasoning explicitly represented	Easy to understand what agent is doing
Software Engineering Guideline	Software easy to understand	Easy for $\mathcal{I}$ to understand workings
	Software easy to change	Easy for $\mathcal{A}$ to change software
Design Information Guideline	Capabilities, limitations, dependencies, etc. explicitly represented	Provides useful information for $\mathcal{R}$
	$\mathcal{A}$	
Deterministic Execution Guideline	Predictable execution	Ability to know what agent will do next
		Ability to know the effects of changes
Explicit Behaviour Guideline	Important aspects of agent's behaviour explicitly represented	Agent behaviour easier to change
		Easier to map from required change to actual change
Building Blocks Guideline	Modular, semantically clear design	Flexibility at runtime
		Easier to isolate data structures responsible for behaviour
No Extra Mechanisms Guideline	None	AA does not increase agent complexity
Design Expecting Failures Guideline	Robust failure handling mechanisms	$\mathcal{A}$ needs to consider less detail

Tab. 4.1: *Summary of the agent features intended to be encouraged by each guideline and why those features are useful to AA.*

5. ADJUSTABLE AUTONOMY FOR INTERACTIVE SIMULATIONS

Virtual environments, inhabited by intelligent actors are emerging as a useful tool with a variety of purposes. Those uses include training (Tambe et al. 1995, Dörner et al. 2000, Cohen et al. 1989), testing (Craft & Karr 1996, Pew & Mavor 1998) and entertainment (Doyle & Hayes-Roth 1998, Reynolds 1995). Intelligent actors often play the roles of humans or other intelligent entities in the virtual environment. Functionality which allows a user to have run-time control over the behaviour of an actor is useful because it gives the user more control over the progress of the simulation. Adjustable Autonomy (AA) is one way of providing a user with that run-time control of an actor in a simulation. *Bringing AA into interactive simulations means that the behaviour of the actor is not fixed when the simulation begins.* This property opens up new, exciting possibilities for developing and using actors, as well as solving some existing problems.

In this chapter we describe the problem of building actors which support the straightforward development of powerful, flexible AA systems. We contend that following the guidelines from Chapter 4 when designing actors will make it straightforward to build those $\mathcal{I}$ and $\mathcal{A}$ interfaces. We describe a specific actor development tool called EASE (End-user Actor Specification Environment). EASE supports a type of *Directed AA*, i.e., the autonomy reasoning ($\mathcal{R}$) is performed by a human while the AA information collection task ($\mathcal{I}$) and the realization of autonomy changes ($\mathcal{A}$) are performed by software. An EASE actor is not "aware" that its autonomy is being changed, the user simply makes decisions about appropriate behaviour and enforces the effects of those decisions on the internal data structures of the actor.

The focus of this chapter is on the features of the EASE actor design that support the development of effective $\mathcal{I}$ and $\mathcal{A}$ interfaces. Prototype interfaces have been developed and are presented to illustrate the utility of the AA services provided by an EASE actor. EASE was designed according to the guidelines from Chapter 4, hence an evaluation of the simplicity with which AA interfaces can be developed serves as an evaluation of the guidelines.

The remainder of this chapter is organised as follows. Section 5.1 introduces the domain of interactive simulations and, in particular, the two simulation environments in which EASE was tested. Section 5.2 looks at the specific requirements that such simulation environments pose for AA. Sections 5.3 through 5.8 look at the design and implementation of EASE. The description highlights the AA aspects of the system and especially looks at the influence of the guidelines on the design. Section 5.7 gives an extended example of the use of the AA in EASE for making run-time changes to the behaviour of a simulated combat aircraft pilot. The chapter concludes by looking at some similar systems for developing simulated actors.

5.1 What are Interactive Simulations?

For the purposes of this work an *interactive simulation* is a simulation of some virtual environment where intelligent software entities and humans interact. Simulations of virtual environments have a variety of different uses. A common use is for testing and analysing the properties of some complex system before detailed design of the system is done or physical prototypes are built. For example, before going through the long, expensive process of designing and building a new military aircraft, designers can evaluate the anticipated abilities of the new aircraft in a simulated theatre of war to determine desirable properties of the new aircraft (Craft & Karr 1996). A variety of popular computer games, such as Creatures (Grand & Cliff 1998), EA Sports NHL (EASports 2000) and The Sims (Smith 2000) rely on having intelligent characters inhabiting an interesting virtual environment. Virtual environments are also used as a cheap, effective training aid. Flight simulators are one of the most common and cost effective virtual environment training systems (Saab 1998), but many other types of training, including disaster management training for emergency services personal can be effectively performed with an interactive simulation (Tadokoro et al. 2000, Granlund 1997).

In interactive simulations it is necessary to not only simulate physical artifacts but also the intelligent entities, usually humans, that inhabit the environment. The intelligent entities in an interactive simulation are called *actors* (Banks & Stytz 1999, Wavish & Connah 1997).[1] The actors are an

[1] Wavish & Connah (1997) uses the term actor when referring to agents playing roles of humans because no effort is made to make the agent brain work in the same way as a human brain, rather, like a human movie actor the agent tries to give the illusion of being something they are not.

integral part of the simulation and the believability of their behaviour is often essential to the usefulness of the simulation (Tambe et al. 1995).

Over the years many approaches have been taken to creating intelligent actors for simulation environments, e.g., (Blumberg & Galyean 1995, Wavish & Connah 1997, Pew & Mavor 1998, Hayes-Roth 1995, Fischer et al. 1994, Rosenbloom et al. 1991). A variety of very different research themes have been pursued – ranging from how to build human like characters (Burt 1998) to dealing with resource constraints (Ogasawara 1993) to easing engineering problems (Bryson & McGongile 1998) to empowering end-users to specify an actor's behaviour (Travers 1996, MacKenzie 1996) to getting actors to learn (Davidson 1998). The current state of the art allows intelligent actors with a wide range of behaviour to be built with a reasonable amount of effort. However, although it is often possible to get intelligent actors to perform as we would like, achieving the required behaviour can be a time-consuming and costly process.

AA for Streamlining Actor Life-Cycle

A feature common to much actor research is the actor's life-cycle (see Figure 5.1). First, an *actor expert*, i.e., an expert in developing actors, probably an AI researcher or professional, specifies the behaviour of the actor using some language. Then, the actor specification is used by a *domain expert* in some simulation environment. The domain expert is the person who is actually interested in the results of the simulation, e.g., trainers, game players or analysts (Pew & Mavor 1998, pg. 11). If the actor does not perform as the domain expert requires, the simulation is stopped. The required behaviour changes are communicated to the actor expert who makes appropriate specification changes and delivers the actor back to the domain expert.

The need to stop the simulation when actor behaviour is not as desired, call in the actor experts and restart the simulation is a costly, time-consuming and error prone process. AA provides a mechanism to *interactively* change the actor's behaviour. AA provides the functionality which allows a user to take over aspects of an actor's decision making, without having to take over permanent or complete control. The user's decisions about how the actor should pursue some goal or what goals it should pursue are realized via changes to the actor's internal data structures. For example, a user might assume decision making responsibility for a particular goal the agent is not acting appropriately towards or assign to the agent goals they are currently pursuing.

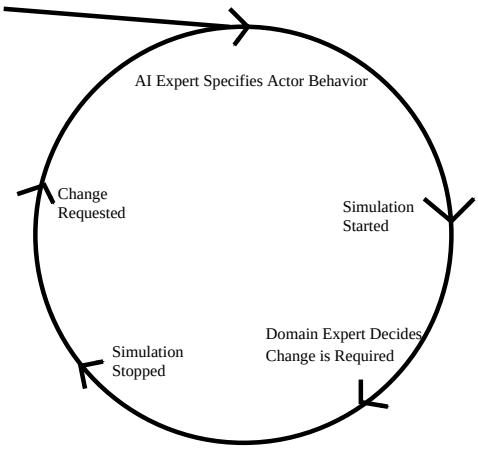

Fig. 5.1: *Standard life cycle of an actor. Each time the domain expert requires a change in behaviour the simulation is stopped and an actor expert consulted to make the change.*

AA can be leveraged as the basis for a radically different, simpler and cheaper actor life-cycle, as shown in Figure 5.2. In this life-cycle, like the previous one, an actor expert specifies the behaviour of the actor and the simulation is started. However, if the domain expert decides that the actor's behaviour should be changed *they can simply change it online, themselves – without even stopping the simulation.* Such a life-cycle promises significant savings in time and cost, thereby making interactive simulations a more useful tool.

Furthermore, with such a life-cycle in place, aspects of the actor specification that are unusually difficult or time-consuming to create can be left out of the specification and the domain expert relied on to "assist" the actor through the "hard to handle" situations when they occur.

The idea of using AA to increase the utility of simulation environments is not a new one. As early as 1995, Blumberg & Galyean (1995) had noted the need for such functionality and implemented a system which allowed humans to manipulate the behaviour of actors in interactive theatre applications (see also Section 3.3.2).

Notice that the use of AA in interactive simulations leverages the same basic features of AA as are leveraged in other domains, but for interactive simulations AA is a useful technology rather than an essential one. That is, we *could* stop the simulation and make the required changes but this

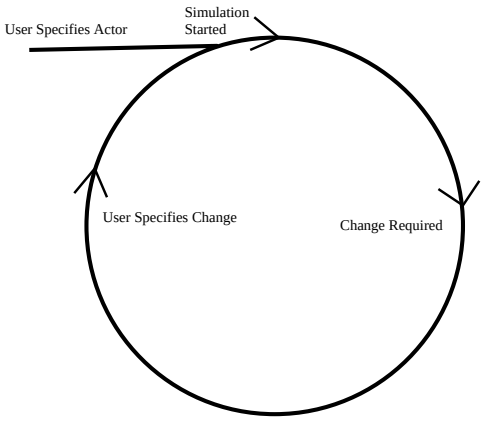

Fig. 5.2: *Actor life-cycle for an EASE actor using AA to change actor behaviour online. Notice that the cycle is much simpler than the one shown in Figure 5.1.*

would be a tiresome process – AA allows a faster process. In other domains, e.g., for control of manned spacecraft, intelligent systems will simply not be deployed without AA technology, i.e., for those systems AA is an essential technology.

Defining AA for Interactive Simulations

AA for interactive simulations is a special case of AA as defined in Section 2.1. In interactive simulation environments, AA allows a human to dynamically take over decision-making responsibilities from an actor when they so desire. In particular, there are only two intelligent entities in an AA system for interactive simulations, the actor, denoted a, and the user, denoted u. Potentially there could be more than one actor, but each actor-user pair can be treated as a separate AA system, so we need to concern ourselves only with the simpler case. The user is strictly in command of the simulation, hence the user's authority set, C_u, will be constant throughout the simulation. However, the actor's authority set, C_a, can change, provided that at all times $C_a \subseteq C_u$.

For the purposes of interactive simulation we assume that all the instructions that the user gives will be accepted by the actor. For example, a simulated pilot will not refuse to, say, attack some target when ordered to by the user. However, sometimes conflicts with higher priority goals

may mean concrete actions are not taken by the actor towards an assigned goal, i.e., actions toward the newly assigned goal might conflict with higher priority goals so no actions are taken to achieve it.

In terms of the definitions from Chapter 2, the system will transfer decision-making responsibility from G_a to G_u when the user takes over decision making for a goal. Conversely, goals move from G_u to G_a when the human gives control to the actor. Notice, a request to the actor to take on an additional goal should be granted only if $g \in A_a$, i.e., the actor should not be asked to do something of which it is not capable.

The transfer of decision-making responsibility will not always be explicit and, in particular, the actor may not even be "aware" it has lost or gained decision making control over some goal.

$\mathcal{I}$ must provide information to the user about the state of the actor's reasoning. This information is extracted via the $\mathcal{I}$ services provided by the actor. $\mathcal{A}$ changes the behaviour of the actor according to the decisions made by the user. The decisions are realised via changes to the actor's internal structures, accessed via the $\mathcal{A}$ services. Because $\mathcal{R}$ is being performed by a human, human-computer interfaces need to interface between $\mathcal{I}$, $\mathcal{A}$ and the user.

The results of the user's decision making, for the goals for which they have taken over decision making control, are realized directly by changes to the actor's internal data structures. That is, when the user has decision making control, they make decisions and realize the effects of their decisions on the actor by changing the actor's internal data structures. The user is responsible for both $\mathcal{R}$ and decision making, i.e., when the user decides that a change will be made to the actor, they are (explicitly or implicitly) deciding that for some goal the actor's reasoning is inappropriate and that they should assume decision making responsibility. For example, a user might decide that an actor's decision making for *acquiring materials to build a house* was not going well and might assume decision making for that goal – this is performing $\mathcal{R}$. $\mathcal{A}$ transfers decision-making responsibility for the *acquiring materials to build a house* goal from the actor to the user. Then, the user fulfills their responsibility to the goal by deciding the best thing to do would be to go to the forest and cut down trees. The user might decide that the sub-goal of cutting down trees in the forest should be assigned back to the actor. That is, $\mathcal{A}$ is used to give decision-making responsibility for the destruction of the forest to the actor. Thus, the user has assumed decision-making responsibility for the high level goal, decided on a course of action and forced the actor to decide on the details of that course of action.

5.1.1 Evaluation Simulation Environments

EASE has been evaluated by creating and using actors for two simulation environments: air-combat simulation and RoboCup football. Additional projects using EASE to provide the intelligence and AA functionality are being undertaken at Örebro Universitet and the Swedish military. Students at Örebro Universitet are applying EASE to a disaster management simulation and the simulation experts at a Swedish military college are evaluating EASE for use in a command and control simulator.

RoboCup Simulation

The first of the simulation environments used was the RoboCup football simulator (Noda 1995). In RoboCup actors are players in a simulated football match. RoboCup players need to react quickly, yet strategically in a highly dynamic and very complex environment. RoboCup simulation presents challenges to actors such as uncertain and incomplete sensing and acting, limited communication, a complex, dynamic environment, hostile, intelligent opponents and a need for team coordination (Kitano et al. 1997).

The RoboCup simulator uses a client-server architecture where each client represents one player in the football game. Each player can only communicate with its team-mates via a strictly limited communication channel provided by the server.[2] The players can take one of three actions, dash, kick or turn, during each simulator cycle. The player can also turn its neck to see a different part of the world. Sensing and acting cycles are asynchronous, so the player must reason about the effects of its action on the world because it will not always get sensor input about the changes caused by previous actions before selecting its next action. Players benefit from careful strategic analysis of game situations but the short time between actions (100ms in World Cup competitions) means that any deliberation either needs to be very fast or performed over a number of cycles.

RoboCup players are generally quite complex and need to act in an environment that presents a very wide variety of scenarios. When developing players, two types of tasks take a large amount of a developer's time. The first is debugging player behaviour. Re-creating problems and determining their cause takes a significant amount of developer's time and, hence, has attracted a considerable amount of attention from RoboCup developers, e.g. Stone et al. (2000) and Takahashi (2000). Finding and correcting prob-

[2] We actually do not use this communication channel at all, so our players do not communicate.

Fig. 5.3: *Snapshot of the RoboCup simulation monitor. The larger half light and half dark circles represent the 11 players of each team. The lighter side of the player shows the direction it is facing. In the bottom left hand corner is a record of the commands the automatic referee has sent to the players. In the bottom right hand corner is a record of the commands sent by the players to the server.*

lems with complex actors in complex environments is a notoriously difficult task (Ndumu et al. 1998). Strange behaviour by a player may be due to strange timing between the server and player, may be due to errors in a specification, may be due to uncertainty in actions or sensing or may be a result of an unexpected situation which the players mis-interpret.

AA can speed up the debugging process in two ways. Firstly, presenting the state of the actor's reasoning to the developer can make the process of forming a hypothesis about the reason for unexpected behaviour faster and more accurate. The $\mathcal{I}$ part of the AA provides the information the user needs to form hypotheses about the cause of a problem. Secondly, AA speeds up the process of testing hypotheses by allowing the user to interactively make changes to the specification of the actor and immediately see the effects.

Debugging requires creativity, experimentation and freedom, using intuitions based on detailed design knowledge (Beizer 1990, pg. 10) – the AA should support this. A typical debugging routine might be to observe the unexpected actor behaviour, form some hypothesis about why the behaviour occurred, make some change to the specification to confirm (or perhaps deny) the hypothesis, then run the simulation with the new specification. If no hypothesis can be formed as to the reason for the incorrect behaviour "random" changes to the actor might be made, in the hope that the resultant behaviour will provide hints to the cause of the problem. EASE speeds up the debugging process by allowing a developer to interact with a running actor, to test hypotheses and try solutions. Anecdotally, developers of the Headless Chickens IV RoboCup team, based on EASE, found the AA facilities invaluable for debugging incorrect player behaviour(Scerri, Reed, Wiren, Lönneberg & Nilsson 2001).

A second time consuming problem for RoboCup team developers is evaluating tactics in a variety of different game situations (e.g., corner kicks, attack, defense, late in the game, etc.) against a variety of different opposition tactics. Running hundreds of games, hoping all possible situations occur, and observing the results is not an efficient way of meeting this aim. In such situations, the user is trying to answer questions starting with phrases like *What would happen if ...?* or *How can I ...?*. This process can be sped up if the developers can interactively specify changes to the player's behaviour, allowing different things to be tried with less effort (Cremer et al. 1995*a*). With AA, the user can experiment very quickly, making changes and directly seeing the results. AA capabilities allow developers to more reliably and quickly create situations for testing, which could have a significant impact on the time it takes to develop a RoboCup team.

Air-combat simulation

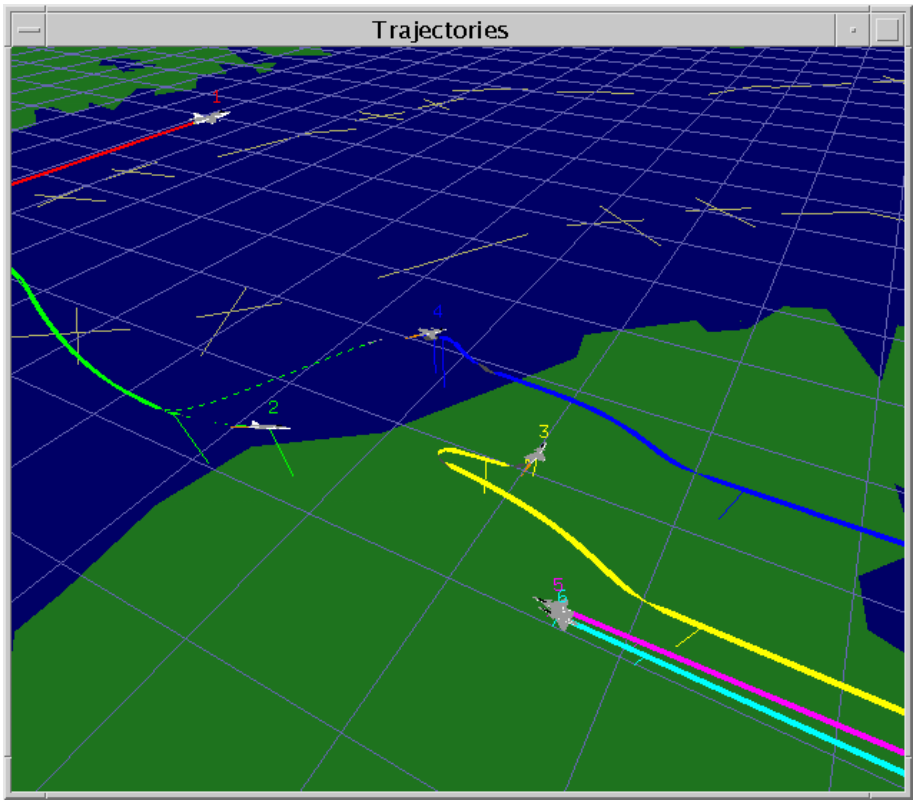

Fig. 5.4: *Snapshot of the TACSI air-combat simulation. Solid lines show the path of the different aircraft as they engage over the East coast of Sweden. The three aircraft in the bottom right corner are engaging the three in the top left part of the screen.*

The second test bed used for evaluating EASE was TACSI (TACtical SImulation), a beyond visual range air-to-air combat simulator developed at Saab Aerospace (Saab 1998). Beyond visual range air combat involves highly complex, military aircraft, fairly widely separated, relying mainly on missiles to attack each other (Coradeschi 1997, Andersson 1995, Thorstensson 1997). The TACSI simulator was originally developed for evaluating systems and tactics but is now also used for pilot training, in conjunction with a "dome" cockpit simulator and desktop units.

In TACSI, actors play the roles of the simulated pilots of both enemy

and friendly aircraft. It is very important that the observed behaviour of the simulated pilots is as close to the behaviour of the human pilots they are simulating as possible (Tambe et al. 1995). Situations develop more slowly in TACSI than in RoboCup allowing more time for the simulated pilots to plan and execute a strategy. Because more time is available for computation, the simulated pilots are expected to seem more intelligent than the football players.[3] A simulated aircraft is a far more complex system to control than a simulated RoboCup football playing robot, with many more degrees of freedom and more complex constraints on dynamic behaviour. For example, the pilot must control the aircraft's speed, altitude, radar, missiles, etc., each of which is complexly constrained by the abilities of the real aircraft, e.g., the turning circle and acceleration of an aircraft are strictly limited (whereas a RoboCup players acceleration is almost infinite and virtually independent of the player's current state). The most important difference between the domains, from an actor designer's point of view, is that a simulated pilot will be simultaneously attending to multiple objectives while a RoboCup player can perform acceptably well focusing on only a single objective at a time (Scerri & Reed 1999). This is significant for AA because an actor's behaviour is significantly more difficult to understand through observation if multiple objectives are being pursued simultaneously.

5.2 AA Requirements

The actor life-cycle utilizing AA set out in Section 5.1 provides a framework from which we can determine general requirements on AA for simulations. In this section these requirements are examined in detail and mapped to specific requirements on the AA services actors should provide. In subsequent sections a solution, i.e., EASE, meeting the requirements is presented.

In Section 2.3, a conceptual model of AA was presented. The model has three parts: $\mathcal{I}$ supplies relevant information to $\mathcal{R}$ which makes autonomy decisions that are realized by $\mathcal{A}$. In simulation domains, $\mathcal{I}$ and $\mathcal{A}$ are implemented in software. $\mathcal{I}$ extracts information about the state of the actor and presents it to the user. $\mathcal{A}$ changes the internal data structures of the actor in accordance with user requests for changes in behaviour. $\mathcal{R}$ is the responsibility of the user because the user needs to decide when the actor's behaviour should be changed. Figure 5.5 shows the instantiation of the conceptual AA model for interactive simulations.

[3] We will avoid the question of whether the same would be expected in real life.

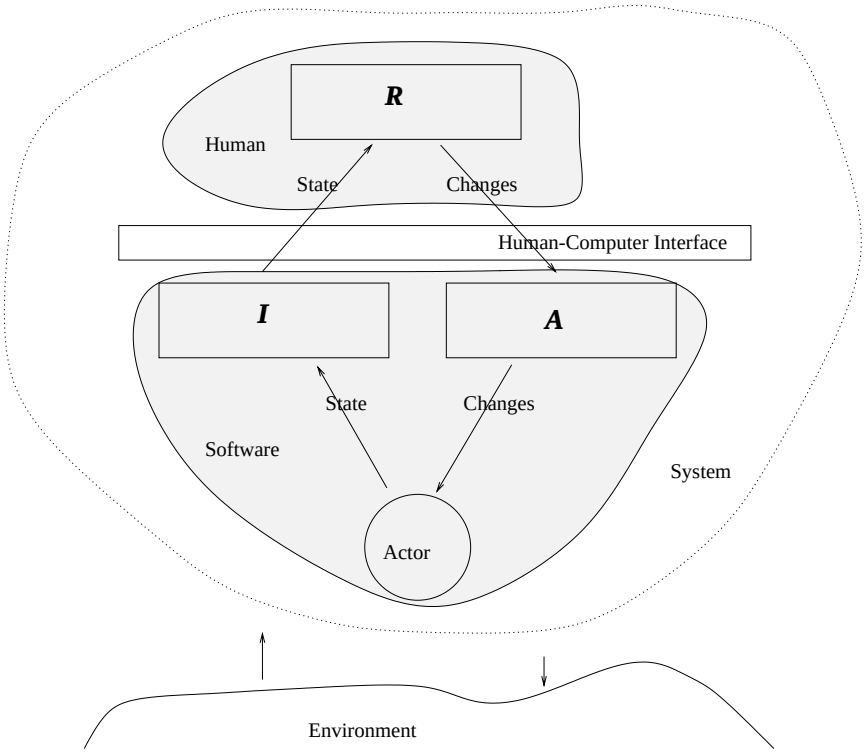

Fig. 5.5: *The conceptual configuration of actor, user and AA for interactive simulations. $\mathcal{I}$ and $\mathcal{A}$ are implemented in software while a human does the $\mathcal{R}$. Some type of human-computer interface needs to connect the software parts to the human reasoning.*

5.2.1 What should AA Information Do?

Recall from Section 2.1.2 that $\mathcal{I}$ is responsible for eliciting and presenting relevant information about an actor's state for use by $\mathcal{R}$. Figure 5.6 shows the basic flow of information from the actor to the user. The user, i.e., the entity performing $\mathcal{R}$, *knows* what the actor *should* do. $\mathcal{I}$ needs to help the user understand *why the required behaviour is not happening and what they can do to make the actor do the right thing.* The information provided by $\mathcal{I}$ supplements the information the user can gather about the actor's behaviour by observing a visualisation of the simulation. The visualisation shows the actor's external behaviour and the $\mathcal{I}$ gives its internal reasoning.

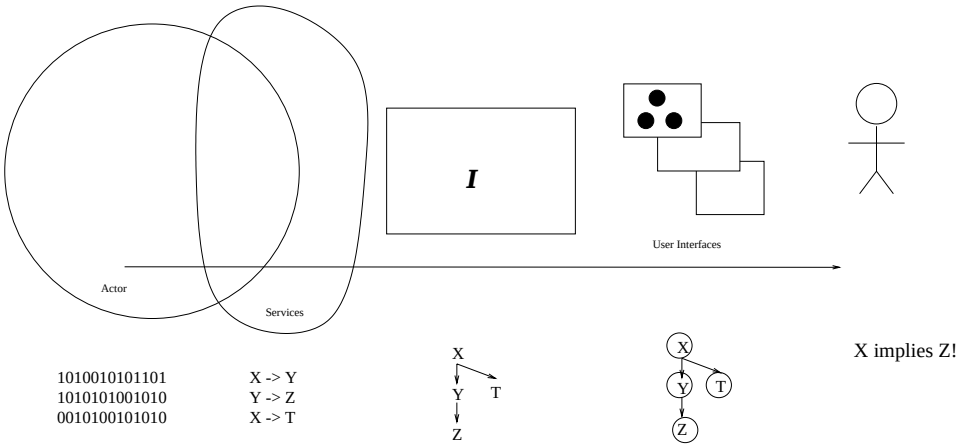

Fig. 5.6: *This diagram shows the flow of information from the actor, via $\mathcal{I}$ services and $\mathcal{I}$ to a user. When the user receives the information it is in a format they can understand. Across the bottom of the diagram is an example of the types of information representations that might be used at each step.*

The $\mathcal{I}$ services need to extract relevant information from the data structures of the running actor. Eventually, $\mathcal{I}$ will need to make the information it gets from the AA services intelligible to the human user. A variety of techniques might be required to implement the desired functionality of the human computer interface. Independent of how the actual interface will work, the more humanly intelligible the information passed from the actor services to $\mathcal{I}$, the easier it will be to implement the $\mathcal{I}$ interfaces. The less intelligible the information the $\mathcal{I}$ services provide the more work that needs to be done translating it, simplifying it, abstracting it, etc., before presenting

it to the user.

When debugging an actor's behaviour, the user will require detailed information about the actor's reasoning (Brann et al. 1996). The user will be attempting to find anomalies in the reasoning process. To this end they will require both the status of the actual reasoning process (i.e., *what is the actor thinking?*), the information that is used in the reasoning process (e.g., *does the pilot think that tank is a threat?*) and the behaviour specification the actor is following (e.g., *what will the pilot do when when it realizes it cannot climb out of range of the tank?*). In training scenarios there is a requirement that nothing untoward happens when the trainer makes a behaviour change because strange actor behaviour will disrupt the training. Hence, $\mathcal{I}$ needs to communicate the effects of proposed changes on actor behaviour so the user can assess those effects a priori. For pre-emptive avoidance of problematic actor behaviour the user needs to know what the actor *intends* to do. For example, *will the simulated pilot pull the aircraft up before it collides with the ground or does the user need to direct the actor to do so?* If the actor is not taking some action that it appears it should be taking, the user needs to know if there is a valid reason for the lack of action that they have not recognized, e.g., *is the aircraft avoiding a missile or positioning itself for attack?*

5.2.2 Summary of AA Information Service Requirements

For $\mathcal{I}$ to be effective $\mathcal{I}$ services need to supply the following basic information:

- The current, future and past *state* of the actor's reasoning process;
- The information on which the reasoning is based, i.e., the actor's understanding of the world and the specification of its behaviour; and
- The likely implications of user actions on the actor's behaviour.

5.2.3 What should the AA Actuation Do?

Sometimes a user will decide that different behaviour is required of an actor and will take over decision making to change the way something is done. The job of $\mathcal{A}$ is to realize the behaviour changes in the actor. The flexibility that $\mathcal{A}$ has to realize changes and the elegance with which it can do so is dependent on the services provided by the actor to change its own behaviour. Further, the ease with which $\mathcal{A}$ can be developed is directly related to the properties of those actor services – some services will make it easy to develop

interfaces, others will make it difficult. Via some interface the user needs to describe the required changes to the actor's behaviour. The basic flow of information, describing the required changes, from the user to the actor is shown in Figure 5.7.

In the most abstract terms, the changes that $\mathcal{A}$ needs to affect are simple – the user will want the actor to start doing something, stop doing something or (un)constrain the way something is done. An essential element of all the tasks is translating from the "language" of the user to specific changes in the actor. Hence, *the closer the representation of behaviour in the actor to the representation the user uses, the easier the translation that is required and hence, the easier the development of the required software.*

Not only must $\mathcal{A}$ make changes to the behaviour of the actor, it must make them in a way that allows the overall behaviour of the actor to remain reasonable, during and after the change. If changes requested by the user bring about abrupt or non-sensical actor behaviour (when abrupt or non-sensical behaviour is not required) the realism of the simulation may be compromised and, hence, the utility of the simulation reduced. For example, if a RoboCup player is dribbling the ball into an open goal and it is assigned the additional objective of conserving stamina, it should not slow down immediately (allowing defenders to take the ball away) but continue with the high priority objective of scoring before conserving stamina. Even better, the timing of the shot could be changed, e.g., a kick from further out, to save stamina *and* score the goal. A clean, smooth transition from the old to the new behaviour allows the simulation to remain effective. Furthermore, smooth transitions are likely to encourage the user to use the AA more because the actor behaviour is more believable – an important characteristic of a successful simulation (Sengers 1998, Blumberg 1997*b*).

Adding Goals or Constraints One type of behaviour change a user might want to make is to assign additional goals or objectives to the actor. If the user wants actions towards the goal to be immediately undertaken then, in terms of the definitions from Chapter 2, goals are added to G_a. That is, the new goal is added to the set of goals the actor has current decision-making responsibility over. Adding goals to G_a should mean that the actor immediately starts working towards the goal. If the user wants the actor to be able to *consider* pursuing some goal if an appropriate situation presents itself, then the goal is removed from the set C_a. Then, the actor may choose to pursue the goal if a situation arises where achievement of the goal is deemed to be useful. How easily this is performed in a particular actor

depends on the design of that actor.

Removing Goals or Constraints The second type of behaviour change the user might want to make is to stop the actor from pursuing certain goals. If the user wants to immediately stop the pursuit of some goal, the goal is removed from G_a. If the actor should not attempt to pursue the goal at any stage the goal should be added to C_a. This type of change allows users to reduce the authority of the actor.

5.2.4 Implementation Issues

Conceptually, the idea of AA for actors is clear, however implementing the changing goals and constraints on an arbitrary actor is not trivial. For example, the goals that the user wants to be achieved might not have a direct counterpart in the actor, e.g., if the user wants a RoboCup player to perform a *three man weave* but the player does not know what this is then somehow the behaviour needs to be explained to the player. The same difficulty occurs in reverse when removing goals from the actor. For example, to remove a goal that is the result of interactions between simpler goals and the environment (Brooks 1991*a*) requires mapping back to the goals that *are* actually represented in the actor and making changes to them. $\mathcal{A}$ needs to be able to translate from the user's request for adding or removing goals to the specific addition or removal of goals from G_a and/or C_a.

Furthermore, the translation can only be done like this if the actor has an explicit representation of goals. The actor may well not represent goals explicitly or even represent its own behaviour explicitly! For example, if the actor is controlled by a neural network, which has no explicit representation of goals, $\mathcal{A}$ would need to translate the user request into appropriate changes in node and connection weightings to affect the change in behaviour.

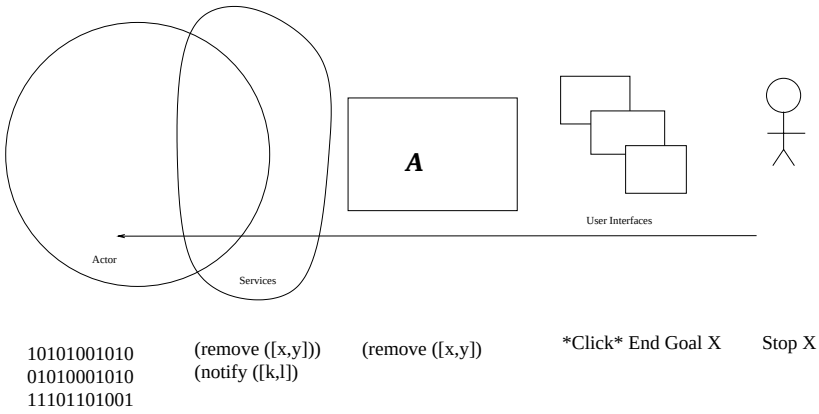

Fig. 5.7: *This diagram shows the flow of commands from the user, via $\mathcal{A}$ and $\mathcal{A}$ services to implementation in the actor. Along the bottom of the figure is an example of how a command from the user might get translated into changes in the actor.*

5.3 Overview of a Solution

The previous sections detailed the requirements on $\mathcal{I}$, $\mathcal{A}$ and the services underlying those sub-systems for interactive simulations. In this section we provide an overview of our prototype solution, called EASE (End-user Actor Specification Environment), that fulfills the requirements. EASE is described in some detail and AA interfaces described to show that the services provided by the actor perform the job required of them. We do not claim that this is the best or only actor architecture for implementing AA, only that this architecture makes implementing AA straightforward. For other applications, different tradeoffs may need to be made with respect to the capabilities of the actor architecture, for example, to do more sophisticated planning or be able to learn, which leads to a different architecture on which AA can be implemented just as straightforwardly.

EASE has a hierarchical *organisation of simple agents* for decision making. The basic idea is conceptually similar to the "Society of Mind" ideas of Minsky (1988) and that of *behaviour-based architectures* (Mataric 1994). Each *agent* in the *agent organisation* is responsible for pursuing a single goal of the overall actor. Explicit *contracts* between agents capture their relationships to one another. Agents in a *generic pool* of agents, representing the latent abilities of the actor, wait to be called on to pursue their par-

Agent feature	$\mathcal{I}$ service
Agents	Goals of actor (G_a)
Hierarchy	Abstraction
Negotiation	Explicit reasoning process, conflict resolution
Determinism	Implications
Generic pool	Abilities/authority of actor ($A_a - C_a$)

Tab. 5.1: *Selection of EASE actor features and the $\mathcal{I}$ service they provide.*

ticular specialty when (and if) it is required. Explicit negotiation between agents at the leaf nodes of the hierarchies determines the concrete actions to be taken by the actor.

For an EASE actor, the services that provide information to $\mathcal{I}$ are straightforward to build. Table 5.1 shows the important actor features from $\mathcal{I}$'s perspective and the information they eventually provide to the user (more detail is shown in Table 5.7, pg. 114). The current state of the agent organisation, which is explicitly represented, captures the *state* of the actor's reasoning. All the $\mathcal{I}$ services do is make this state information available to the user. The hierarchical aspect of the agent organisation is useful because it provides a built in abstraction facility which can be used to easily filter information for the user (Bryson 1999*b*). Little translation is required by $\mathcal{I}$ because the hierarchical goal structure is intelligible and intuitive to users (Travers 1996). The deterministic workings of the organisation make it easy for the history of the organisation to be maintained by monitoring changes to the organisation. The negotiation process by which the agents at the bottom of the hierarchies reach a decision on an actor action is explicit and easily extracted and translated so the user can see the conflicts between goals (and their resolutions).

$\mathcal{A}$ can also be straightforwardly implemented. Table 5.2 lists some important actor features from $\mathcal{A}$'s perspective and the controls they give to the user (more detail is shown in Table 5.7, pg. 114). Addition or removal of goals corresponds to addition or removal of agents from the agent organisation. Addition or removal of constraints corresponds to changing which agents can or cannot be contracted, i.e., adding or removing agents from the generic pool. A variety of other mechanisms allow details of agent behaviour to be manipulated.

Agent feature	$\mathcal{A}$ service
Adding agents	Assigning goals
Removing agents	Withdrawing goals
Removing/adding generic agents	Changing constraints
Changing named constants	Changing aspects of abstract behaviour
Changing intrinsic priority	Changing relative importance of goals

Tab. 5.2: *Selection of EASE actor features and the $\mathcal{A}$ service they provide.*

5.4 A Note About Agents and Actors

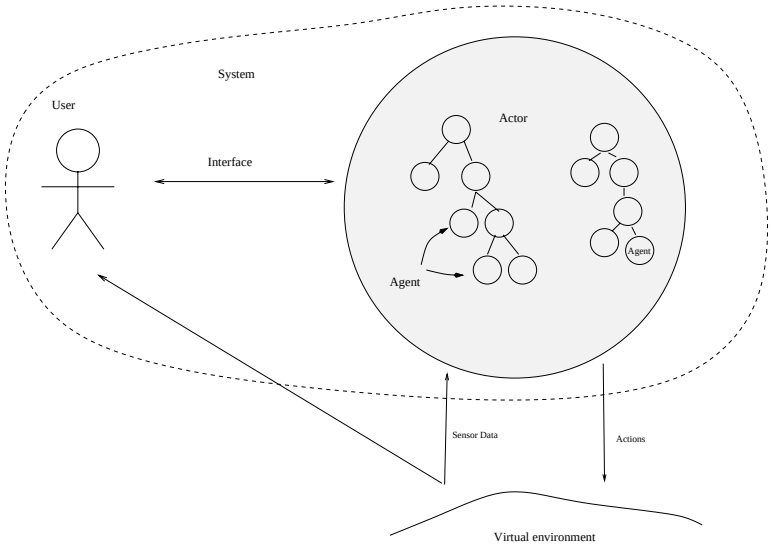

Fig. 5.8: *Relationship between an actor, agents and a user.*

The terminology of actors and agents may be a little confusing. An *actor* contains an *agent* organisation (see Figure 5.8). We use the name "agent" because we believe it best characterizes the autonomous goal directed behaviour of the components. Our concept of an agent is quite similar to the idea of a *behaviour* in a behaviour-based system.

The user interacts with the *actor*. In this particular prototype there is

no AA *within* the agent organisation, i.e., a particular agent's autonomy relative to the other agents is constant. The autonomy of the *actor* relative to the user changes dynamically. Other similar architectures may have AA between *agents*, e.g., (Gerber et al. 1999, Barber, Martin & Mckay 2000), but for EASE the AA is *only* between the actor and the user.[4]

5.5 EASE

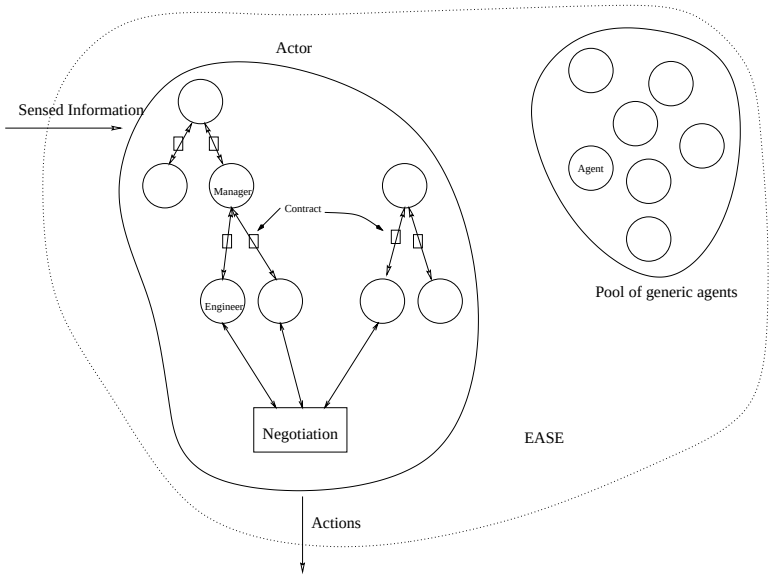

Fig. 5.9: *A high level view of the EASE run-time architecture*

Decision making in an EASE actor is performed by a society of interacting agents, called the *agent organisation*, as shown in Figure 5.9. The agents are hierarchically arranged, with agents higher in the hierarchy *contracting* agents further down the hierarchy to achieve specific goals for them. We use the names *contractor* and *contractee* when referring to the parent and child agent of a contract. Agents at leaf nodes of the hierarchy are referred to as *engineers*, as they are the agents that actually get things done. Other agents in the system are called *managers* – their job is to find and contract others to get things done.

[4] The ability to change the intrinsic priority of agents provides similar functionality to changing the number of votes in DAA (Barber, Martin & Mckay 2000).

Agents further up the hierarchy are responsible for more abstract tasks, while their contractees are responsible for more specific tasks (similar to Blumberg (1997*a*)). As the goals that the actor needs to achieve change, contracts are made and broken, hence the structure of the organisation changes. At any point in time the organisation, in particular the hierarchies, reflects the current task breakdown of the actor. Each agent in the organisation corresponds to exactly one g in G_a and the contracts correspond exactly to a parent-child goal relationship.

Engineers negotiate with each other about the concrete actions the actor will make. Each engineer "argues" for actions it decides will lead to the goal it is responsible for achieving. The negotiation algorithm draws inspiration from behaviour fusion algorithms (Saffiotti 1997) and partial distributed constraint satisfaction algorithms (Eaton et al. 1998, Ghedira 1994) but has been adapted to the requirements of a soft real-time environment.

As well as the agents within the agent organisation there is a pool of *generic agents*, each of which can be contracted to perform its specialty by an agent already part of the agent organisation. The generic pool effectively represents the *abilities*, A_a, of the actor, i.e., each agent in the generic pool represents something the actor can do. If there is an agent in the generic pool capable of achieving some particular goal then the actor can achieve that goal by contracting the agent to pursue it. If there is not an agent with the capability to pursue a certain goal in the generic pool then the actor cannot pursue that goal.[5] If the actor's authority to pursue a goal is withdrawn the agent(s) with the capability to pursue that goal is marked "unavailable", meaning that it may not be contracted. Hence, the agents that can actually be contracted from the generic pool correspond to $A_a - C_a$.

It is useful to compare the running of an EASE organisation with that of a human organisation. In a human organisation a manager will be responsible for a complex, abstract task like *Sales*. Subordinates of that manager are responsible for specific aspects of *Sales*, for example *Melbourne Sales* or *After Sales Service*. These people might have other people under them until finally we get down to the people that perform the concrete tasks that are the actual work of the company, e.g., selling to a customer. EASE agent hierarchies are designed and work in an analogous manner to such human organisations.

[5] The goal may still be achieved by the actor, but unintentionally rather than intentionally.

5.5.1 Social Conventions

All organisations require some conventions, rules or procedures to keep them running smoothly – the EASE agent organisation is no exception. The social conventions in the EASE agent organisation are far simpler than in most organisations. A contractee must inform its contractor about its progress toward the goal it has been assigned (we look at the process of forming a contract in the next section).

A contractee has three different *status messages* it can send to its contractor. The messages abstractly describe the agent's progress towards its assigned goal and allow managers to make decisions based on the status of sub-goals. One message indicates that the goal has been achieved (or is being maintained), one indicates that the goal cannot be achieved and one indicates that normal progress is being made towards the achievement of the goal (i.e., there is no reason to believe the goal cannot be achieved but it is not yet achieved). These messages are the only communication that occurs between agents once a contract is made. Although some aspects the organisation's functioning might be improved if, say, more status information were shared, the simplicity of the interconnections between agents makes the adding and removal of agents to and from the organisation easier. Being able to add and remove agents easily is critical when the *user* wants to add and remove agents, i.e., add or remove goals, at *run-time*.

Continuing the analogy with the human organisation, status messages are analogous to an employee keeping their manager informed of their progress toward the task(s) assigned to them. The manager uses this information in their decision making, as well as informing their superiors if the status of their own task is impacted by the information received from their subordinates. Without such information flowing around the organisation the organisation's operations would likely fall apart.

Contracts

There are three types of contracts a manager can make: fixed, dynamic or list contracts. The contract types differ in the way an agent is found to fulfill the needs of the contractor. The designer specifies which type of contract the manager will make at design time, the manager does *not* decide which is appropriate at run-time. The different types of contracts are summarised in Table 5.3 and discussed in more detail below.

Agents may be capable of fulfilling a *generic goal*, i.e., goals that can be tailored to a specific situation via the instantiation of parameters. The

Contract Type	Properties
Fixed	Specific agent to contract known in advance
Dynamic	Agent to contract found via capability matcher
List	One agent per environment feature contracted

Tab. 5.3: *The three different contract types a manager can make and their basic functionality.*

instantiated parameters for the goal are part of the contract with the agent. For example, a generic *get to waypoint* agent for a simulated pilot might have parameters specifying the location of the waypoint. The actor designer can parameterize agents very flexibly and the more generic the agents are the more reuse of agent specifications is possible.

Notice that the making of a contract actually creates a new instance of the contracted agent. Hence, the same "generic agent" can be contracted simultaneously many times and have many instantiations. This is an implementation detail that does not effect the basic functionality of the system.

Once entered into, each contract type works in the same way, i.e., the contracted agent continues working toward its assigned goal informing its contractor of its progress until specifically told to stop. If it achieves its goal and is not told to stop the agent may work to maintain the goal state or be idle. Contracts corresponds closely to the idea of delegation (Falcone & Castelfranchi 1999). In terms of the definitions given in Chapter 2, the process of finding and contracting agents to contract corresponds to the breakdown of a goal into sub-goals.

Fixed Contracts A fixed contract is used when a *pre-determined* agent will be used to perform a specific task at run-time. When the contractor determines that a particular sub-goal should be fulfilled at run-time it is already known precisely which agent will work toward the goal. If that agent is not in the generic pool, the sub-goal immediately fails, i.e., the contractor gets a failure message. Fixed contracts implement a fairly standard hierarchical decomposition of overall behaviour, e.g., Edmund (Bryson 1999*b*). Figure 5.10 shows the designer interface for specifying a fixed contract.

Dynamic Contracts For a dynamic contract the developer specifies only the *capability* required of the agent to be contracted. So the designer is specifying what needs to be done, rather than how it should be done. This is a more high level form of specification where more details are left to the

Fig. 5.10: *Tool for specifying a fixed contract. The manager being specified will make contracts with a safety agent, a conserve fuel agent and a patrol mission agent. The patrol mission agent contract has two parameters, "Rules of Engagement" and "Minimum Height" which are instantiated with the values "Aggressive" and "Safe Altitude" (which are named constants or more complex expressions).*

system. This type of contracting means that changes to the actor are easier to make because the structure is more flexible. For example, when an agent for pursuing some goal is removed and replaced with another agent that pursues the goal in a better way the contracts do not need to be changed.

At run-time the contracting manager checks with a *capability matcher* to find an agent with the required capability. The capability matcher in the prototype is very simple, merely doing string comparisons between the advertised agent capabilities of agents in the generic pool and the required capability. If a capable agent is located, a contract is created between the capable agent and the contractor. If no agent can be found with the required capability then the contractor receives a failure message and takes the same actions it would have if an agent had been found but failed to fulfill its assigned goal. Dynamic contracts are similar to Hayes-Roth's *abstract control plans* (Hayes-Roth et al. 1997), PRS's plans (Ingrand et al. 1992) and even somewhat like the idea of synergies (Kelso 1995). Figure 5.11 shows the interface for specifying a dynamic contract.

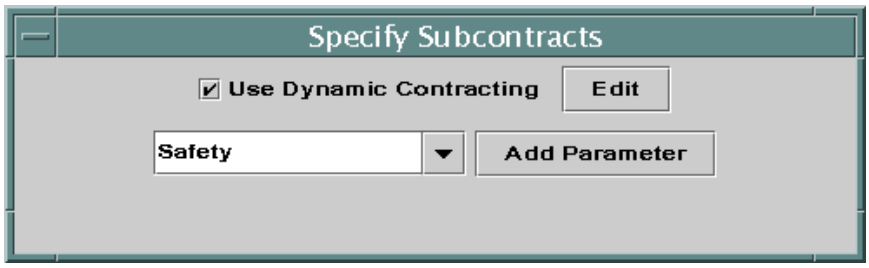

Fig. 5.11: *The end-user tool for specifying a dynamic contract. In this case an agent is required with the capability "Safety".*

List Contracts It has proven to be useful to implement one more type of contracting. In the domains we have looked at there are often multiple instances of an object in the environment all of which need to be dealt with in a similar manner. For example, a number of aircraft that need to be avoided, a number of football players that need to be marked or a number of missiles that need to be tracked. As a development aid for these situations *list contracts* were introduced. The designer specifies that for every instance of a particular environmental phenomena (e.g., an aircraft) the contractor will contract a separate agent with the particular environmental phenomena used to instantiated a parameter in the agent contract. Each object of that

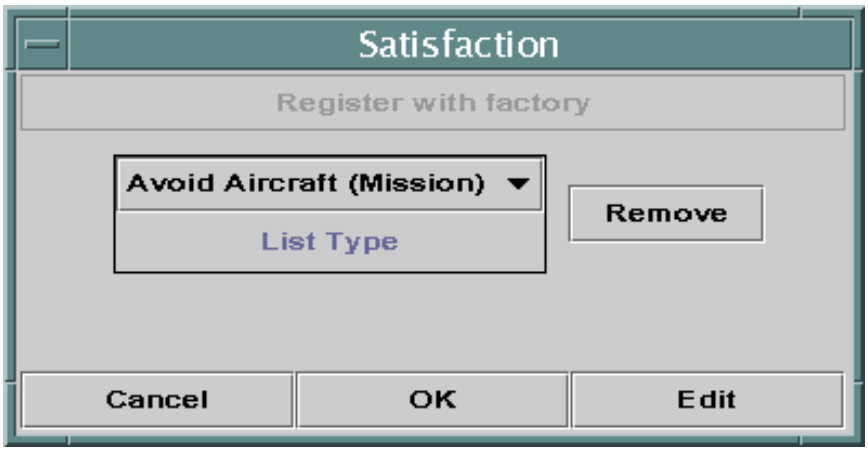

Fig. 5.12: *The tool for specification of a list contract.*

type that is "detected" at run-time, e.g., by the simulated radar or vision, results in one agent being contracted. So, for example, a separate agent might be contracted to track each missile or a separate agent might be contracted for watching each football player or a separate agent contracted for avoiding each aircraft. The system handles the details of contracting agents as instances of the object are detected. All the contracted agents report to the same manager – a special list contract manager, who in turn reports to the original contractor. Figure 5.12 shows the interface for specifying a list contract.

5.5.2 Managing Conflicting Goals

The agent organisation makes decisions about the actions of the actor via a negotiation process between engineers, i.e., the agents at leaf nodes. The basic idea is that each engineer argues selfishly (i.e., it argues for actions that will lead to its goal being achieved) and the negotiation protocol ensures that a fair (i.e., taking into account all the agents' wishes) outcome is reached. The way that such a process leads to good overall decisions is described mathematically in Ossowski & García-Serrano (1999) and informally below.

Each engineer has a set of *ideal states* it wants to bring about. It also has a model of the how each possible action will change the state of the environment and a model of the distance between an arbitrary state and its ideal state(s). The agent's *satisfaction* with an action is proportional

to the relative reduction in the distance between the current state and its ideal state(s) that can be expected from that action compared to other possible actions. So, the action(s) expected to lead to the biggest reduction in distance to the ideal state(s) will *completely satisfy* the engineer. Any actions expected to increase the distance to ideal states will *completely dissatisfy* the engineer. All other actions, i.e., those that are expected to reduce the distance to the ideal state(s) but not so much as the best action(s) will satisfy the engineer to some level proportional to the relative reduction in distance to the ideal state(s) they are expected to result in.

The engineer's satisfaction is captured by a *satisfaction function* which takes the current state of the environment and the proposed action and returns a satisfaction level. The satisfaction function can be specified in the spreadsheet-like function specification tool included with EASE. Figure 5.13 shows a screenshot of the tool.

The negotiation process involves generating potential actions (either randomly or via some intelligent stochastic algorithm) and asking each engineer for their satisfaction with the proposed action. The *best* action is determined by considering the relative satisfaction levels of the engineers for that action and their *priority* (see below). The higher the priority of the agent the more influence it has on the negotiations. The best action is "recorded" and periodically executed.

Only engineers are directly involved in the negotiation. However, the desires of the managers are conveyed indirectly via their contracts. Engineers involved in the negotiations will have been contracted by managers. Those engineers represent an aspect of the manager's interests in the negotiation. The manager affects how much influence its contracted engineers have in the negotiation via its influence on their priority (see below).

Negotiation / Behaviour Fusion There are a variety of other mechanisms that could have been used to settle conflicts between engineers. Brooks' original architecture used *subsumption* which meant behaviours could *inhibit* other behaviours preventing them from acting (or just reducing conflicting effects), hence allowing the agent's observed behaviour to be (relatively) cohesive. However, determining appropriate inhibition relationships in more complex behaviour based agents is difficult. Recent research has looked at mechanisms which allow active behaviours to *vote* on possible agent actions. The voting ideas introduced in DAMN (Rosenblatt & Thorpe 1995) sparked a flurry of research into better voting schemes, including (Yen & Pfluger 1995, Riekki 1998, Pirjanian 1998). Another way of resolving conflicts be-

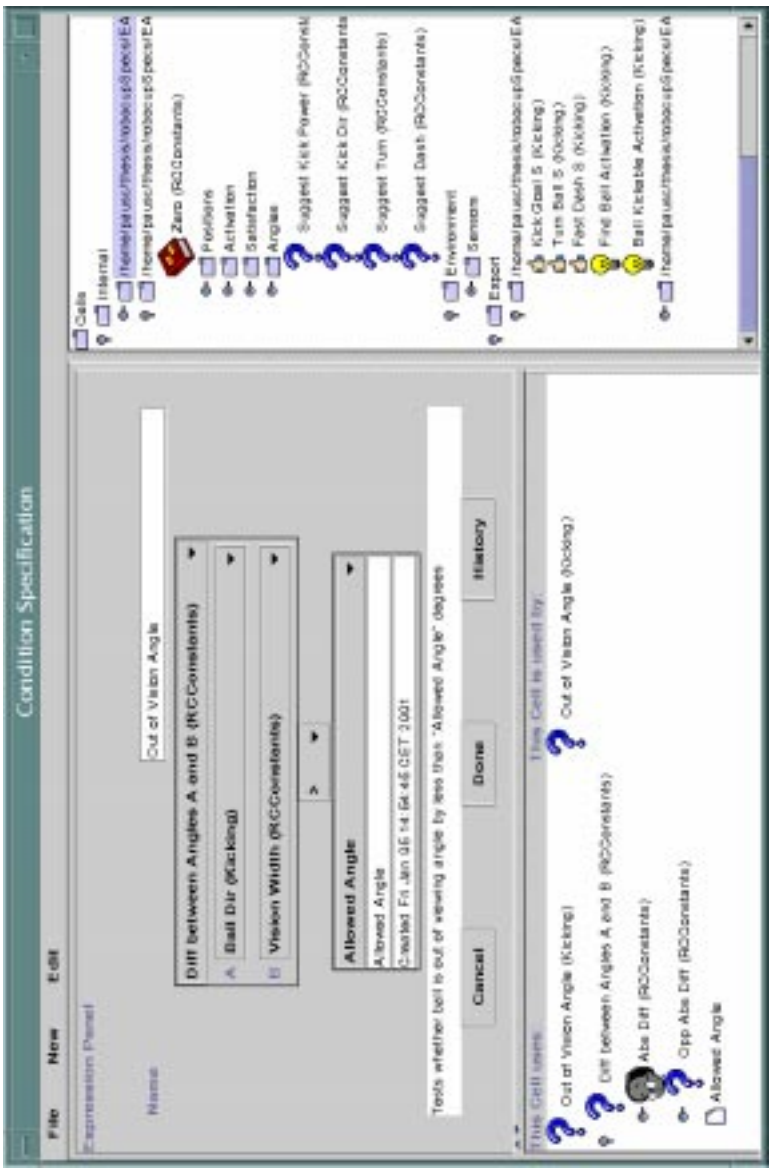

Fig. 5.13: *A snapshot of the condition specification sub-system for end-user programming of complex agent functions, showing a user editing one* cell, *a comparison between other cells, in the center of the window. At the bottom of the window is information about where the cell is used and the cells it uses. On the right of the window are the cells that have already been created.*

tween behaviours is to use fuzzy logic (Passino & Yurkovich 1998, Saffiotti 1997, Tunstel 1996). Different methods for defuzzification approximately mimic different voting algorithms. A variety of behaviour fusion algorithms are now well understood and have been used in a variety of real applications.

Priority

The importance of an agent is reflected by its *priority*. The priority of an agent affects how much influence it has in negotiations about actor actions – directly for engineers and indirectly for managers. An agent's priority, p, is a function of three factors, one static and two dynamic:

$$p = \min(e * i,\ o) \tag{5.1}$$

where e is the *environmental priority* of the agent, o is the *organisational priority* of the agent and i is the *intrinsic priority* of the agent.

Environmental priority, e, is the importance of the agent given the current state of the world. For example, clearly, avoiding an aircraft is more important when the aircraft to be avoided is close than when the aircraft is far away. Environmental priority will be higher when the obstacle aircraft is close and hence the agent with that environmental priority will have more influence on negotiations. Figure 5.14 shows an example of an environmental priority function for an obstacle avoidance agent. The environmental priority of the aircraft is higher the more directly the aircraft heads towards the obstacle and/or the shorter the distance to the obstacle.

Intrinsic priority, i, is the basic priority of the agent given the task it is assigned. Intrinsic priority allows the designer to capture the fundamental importance of the achievement of the goal, e.g., for simulated pilots avoiding aircraft is fundamentally more important than getting to a waypoint.

Organisational priority, o, is the component of the agent's priority given the task it is part of. For example, flying to a waypoint is more important when it is part of the task of rescuing trapped allies than when it is part of a routine reconnaissance mission. The organisational priority is a function of an agent's contractor's priority.

Table 5.4 gives some sample priority calculations, showing how the factors combine.

Agent Internals

We have so far skipped over how an individual agent makes decisions. Primarily, this is because it is not particularly important how the agents work

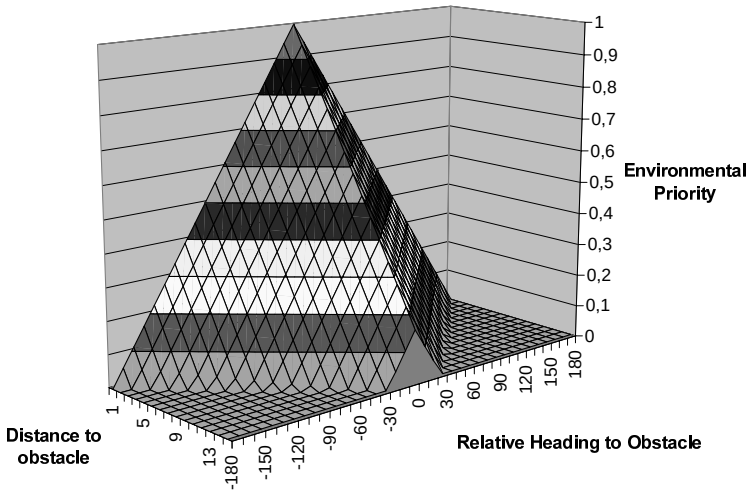

Fig. 5.14: *An example (simple) environmental priority function for an obstacle avoidance agent. The agent's environmental priority is higher if the obstacle aircraft is close and/or the aircraft is heading towards the obstacle.*

Situation	e	i	o	$\min(e * i, o)$
Near obstacle, contracted by *keep low profile* manager	1.0	0.6	0.3	0.3
Near obstacle, contracted by *safety* manager	1.0	0.6	0.9	0.6
Obstacle detected, but distant, contracted by *safety* manager	0.5	0.6	0.9	0.3
No obstacle detected, contracted by *safety* manager	0.0	0.6	0.9	0.0

Tab. 5.4: *Some examples of how overall priority is a function of environmental, organisational and intrinsic priority for a priority function in the obstacle avoidance agent.*

– any decision making mechanism will do. The important characteristics of the EASE architecture from the perspective of AA is that the complexity and "intelligence" of an actor is a function of the organisation and the negotiation process. This means that we can consider the inner workings of an agent to be a "black box" which cannot be observed or manipulated. The agents are intended to be simple. Such an approach is consistent with the Building Blocks Guideline. We describe the inner workings of the agents only because it makes understanding the architecture and subsequent evaluation easier.

The *external* properties of the agents are important to the organisation. For the proper running of the organisation, managers need to act by making contracts, engineers must participate in negotiations and all agents need to abide by the organisation's social conventions. How individual agents achieve these properties is irrelevant (and need not even be homogeneous across the actor) provided the external properties are correct. The important external characteristics of the agents and the role each characteristic plays are summarised in Table 5.5.

In the prototype, agent decision making is done via simple Moore state machines.[6] In each state, a manager contracts particular agents or, in the case of an engineer, uses a particular satisfaction function in negotiations. Figure 5.15 shows a screenshot of the end-user tool for specifying a state-

[6] State machines were chosen because they were most familiar to the intended users of the system – Saab system engineers.

Agent feature	Reason for feature
Important to organisation	
Satisfaction function	Used in negotiations
Activation function	Determines environmental importance of the agent
Capability	Specifies the tasks(s) that the agent can achieve
Success/Failure status	Fulfills social responsibility
Implementation details	
State machine	Determines agent's discrete behaviour
Success/Failure transitions	Handling of contractee status messages

Tab. 5.5: *Summary of the different aspects of an agent and their effect on overall agent behaviour.*

machine. As well as the state machine, each agent has a function that determines the environmental aspect of its priority.

To fulfill its social obligations each agent must inform its contractor of success or failure (no message is assumed to mean "normal progress"). This is implemented by allowing the designer to designate certain states as *success* states or *failure* states. When the agent transitions into or out of one of these states a corresponding message is sent to the agent's contractor.

Two special state transition types are provided which allow the behaviour of an agent to depend on the *success* or *failure* of its contracted agents or its performance in negotiations. Failure transitions provide a mechanism with which the designer can deal with the failure of an agent's sub-goals, i.e., the failure of contracted agents. Success transitions provide a means for the designer to specify how the agent is to move on with other parts of a task when certain milestones have been reached. If any contractee has sent a failure message then a failure transition can be taken. Conversely, if all a manager's contractees have sent success messages to the manager a *success* transition can be taken if it has one from the current state. Likewise an engineer can take a success transition if satisfied by the negotiations or a failure transition if dissatisfied by the negotiations. Table 5.6 summarises the transition conditions.

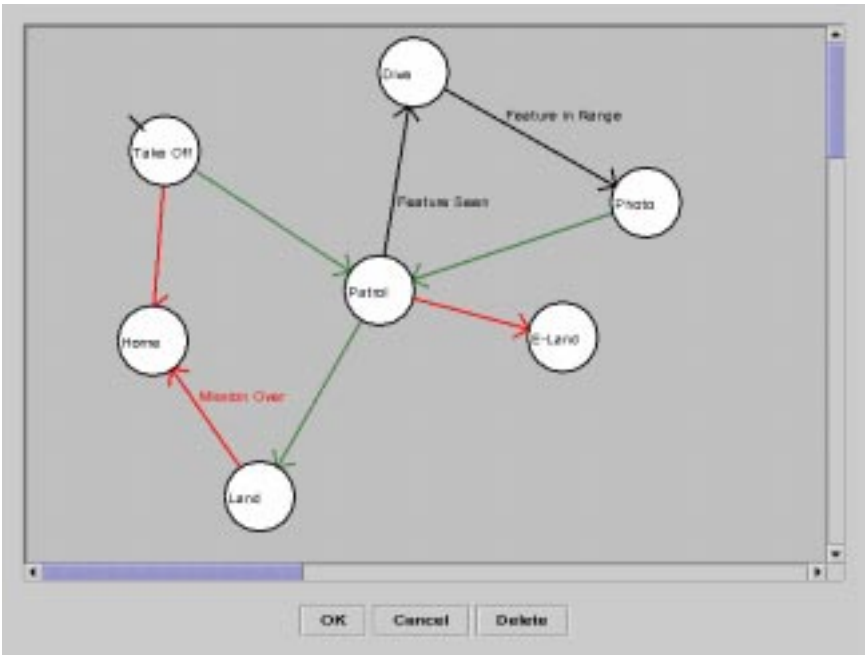

Fig. 5.15: *The end-user state machine specification tool, showing a state machine for a simple patrol agent. Circles represent states, while arrows represent transitions. The state in the top left corner, i.e., take-off, with an extra line is the start state. Success transitions are colored green while normal transitions are annotated with the condition for their traversal.*

Transition	Engineer	Manager
Failure	Completely unsatisfied with negotiation	Any contractee failing
Success	Somewhat satisfied with negotiation	All contractees succeeding

Tab. 5.6: *The table shows the conditions under which an engineer and manager will take success or failure transitions.*

5.6 Actor Services and Prototype Interfaces

In this section we describe the functionality of the EASE AA services and the interfaces built using those services. The services provided by the actor provide a large volume of information for $\mathcal{I}$. In addition, the actor services allow a wide range of control over both the agent organisation and generic agent pool giving $\mathcal{A}$, and thus $\mathcal{R}$, a wide range of flexibility. Figures 5.16, 5.17 and 5.18 show the relationships between the actor software and the different actor services.

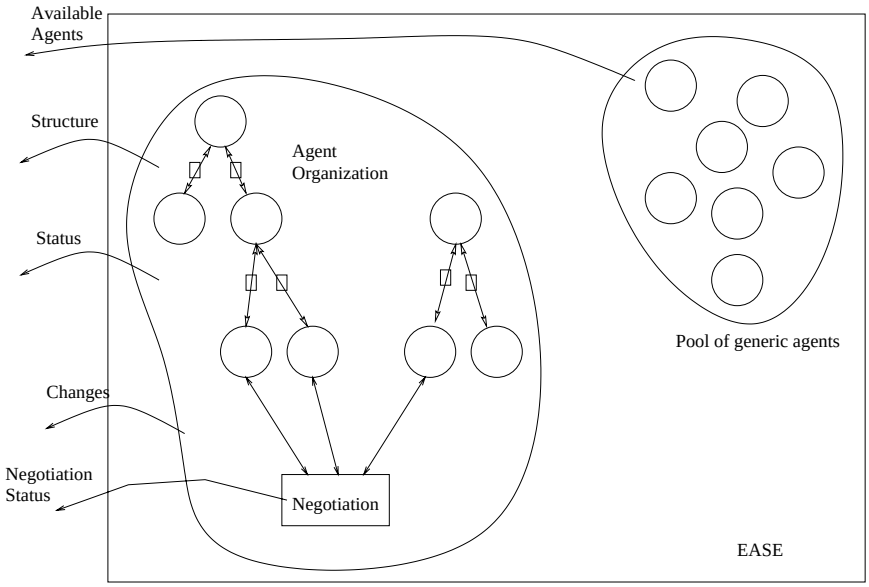

Fig. 5.16: *Information extracted from the agent organisation and generic pool. The surrounding box represents the extent of the EASE actor software. The arrows out of the box show what information is extracted and the parts of the software where that information originates.*

Recall from Section 2.1.2 that $\mathcal{I}$ provides information to the user (via whatever abstraction, inference and/or presentation techniques are required) which supplements the information the user gets from observing the simulation. $\mathcal{A}$ implements changes of decision making responsibilities from actor to user and vice versa, as well as realizing the results of the user's decision making on the behaviour of the actor.

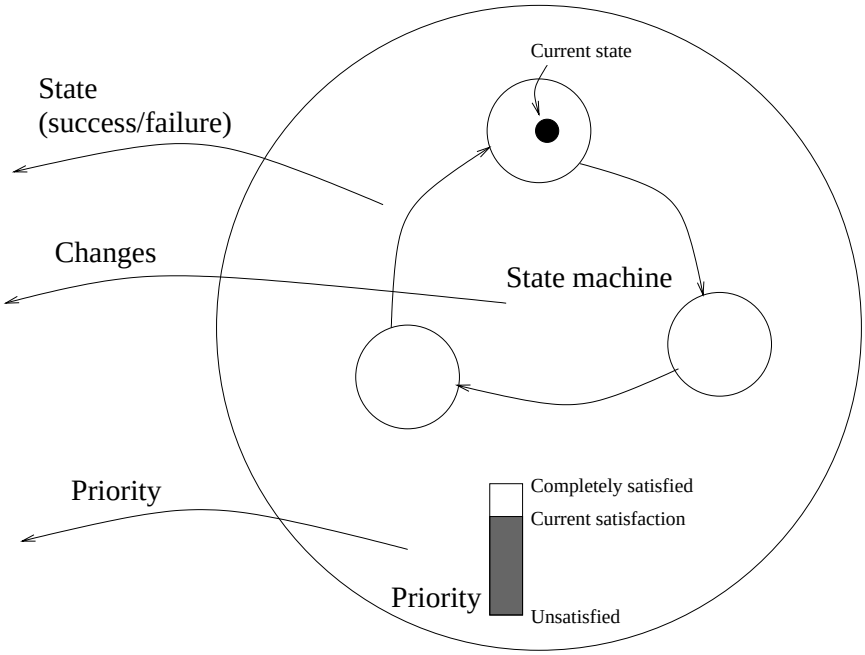

Fig. 5.17: *Information extracted from a single EASE agent. The large circle represents an agent, the arrows show which information is extracted from where.*

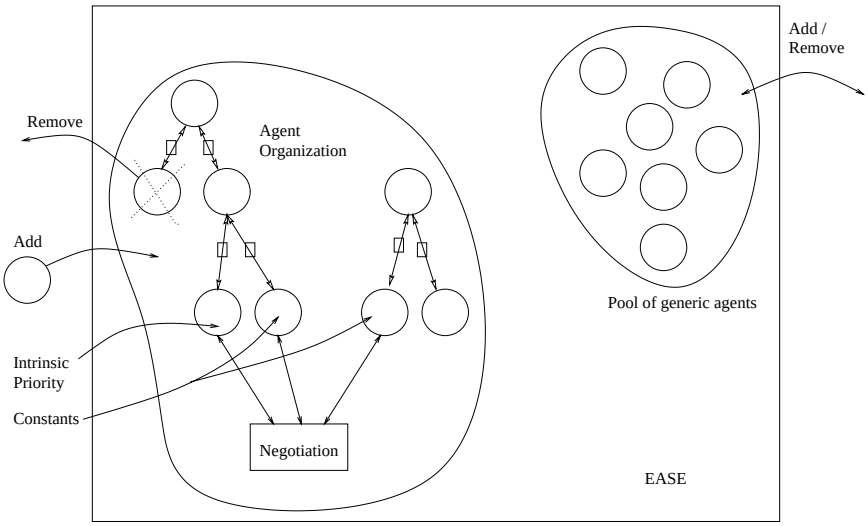

Fig. 5.18: *How the different AA actuation services affect an actor. The box represents the extent of the EASE software and the arrows show where $\mathcal{A}$ changes occur.*

Notice that the interfaces presented below are only prototypes, built to demonstrate the properties of the actor AA services and are not necessarily suitable for practical use. Better interfaces could and should be built if average end-users are to have a good level of control over actors in real environments. However, it is only the *details* of the human-computer interaction that are lacking in the interfaces. The interfaces *do* provide the basic functionality required for $\mathcal{R}$ to be performed by the user. Hence, by showing that interfaces with the basic required functionality can be built, given the services provided by an EASE actor, we show the utility of those services, which is our aim.

5.6.1 Information Services

The $\mathcal{I}$ services provide the following information to $\mathcal{I}$:

- The configuration of the current agent organisation
- The status of each agent in the organisation, including whether it is succeeding or failing, its priority and the current state of its state machine

- Details of ongoing negotiations
- Structural organisation changes that will result from different events
- Details of all the calculations performed by all the agents
- Available agents in the generic pool and the details of the agents, including their intrinsic priority and capabilities
- The values of named constants

Table 5.7 shows the important actor features and the services they provide to $\mathcal{I}$. The table also shows how each of the guidelines were followed in some aspect of the design and further led to desirable AA services. We show the value of each of the services by showing prototype interfaces that have been built leveraging the service. By showing the strengths and weaknesses of the prototype interfaces we demonstrate the strengths and weaknesses of the underlying services.

Together the different pieces of information provided by $\mathcal{I}$ services give the user an accurate and comprehensive picture of what the actor is doing, what it has done and what it will do next, which are important according to Brann et al. (1996). The user is responsible for finding the particular information they need to make decisions. Finding the appropriate information from the large amount generated and presented may be far from trivial but the format is generally appropriate for the task.

A key to the effectiveness of the EASE design for making $\mathcal{I}$ easy to develop is that the interactions between the agents are explicit and very limited. Each agent has a clearly defined purpose. Most relevant information is explicitly represented by the organisation, either in its structure or explicitly in an agent. The organisation is easy to understand because the agents are simple and loosely coupled.

Below we look at each of the services and the prototype interfaces leveraging the services. The prototype $\mathcal{I}$ interfaces and the information they provide to the user are summarised in Table 5.8.

Guideline	Agent Feature	AA Facility
Explicit Information Guideline	Agents and contracts	Goal hierarchy easily extracted, actor intentions known
	Negotiations	Conflict resolution easily extracted
Design Information Guideline	Agent hierarchy	Abstraction, decomposition information
	Design names kept at run-time	Easily understood functionality of components
Software Engineering Guideline	Naming conventions	Design information
	Parameterization	Simplified implementation of user commands
	Hierarchical decomposition	Relationships between agents clear
Deterministic Execution Guideline	Agent organisation	Implications viewer
Explicit Behaviour Guideline	Agent organisation mirrors goal hierarchy	Goal hierarchy is easy to change
	Named Constants	Changing behaviour dictated by constants
Building Blocks Guideline	Agents	Flexibility in changing behaviour
No Extra Mechanisms Guideline	None required	
Design Expecting Failures Guideline	Success and failure transitions	Smooth addition and removal of goals

Tab. 5.7: *Summary of some of the useful EASE actor features resulting from adherence to each guideline and the subsequent AA facilities.*

Information provided	EASE property	Tool
Structure of the current organisation	Explicit representation of agents	The Boss
Contents of the generic pool	Explicit representation of capabilities	Generic Pool Viewer
Details of specification calculations being carried out by specific agents	High level representation of calculations	Calculation Debugger
Details of the goal conflict management process	Explicit goal conflict management process, i.e., negotiation	Negotiation Viewer
Details of the priorities of goals	Explicit representation of agents and their priorities	Priority Viewer
Current values of all system constants	High level representation of calculations	Constant Viewer
History of the organisation	Explicit representation of agents	Transition Viewer
Effects of organisational changes	Deterministic execution	Implications Viewer

Tab. 5.8: *The relationships between information provided by the system, the property of EASE that makes that information available and the end user tool that leverages that information.*

The Boss

The Boss is a tool that (among other things) shows the status of the agent hierarchy within the actor. A snapshot of The Boss is shown in Figure 5.19. The tool is nicknamed The Boss because it provides most of the functionality for controlling an actor. A tree (or forest) is used to display the hierarchy (or hierarchies) in the agent organisation. A simple, collapsible tree structure allows the user to interactively change the amount of hierarchical detail that they see, i.e., the user changes the level of abstraction at which they see the organisation by expanding or collapsing branches. Details on the tree nodes show the type of agent (via colour), its name and its current state (in parentheses). The visualisation gives an accurate and comprehensive picture

of the agent organisation. The user can extract information such as "the actor is attempting to do X because goal Y is currently in a state where achievement of X is necessary" or "the safety management agent of the simulated pilot has not reached the point where it believes it is necessary to do an emergency climb", by making simple interpretations of the hierarchy.

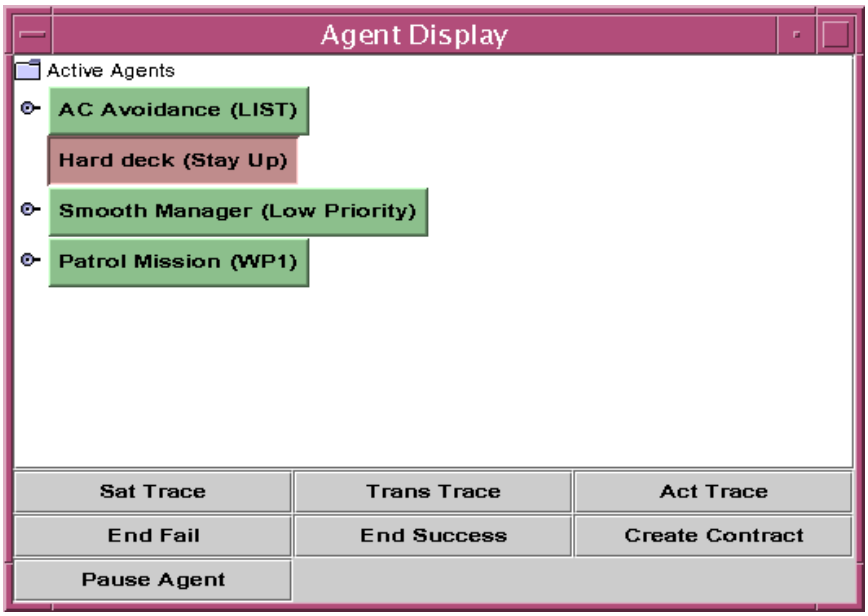

Fig. 5.19: *The Boss visualisation of the agent organisation. The state of each agent is shown in parentheses after its name. Agents with a small circle to their left are managers which can be clicked on to show contracted agents.*

The information for The Boss comes directly from the structure of the organisation which is explicitly represented in the actor. Notice that the interface makes virtually no translation from the underlying situation to the view presented. Firstly, this means the interface was (more or less) trivial to build. Secondly, it means that under a very wide range of circumstances the interface presents a faithful picture of the underlying situation. Any technique that needs to do a non-trivial translation will sometimes lose or mis-interpret information. Hence, the need not to do any translation gives us increased confidence in the accuracy of the visualisation.

Notice that it is the user's responsibility to determine what information is relevant. For example, the visualisation shows the whole hierarchy, but

only some parts will be important for any particular user purpose and the user needs to determine what those important parts are. Notice also, that the user is responsible for determining when details are and are not required (by expanding or collapsing branches of the tree). Better interfaces might be designed with some "intelligence" used to bring important parts of the organisation to the user's attention, e.g., agents that are failing. These more "intelligent" designs would, however, *not* necessarily require that the actor provide any additional information, the difference would be in the processing that the interface does with the information before presenting it to the user.

Transition Viewer

The *Transition Viewer* maintains a record of all the organisational changes and individual agent state changes that occur and the reasons for those changes. The Transition Viewer is shown in Figure 5.50, pg. 158, as a part of the example at the end of the chapter. For example, the viewer shows which transition became active to cause a state transition. From the point of view of the actor services, no different information needs to be provided than is provided for The Boss, i.e., both viewers use information about the changing state of the agent organisation but present that information in different ways. In The Boss a snapshot of the current state of the organisation is always available, in the Transition Viewer a history of important changes is shown.

The information in the Transition Viewer allows the user to see the development of the organisation over time. This can in turn help answer questions regarding *why* the system is in the state it is. Intelligent filtering facilities are likely to be useful for larger systems or information overload might occur because so many transitions will be executed and organisational changes made.

Negotiation Visualisation

The Negotiation Viewer tracks the results of the negotiation (see Figure 5.32). At the end of each cycle the viewer adds a point on the graph for each engineer, representing the engineer's satisfaction level with the negotiation. Over time a graph builds up showing how each engineer's satisfaction level varied.

There is a direct correlation between the satisfaction an engineer has with the ongoing negotiations and the success it is having in achieving its goals. In particular, if an engineer is satisfied by the negotiations it will be making progress towards its goals (provided its satisfaction function has been

designed correctly). If an engineer is having success in a negotiation then its progress towards its goals is not being hindered by conflicts with other goals. A manager is not being hindered by goal conflicts if the engineers it has contracted (either directly or indirectly via other contractees) are succeeding at their tasks, i.e., are being satisfied in negotiations.

The negotiation process allows the achievement of different goals to be traded off with one another, e.g., *conserving fuel* and *gaining altitude* might "compromise" to an intermediate altitude which partially achieves both objectives simultaneously. Such tradeoffs are clearly shown in the Negotiation Viewer.

The user can also use the Negotiation Viewer to see how sets of different goals conflict. For example, by noticing that consistently when one engineer was satisfied another was unsatisfied the user can detect a conflict between the engineers (and therefore a conflict between the goals). Perhaps the conflict was obvious a priori, e.g., conflicts between the *conserving fuel* and *rapidly gaining altitude* goals are inevitable, but in more complex cases, more subtle conflicts can manifest themselves in unexpected ways. In other cases, the consequences are major, e.g., in TACSI having a ground avoidance and aircraft avoidance in unresolvable conflict is a major problem.

Conflicts are an inevitable part of a complex task, the key is to find unexpected, problematic conflicts that need to be resolved. The aim is to have the right goal being satisfied by the negotiation process at the right time.

Notice, the *importance* of a conflict needs to be determined by the user because the actor has no knowledge of the importance of the conflict. For example, the actor does not "know" that a conflict between *conserving fuel* and *gaining altitude* is inevitable. This is a good example of how a violation of one of the guidelines, i.e., the Design Information Guideline, by not representing which conflicts are important and which are not, leads to limitations in what $\mathcal{I}$ can do. For example, if we had followed the guideline the actor *would* have a representation of which conflicts were important and which were not, so $\mathcal{I}$ could provide information about only the important conflicts, instead of them all.

The actor does have some implicit knowledge of the importance of goal conflicts, via the environmental priority functions. Priority reflects the importance of the agent given the current situation. If two agents are conflicting when both their priorities are high it is likely to be more of a problem than if their priorities are low (or even if one of their priorities is low). Notice, this sort of implicit information is more difficult to extract and less reliable than explicit designer information. However, it is easy to imagine

a more intelligent visualisation that uses simple reasoning based on satisfaction levels and priority levels to highlight conflicts that are likely to be important.

If it appears the circumstances should allow achievement of a goal the Negotiation Viewer provides some of the information necessary to understand why it is not being achieved. If the agent responsible for a goal is satisfied with the negotiation outcome but the goal is not being achieved then there is probably an error in the agent's satisfaction function. Alternatively, if the agent is not satisfied with the negotiation outcome then it is likely that another, higher priority, engineer is responsible for a conflicting goal.

That such an informative visualisation of the goal conflict resolution can be easily built is due to the explicit, human interpretable nature of the negotiation mechanism. In a negotiation each engineer summarises its *point of view* on the actor output with a single number. The relative size of these numbers for different engineers gives over time all the information described above when interpreted in a straightforward manner.

This Negotiation Visualisation has a clear deficiency – only the agents' satisfaction levels for the actual actions taken by the actor are shown. It would be useful to see the satisfaction levels for *rejected* actions, as such information may be useful for determining why *other* actions were not taken. This is potentially as useful as knowing why the chosen action was taken. For example, understanding why a simulated pilot *banked* instead of *diving* to avoid an incoming missile might be shown by looking at the satisfaction information for the rejected *diving* action. This is purely a deficiency with the current interface – the services provide information about losing bids but the interface ignores that information. The reason why the information is not shown is that it is difficult to create an appropriate visualisation of the data.

Another deficiency of the visualisation is that only the effects of engineers are shown so the user is required to calculate the effect of a manager (if such information is required). Since there is a straightforward relationship between managers and negotiations it would be straightforward to add a visualisation of a manager's impact on the negotiation.

Priority Viewer

The Priority Viewer is a simple viewer that shows the engineers' changing priority levels over time. The viewer uses the same visualisation techniques as the Negotiation Viewer. This information supplements information

supplied by The Boss, and compliments information from the Negotiation Viewer. If the Negotiation Viewer shows that an engineer is not being satisfied in negotiations, the priority viewer can be used to check whether priority is the reason for the lack of success. That is, the Priority Viewer can be used to identify the higher priority engineer with conflicting preferences that is "winning" the negotiation.

Potential goal conflicts may also to be discovered with the Priority Viewer. Agents' goals may not currently conflict but may conflict later, in which case the higher priority agent will be satisfied at the expense of the lower priority agent. The user can check which engineer will succeed, if a conflict occurs, and consider the consequences of this (potentially forcing a change) before the situation occurs. Notice, that such a check is not completely safe because the environmental component of priority may change.

Implications Visualisation

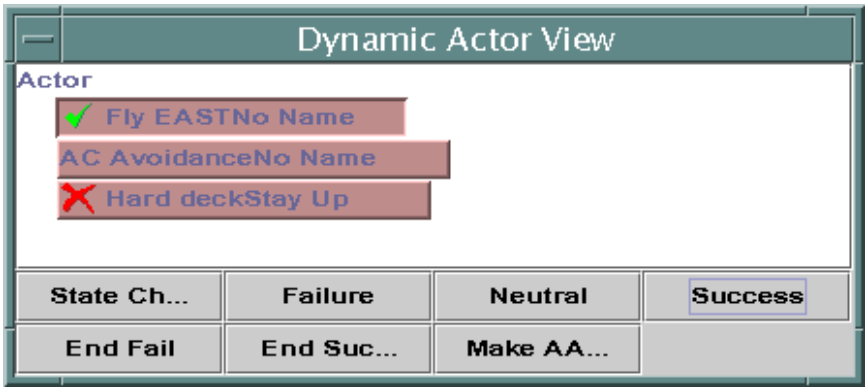

Fig. 5.20: *Implications Visualisation tool for investigating the implications of different actions on the agent organisation. The buttons along the bottom of the window allow the user to specify different changes to the organisation.*

The Implications Visualisation allows the user to experiment with the organisational hierarchy of the actor without the effects actually occurring in the actor (see Figure 5.20). The Implications Visualisation provides a sort of "intelligent blackboard" with which the user can experiment. This visualisation allows the user to make a limited check (excluding the negotiation outcome) on the implications of their actions before they actually take

them. The user is presented with a view of the hierarchy and can view simulations of the effects of different changes they might wish to make on the agent organisation. The user can also see the effects of the success or failure of different agents and of different state changes occurring in the agents which can be used to see what will happen if environmental circumstances or conflicts lead to failure or success.

The information provided by the Implications Viewer supplements the user's mental model of the organisation. When working with a very complex actor a user may not be capable of completely understanding the effects of a particular action they intend to take without the Implications Viewer to assist.

The Implications Viewer is possible *because* the functioning of the agent organisation is deterministic. The user can be shown exactly what will happen inside the actor when a change is made because the outcome of that change is certain.

This visualisation does not show (potential) changes to negotiation status, only changes to the agent organisation. Showing potential negotiation outcomes is not possible because the negotiation is non-deterministic. (Neither does the visualisation take into account potential environmental changes.) This is an example of where breaking our guidelines, i.e., the Deterministic Execution Guideline, makes it difficult to develop AA. If the negotiation process was deterministic then the Implications Viewer could have shown the outcome of the negotiation after user changes. Not showing the outcome of the negotiation restricts the user's ability to assess the effect of their AA decisions before they commit to them.

Calculation Details Display

All the calculations to be performed by each of the agents in the actor can be specified in the spreadsheet-like function specification tool included with EASE (see Figure 5.13). If the spreadsheet facility is used to specify a calculation then at run-time the details of the calculation can be displayed for the user, as shown in Figure 5.21. Such information is primarily useful for debugging calculations. For example, close examination of an agent calculation may reveal that the agent is making correct calculations but sensors are inadequate to provide correct information.

This detailed display is possible because the spreadsheet calculation is "compiled" into a tree structure that executes the calculation. To display a calculation the tree is traversed and intermediate results displayed to the user. Thus, the *service*, which is simple because of the EASE design, is to

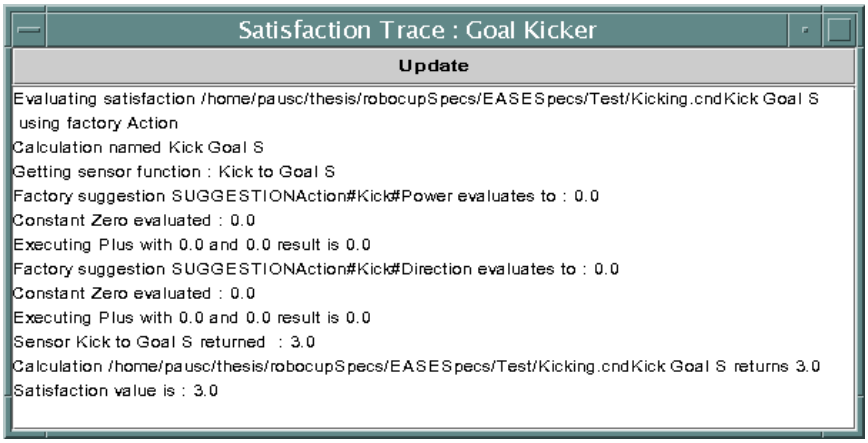

Fig. 5.21: *The Calculation Trace window shows the details of a calculation one of the agents is performing. Each line shows the result of computing the value of one cell. The names used to describe the cells are used to name the intermediate results. The bottom line in the window shows that the satisfaction calculated was 3.0.*

traverse the tree and provide intermediate results.

The compilation process is an example of retaining design information explicitly, i.e., adherence to the Design Information Guideline, so it can be used for AA. It turns out to be very useful for understanding the details of the actor's behaviour. Notice that both the structure of the calculation and the names of intermediate calculations are retained in the agent, hence they can be displayed in the viewer. A more "standard" compilation process (e.g., to binary code) would render much design information unusable at run-time. This happens when the spreadsheet tool is "avoided" by coding calculations directly in a low-level Java program. However, we still often specify expressions in Java because it results in a more efficient actor. This is a good example of a trade-off between designing for AA services and meeting other constraints.

5.6.2 Actuation Services

The $\mathcal{A}$ services allow the following actions to be taken:

- Add agents to the organisation
- Pause agents in the organisation

EASE service	Effect	Tool
Agent organisation change		
Add agent to the organisation	Add goal to be immediately perused	The Boss
Suspend agents in the organisation	Suspend pursuit of goal	The Boss
Remove agents from the organisation	Stop pursuit of goal	The Boss
Generic pool change		
Add agents to the generic pool	Add capabilities to actor	Generic Pool Viewer
Remove agents from the generic pool	Remove capabilities from actor	Generic Pool Viewer
Change the intrinsic priority of agents	Increase importance of goal in overall actor behaviour	Generic Pool Viewer
Change constants	Dependant on the use of the constant	Constant Viewer

Tab. 5.9: *The table maps the different actuation services offered by an EASE actor to the resulting changes in behaviour.*

- Remove agents from the organisation
- Add agents to the generic pool
- Remove agents from the generic pool
- Change the intrinsic priority of agents
- Change named constants

The $\mathcal{A}$ interfaces provide the user with the controls needed to manipulate the behaviour of the actor. Once the user has decided on an appropriate change in behaviour, they need to issue the instructions to ensure the appropriate change occurs. The $\mathcal{A}$ interfaces build on $\mathcal{A}$ services which in turn make actual changes to the actor's internal data structures. The $\mathcal{A}$ services are of two types: those that deal directly with the current agent organisation and therefore the current actor behaviour; and those that deal

indirectly with the actor behaviour by manipulating the generic agent pool. Manipulation of the agent organisation produces immediate, but transient effects on the actor behaviour, i.e., they only last until the contract with the affected agent is ended. Manipulation of the generic agent pool does not necessarily have immediate effects on the actor's behaviour, because the generic agents are not doing anything, but the effects of such changes are persistent, i.e., the effects on the actor's behaviour occur every time the changed generic agent is contracted.

$\mathcal{A}$ interfaces are easy to build because the EASE architecture provides a solid base. Interfaces that allow flexible, powerful manipulation of actor behaviour are possible largely because the agent organisation is represented explicitly at run-time in the agent. The agent organisation represents the goal hierarchy explicitly making the actor's goals easy to manipulate. If the organisation was "compiled" out at design time it would be significantly more difficult to realize the changes because it would be harder to map the required changes to the internal data structures.

The social conventions obeyed by the organisation mean that the actor can often handle the details of user changes, allowing the user to focus only on the specific aspects that interest them. The idea is that the details and inter-relationships involved when changing a single agent will be looked after by the social conventions of the organisation. The relative independence of the agents from each other (they are related only via contracts and negotiations) is the key factor in allowing the agents (and hence the goals) to be easily added and removed from the actor. So, when an agent is removed from the actor, the other agents in the actor (often) do not need to be altered by the user, though changes managed by the system may occur because of the changed situation, e.g., another agent may make a state transition to deal with the removed agent.

This section describes prototype $\mathcal{A}$ interfaces that have been built and the details of the services underlying them. The descriptions of the interfaces below are only brief, providing just enough information to show the use of the service. By showing how each of the agent features are used by different interfaces to provide different controls, we illustrate why the EASE architecture is an appropriate one for building actors for AA systems.

Manipulating the Agent Organisation

The most commonly used $\mathcal{A}$ services allow a user to add, remove or suspend agents (or agent hierarchies) in the organisation of a running actor. Those services correspond to the user facilities of being able to start, stop

or suspend the pursuit of goals. The mechanism works effectively because of the independence of, and teamwork between, the agents, which means (fairly) arbitrary changes to the agent organisation can be made without undue disruption to its overall smooth operation.

User manipulation of the agent organisation is done via The Boss (Figure 5.19). To add an agent to the organisation the user clicks "Create contract", then uses a dialog box to select which agent to add and instantiate any parameters of the agent's contract. To remove an agent from the organisation the user selects the agent in The Boss and clicks the "End Success" or "End Failure" button. "End Success" means that before finishing the agent will report that it has succeeded to its contractor, while "End Failure" means a failure message is sent to the contractor. The user can suspend an agent in a similar fashion. We discuss each of these actions in more detail below.

Adding agents

When adding agents to the agent organisation, the user is adding goals for the actor to pursue, i.e., conceptually each agent added to the actor corresponds to a goal being added to G_a. The goals/agents that are added are handled in the same way as any other goals of the actor. It is possible that a whole hierarchy of agents is eventually contracted to achieve the added goal, e.g., if the user adds a manager to the organisation the manager will contract other agents in the normal manner.

In the usual case a newly added agent, or its contractees, will result in some change in actor behaviour due to its involvement in negotiations. For example, if an agent for flying at some particular altitude is added to the agent organisation of a simulated pilot, the aircraft might climb (or dive) to that altitude while continuing to pursue whatever other goals it has. However, the addition of new goals will not always be so smooth.

If the added agent's priority is low and there are agents with higher priority and conflicting goals, the addition of the agent may have no effect on the actions of the actor because the higher priority agents "win" in the negotiation.

Alternatively, the effect of adding an agent may be subtle, with a small, qualitative change in the actor's behaviour because higher priority, existing agents can be satisfied by a different course of action than they are currently taking that also satisfies the newly added agent. That is, the higher priority agents accept less satisfactory actions because they allow both themselves and the added agent to be at least partially satisfied. For example, adding a fuel conservation agent (which dislikes fast turns) to the organisation of

a simulated aircraft actor might result in waypoints being rounded more smoothly than before. During a turn the agent responsible for getting to a waypoint will be less satisfied (because it prefers to fly directly at the waypoint) but both the waypoint and fuel conservation agents will at least be partially satisfied.

If an added agent (or its contractees) has sufficiently high priority so that it has a significant say in negotiations it is possible that the achievement of other goals already in the system are detrimentally effected. That is, if goals conflict, the lower priority ones (that were already in the hierarchy) may be unsatisfied by the negotiations to the newly added agents. The result is that agents that were previously satisfied may now be failing because of conflicts with (higher priority) added agent(s). For example, originally a low priority agent for making a simulated pilot actor keep a low profile might be satisfied by staying at a low altitude, but when another, higher priority agent is introduced requiring that the aircraft move to a higher altitude the agent for keeping a low profile might fail.

Removing Agents

The removal of agents from the actor has the opposite effect of adding them (naturally). The goal that the removed agent was pursuing is removed from the set of goals the actor is pursuing, i.e., it is removed from G_a. Analogous to the effect of adding goals, the removal of goals could potentially allow previously failing goals to succeed, i.e., the removal of some engineers from a negotiation may allow other (previously lower priority) engineers to succeed where they were previously failing.

If the user removes a manager, the manager's contractees are also removed, i.e., removing a goal means that sub-goals are also removed. So, the set of goals that are removed, R, when g is removed by the user is $R = \{r : r \in G_a, (parent(g, r) \vee (parent(x, r), x \in R))\}$.

If the removed agent is contracted by another agent (i.e., if the agent has goal $g, \neg is\_top\_level\_goal(g)$) it is necessary for the ongoing smooth functioning of the actor that before finishing the removed agent informs its contractor that it will no longer be pursuing its assigned goal. The question is what the contractor should do in such a situation. In keeping with the No Extra Mechanisms Guideline, we reuse the built in social conventions of the organisation for this purpose. The agent may send either a success or failure message to its contractor before terminating activity. The user decides which message is sent depending on how it wants the contractor to react to the removal of the agent. In both the success and failure cases the

messages that are sent are the same as would have been sent if the agent detected its own success or failure (rather than being removed and having the success of failure information provided by the user). The two options give the user the ability to provide *semantic information* to the contractor which it can use in its decision making. For example, if the agent's goal had become irrelevant the user might specify that a success message is sent so that the contractor moves onto the next stage of its task.

The contractor agent uses the success/failure information to guide its future actions. Clearly, *success* messages indicate that the contractor should act assuming that the goal assigned to the (now defunct) contractee has been achieved. Conversely, failure messages imply that the contractor should assume that the contractee has failed in their assignment and act accordingly. Precisely what the contractor will actually do will have been specified by the designer via the use of success and failure transitions in the contractor's state machine.

Suspending agents

As an additional aid to users, agents can be *suspended*. While suspended agents have no effect on the output of the actor. All contractees of a suspended agent are also suspended. The paused agents can be restarted again in the same state they were in when they were paused. We will not discuss this facility in detail because it is functionally equivalent to removing then adding the same agent to the actor.

Modifying the Generic Pool

The second group of services an EASE actor provides are those for modifying the generic pool of agents available to the actor. These services allow manipulation of the authority of the actor. They are possible because the agents which can be contracted for specific goals are explicitly represented and because the process of finding capable agents is a flexible, run-time process. As well as changing which agents are available to be contracted the user can modify other properties of an agent, including its intrinsic priority and constants used in its calculations.

The generic agent pool contains the agents which can be contracted to pursue their specific capability. When dynamic contracting is used the capability matcher looks in the generic pool for agents with the capability to pursue the goals they need achieved. (When fixed contracting is used, the agent that will fill the contract is known at design time.) The existence or

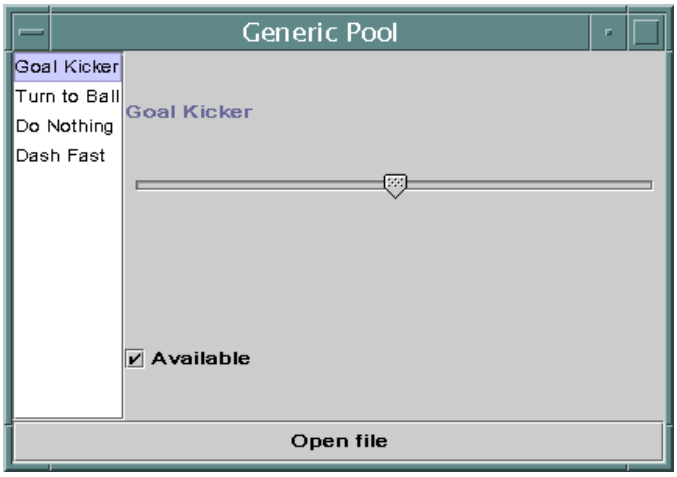

Fig. 5.22: *The tool for modifying agents in the generic pool. Unchecking the "Available" checkbox would mean the "Goal Kicker" agent could not be contracted. The list on the left shows all the agents in the generic pool. The slider in the middle allows the intrinsic priority of "Goal Kicker" to be changed.*

otherwise of agents with the required capabilities affects whether contractor agents can achieve their goals and the details of the contracted agents dictate precisely how the goal is achieved.

To add agents to the generic pool the user opens the file containing the agents to be added. Adding agents to the generic pool gives the actor the authority to pursue the goals the added agents are capable of. Thus, when the achievement of the goals is deemed useful they can be pursued. For example, if the user adds an agent for *dummy move* (i.e., the player pretends to do something to confuse its opponent), a RoboCup actor, previously without that authority could now call on it when appropriate. Obviously, an agent already in the organisation will need to attempt to contract the agent with this capability for it to be useful. In other words, adding an agent to the generic pool only allows the actor to contract it, it does not necessarily mean the actor *will* attempt to pursue the goal by contracting the agent.

To remove agents from the generic pool the user selects, via a dialog box, the agents from the generic pool that should be removed. The agent is marked "unavailable" so it can be easily reactivated later. When the user

removes agents from the generic pool they remove the possibility for any agents in the organisation to contract those agents, potentially removing the authority of the actor to pursue the goals pursued by the removed agents. For example, if the user removes the only agent in the generic pool with a capability to *dribble*, a RoboCup actor will no longer be able to *dribble*.

In terms of the definitions in Chapter 2, adding generic agents to the generic pool gives the actor the authority to pursue the added goals. Hence, added agents correspond to goals being removed from C_a. (Recall C_a is the set of goals the agent may not pursue.) The opposite is true for the case of removing agents from the generic pool, i.e., it corresponds to goals being added to C_a. An alternative way of viewing this might be to think of adding or removing agents as changing the actor's *ability* to pursue some goal. However, we prefer the authority view because we believe it is non-intuitive to change something's ability at run-time but reasonable to think about changing its authority.

A combination of remove and add allows the user to substitute one agent that will achieve a goal by following one course of action with another agent that will achieve the same goal via a different course of action. For example, there might be an agent in the generic pool with the capability of *dribbling*. That agent might implement, for example, a very cautious form of dribbling (i.e., keeping the ball close, moving slowly). By replacing that agent with one that implements *dribbling* in a very aggressive manner the user can change how the player dribbles – every time the player needs to dribble. Making such a replacement allows a user to change the way an actor goes about achieving goals, i.e., effectively changing constraints on the way goals are achieved. Table 5.10 gives an example of this type of exchange in a RoboCup actor. Part way through simulation the user removes the *dribble_normal* agent from the generic pool and replaces it with another agent with the same *dribble* capability but that dribbles faster. Subsequently, a *striker* agent contracts the new *dribble_fast* agent.

Furthermore, the ability to exchange agents also opens up a neat mechanism for debugging and fixing long running simulations. Agents the are not functioning correctly can be removed and replaced with updated versions. Such a capability might be very useful for some applications (Maes 1994*b*).

Intrinsic Priority The user can adjust the intrinsic priority of an agent using a "slider" in the generic pool manipulation tool (see Figure 5.22). This functionality is built on top of the ability to get and set the explicitly represented priorities of agents. Increasing the intrinsic priority of an agent

Event	Striker State	Contract	Generic Pool
	Waiting	watch_ball	{dribble_normal, ... }
Ball comes close	Chasing	get_to_ball	{dribble_normal, ... }
Get to ball	Dribbling	dribble_normal	{dribble_normal, ... }
Opponent near	Pass	pass	{dribble_normal, ... }
Pass succeeds	Return to position	go_to_position	{dribble_normal, ... }
At position	Waiting	watch_ball	{dribble_normal, ... }
User changes "dribble_normal" for "dribble_fast"	Waiting	watch_ball	{dribble_fast, ... }
Ball comes close	Chasing	get_to_ball	{dribble_fast, ... }
Get to ball	Dribbling	dribble_fast	{dribble_fast, ... }
Opponent near	Pass	pass	{dribble_fast, ... }
⋮			

Tab. 5.10: *An example of how exchanging one agent with another with the same capability can affect a RoboCup actor's behaviour. The first column lists the events that triggered changes. The second column lists the state of some striker agent responsible for trying to make the player kick goals. The third column lists the contracts the striker agent has. The final column lists part of the contents of the generic pool.*

increases the importance of the goal the agent is pursuing. Decreasing the intrinsic priority has the inverse effect. Notice that the effects of increasing priority applies to both engineers and managers. The organisational priority of contractees will increase (decrease) when the intrinsic priority of their contractor is increased (decreased). Hence, the influence of the manager in negotiations is changed by changing its intrinsic priority. The effect of changing the intrinsic priority of an engineer is directly on its influence in negotiations.

Conceptually, changing an agent's intrinsic priority allows the user to dynamically change the relative importance of different goals. One case where this would be important is when there are conflicting goals (i.e., agents in the negotiation that cannot agree) and the wrong agent is "winning" the negotiation. A change in priority can change the "winner" of the negotiation by giving agents more or less say in the negotiation.

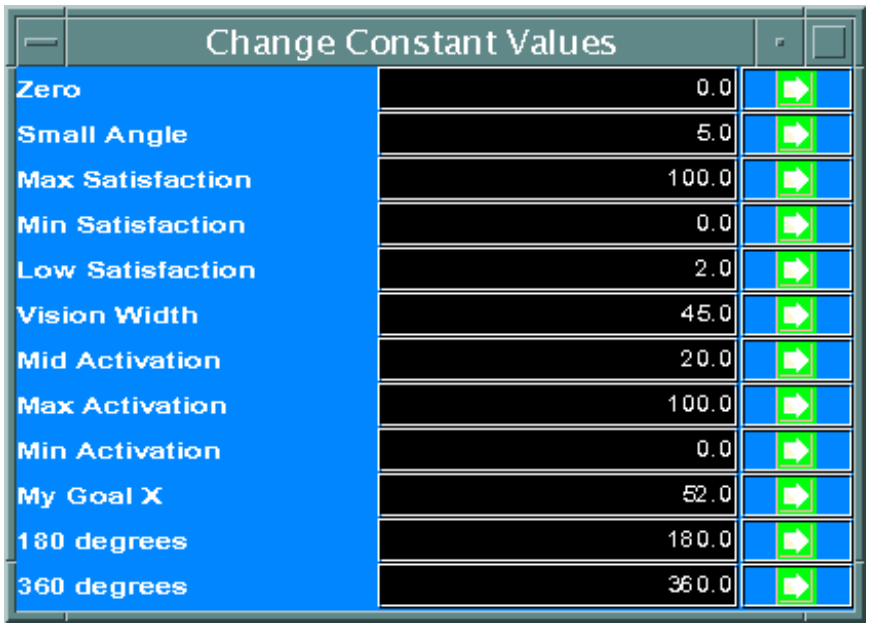

Fig. 5.23: *The tool for changing the value of named constants while an actor is running. The name is on the left, the current value in the middle and a button for making the changes on the right.*

Changing constants Another run-time tool allows the user to change the value of named constants that the actor is using in calculations (see Figure 5.23). This facility is built over a simple service that allows getting and setting the values of explicitly represented constants. Named constants can be used in any agent function, e.g., satisfaction functions, state transition conditions, environmental priority functions and contract parameters. If the actor has been well designed, named constants will capture abstract aspects of the actor's behaviour. Hence, the ability to change those constants opens up a wide range of control to the user. Precisely what effects changes to constants have on the behaviour of the actor will depend on exactly what the constant represents and how it has been used in various agent functions. For example, a well designed RoboCup player might have a constant representing *desperation*. The constant could be used, for example, as a parameter in environmental priority functions for *shooting for goal* and *dribbling*, e.g., the higher the *desperation* the more likely the actor is to shoot for goal in a risky position and go more directly towards goal when dribbling. Hence, changing the constant changes how the actor behaves. Thus, the constant captures an abstract characteristic of the actor's behaviour which can be changed at run-time.

Changing constants is functionally similar to the "control panel" ideas in HCSMs (Cremer et al. 1995*a*) (see Chapter 3) and hence it suffers from the same limitation, i.e., only those constants explicitly designed in at design time can be changed at run-time. For example, if the designer has used no *desperation* constant the user cannot use the constant to change that aspect of the behaviour (although, in EASE unlike HCSMs, there may be other ways to achieve the same change).

5.7 Example

In this section we present an extended example of an EASE actor for the simulated air-combat domain using Saab's tactical air-combat simulator (Saab 1998). The aim of the example is to illustrate the breadth and use of the various AA interfaces, so as to show that a useful AA system has been implemented. The actor's behaviour is fairly simple so we can focus more easily on the AA aspects of the example.

5.7.1 The Scenario

Figure 5.24 shows the starting positions of the aircraft in the scenario. An EASE actor is controlling the aircraft labeled with a "1" in the upper left

Fig. 5.24: *A snapshot of the TACSI visualisation at the start of the scenario.*

part of the screen. The other aircraft will fly from right to left of screen at a fixed altitude (they are not controlled by EASE actors). The scenario takes place over the water just off the east coast of Sweden. When the scenario begins all the aircraft are in the air (at the same altitude) and fully fueled.

In the example, the actor has control over the altitude, heading and speed of the aircraft. Low level routines built into the TACSI simulator perform the very low level details of keeping the aircraft in the air. For example, built-in TACSI routines might engage the after-burners when the aircraft does a tight turn. Notice also that the dynamics model built into TACSI restricts what the aircraft can do. For example, even if the actor asks for an immediate 180 degree turn the aircraft cannot, and will not, perform that turn instead it will turn as quickly as the flight dynamics allow.

5.7.2 Actor Specification

Fig. 5.25: *A snapshot of the "Start control" tool from which the actor and other tools are started.*

Figure 5.25 shows the "Start control". This is the central tool for an EASE actor. The EASE actor is started and stopped with this control. (The "Start" button has changed to "Stop" since the actor was started.) Other tools are launched by clicking buttons in the window. The name of the actor specification being used is displayed in the text field at the bottom of the window. In this case the specification is named *Example.act*.

The user clicks on the "Watch Agents" button to launch The Boss. Figure 5.26 shows the initial agent organisation, consisting of three agent hierarchies. At the top is the *AC Avoidance* list manager. This manager will contract agents to avoid each aircraft the actor detects.

Second from the top is an engineer called *Hard deck* which is responsible for ensuring that the aircraft stays above a certain altitude. The *Hard deck* agent has all those states where the aircraft is above a fixed altitude as its ideal states. Hence, in negotiations, the *Hard deck* agent will be fully

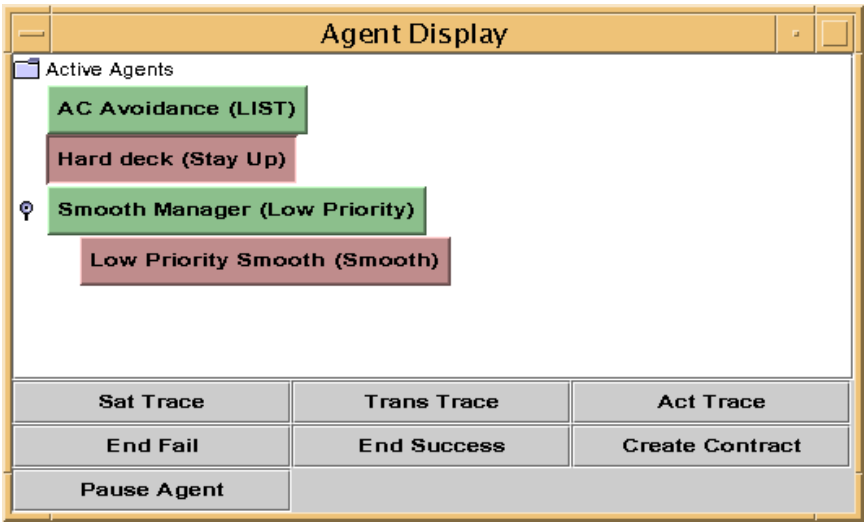

Fig. 5.26: *The Boss showing the starting agent organisation of the actor.*

satisfied with actions that lead to the aircraft staying above the minimum altitude and completely dis-satisfied with actions that lead to the aircraft going below the minimum altitude. The *Hard deck* agent's intrinsic priority is high and its environmental priority function returns a medium, constant value.

The final hierarchy, i.e., the one headed by the *Smooth Manager*, is for maintaining a "smooth" line for the aircraft. This hierarchy is responsible for ensuring turns are not too tight and the aircraft does not try to change speed or altitude too quickly. This hierarchy requests much smoother changes than are enforced by the low-level aircraft dynamics routines in TACSI.

Maintaining a smooth line conserves fuel, which is a good thing but not a high priority, unless the aircraft starts running low on fuel in which case it becomes a high priority. This is modeled by having the *Smooth Manager* transition between a state where it contracts a low priority smooth agent and one where it contracts a high priority smooth agent, when fuel levels become low. Despite having different priority functions, the contractee agents have the same satisfaction functions, i.e., their ideal states are those where the aircraft is not changing speed, altitude or direction too quickly. Notice, that the same functionality could also have been modelled with a single engineer which transitioned between states when the fuel level dropped or with a single engineer with a single state but with an environmental priority

function that depended on the amount of fuel remaining. (The design was chosen for the purposes of this example.)

The initial agent organisation configuration does not have the actor actually *doing* anything, i.e., the agents only prevent the actor from doing "bad" things but none give it a "purpose". It is basically a "generic" pilot specification that needs to be given a mission. Figure 5.27 shows the user selecting a *Patrol Mission* agent after clicking on the "Create Contract" button on The Boss. The mission implemented by the *Patrol Mission* agent is to fly to a patrol area about 80km south-south-east of the starting point then fly a triangular patrol pattern. The patrol is flown indefinitely (there are no airports in the simulation so landing is not part of the necessary behaviour of the actor.)

The *Patrol Mission* agent is a manager which contracts parameterized agents to get it to each of the mission's waypoints. The contractees are found dynamically, i.e., the *Patrol Mission* agent requests the capability matcher find an agent capable of getting it to the next way point. The agents for getting to waypoints will send success messages when they reach the waypoint. The *Patrol Mission* agent reacts to the success messages by cancelling the current contract and creating a new contract with a new waypoint agent for the next waypoint.

Figure 5.28 shows The Boss after the *Patrol Mission* agent has been created. Now the actor has something to achieve and after some simple negotiation, the actor turns the aircraft toward the patrol area.

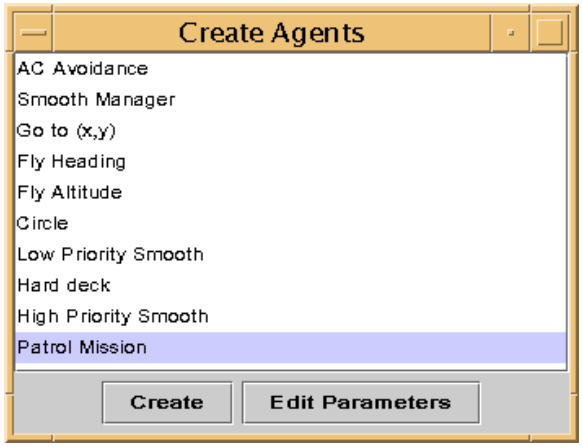

Fig. 5.27: *The user has selected the* Patrol Mission *agent to be added to the agent organisation.*

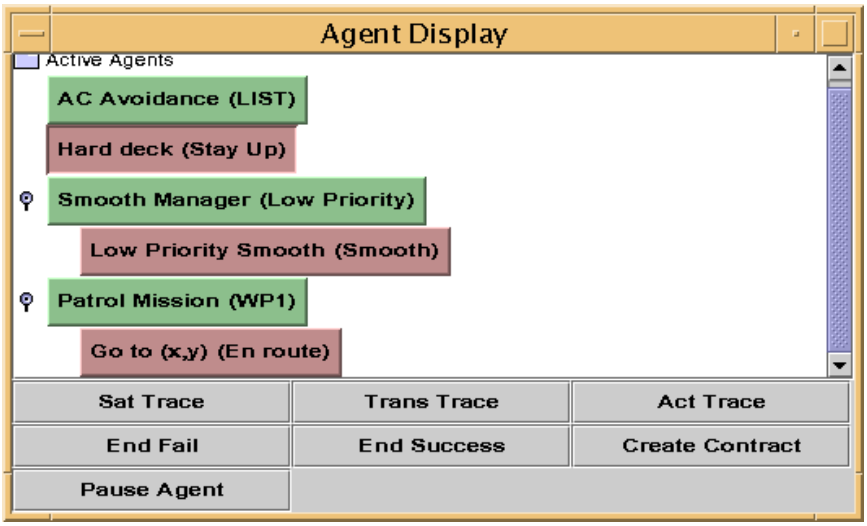

Fig. 5.28: *A snapshot of The Boss after the* Patrol Mission *agent has been contracted.*

5.7.3 Avoiding Aircraft

A short time into the mission, the EASE controlled aircraft detects an aircraft that needs to be avoided (Figure 5.29 shows the situation). The *AC Avoidance* list manager contracts a specific *AC Avoidance* agent for the job. Figures 5.30 and 5.31 show two snapshots of The Boss after the engineer has been contracted. The first figure (Figure 5.30) shows The Boss with the visualisation of the hierarchies fully expanded giving a detailed view of the actor hierarchy, while the second figure (Figure 5.31) shows the hierarchies collapsed giving a more abstract view.

The *AC Avoidance* engineer will be satisfied by actions that lead to either a reasonable "lead angle" to the obstacle aircraft or a reasonable altitude difference. That is, the ideal states for the agent are those where the aircraft is either at a much different altitude to, or flying away from, the obstacle. Such a satisfaction function is very simple, paying no heed to relative directions of motion, upcoming positions, etc.

The negotiation process becomes more interesting when the *AC Avoidance* joins because all engineers can no longer be completely, simultaneously satisfied. (Before the obstacle was detected, the *Smooth*, *Hard Deck* and *Go to (x,y)* engineers could be easily satisfied by flying directly at the waypoint.) Figure 5.32 shows a trace of the satisfaction levels of the agents as

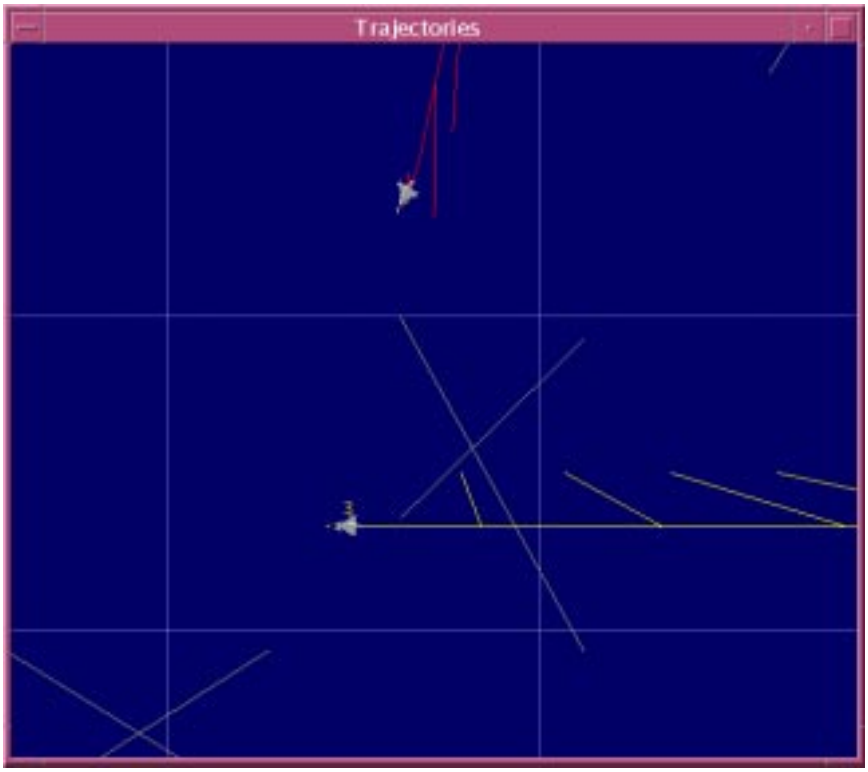

Fig. 5.29: *A snapshot of the TACSI window shortly before the EASE controlled aircraft (labeled "1", at the top) takes evasive actions.*

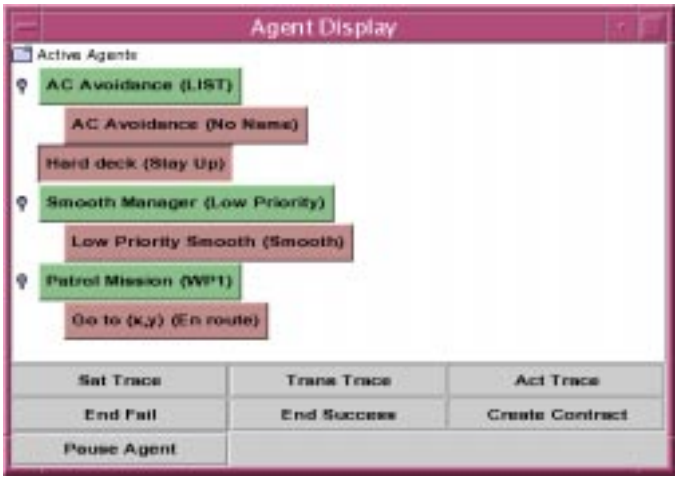

Fig. 5.30: *A snapshot of The Boss, with hierarchies fully expanded, after an aircraft avoidance agent has been contracted to avoid a detected aircraft.*

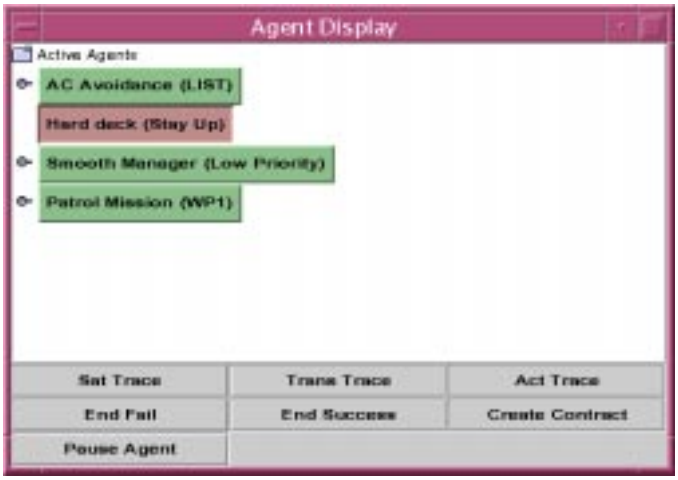

Fig. 5.31: *A snapshot of The Boss, with hierarchies fully collapsed, after an aircraft avoidance agent has been contracted to avoid a detected aircraft.*

the obstacle aircraft is avoided. Each line on the graph shows the satisfaction level of one of the engineers over time. The mapping between colours and lines can be seen in the legend in the top right hand corner. The level of satisfaction that all the agents are at when the snapshot is taken is shown to the right of the graph. When the snapshot was taken all engineers were completely satisfied. The line across the bottom of the graph represents complete dissatisfaction. There are sharp dips in the satisfaction levels of the *Go to (x,y)* and *Smooth* agents as the aircraft makes small turns to ensure the obstacle is safely avoided. The dips do not go all the way to the bottom of the graph, indicating that some compromise was reached so no agent was completely dis-satisfied. Notice that the high priority engineers, *Hard Deck* and *AC Avoidance* are always completely satisfied. Notice also, that the dips of the other agents are very brief (less than one second).

The path taken by the aircraft is shown in Figure 5.33. It shows a fairly smooth curve around the obstacle (which has since moved on). At no time has the aircraft made sudden movements. This is because of the influence of the *Smooth* engineer. Furthermore, a fairly direct path has been maintained towards the waypoint due the influence of the *Go to (x,y)* engineer. The aircraft's path has a distinct impact from the lower priority engineers, i.e., *Go to (x,y)* and *Smooth*, despite the deep dips in their satisfaction levels at different times.

5.7.4 Suspending Agents

So far this example has looked pre-dominantly at the autonomous behaviour of the actor (with the exception of the adding of the *Patrol Mission* agent). The actor could continue to act autonomously but the aim of this example is to show the usage of the AA facilities.

The first thing we will do is show how the effect of a particular agent on the overall actor behaviour can be investigated by suspending the behaviour of that agent. In this case we look at the influence of the *Smooth Manager* on the aircraft's behaviour.

By selecting the *Smooth Manager* and clicking "Pause Agent" the influence of that manager is removed. Figure 5.34 shows a snapshot of The Boss after the *Smooth Manager* has been suspended. Notice that the whole hierarchy has been suspended. Figure 5.35 shows a trace of part of the patrol. The longer, smoother turn at the bottom was made while the *Smooth Manager* was active while the tighter turns at the top were made when the *Smooth Manager* was suspended. In this case the effect of the *Smooth Manager* is evident but not extremely strong. The designer might use this

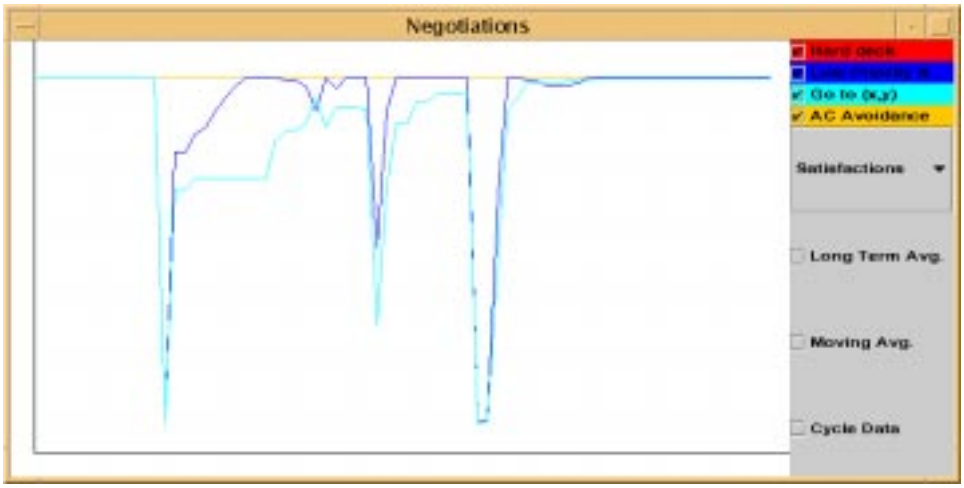

Fig. 5.32: *A snapshot of the Negotiation Viewer shortly after an aircraft was successfully avoided. The graph shows about two minutes worth of negotiation.*

observation to alter the specification later on, e.g., by increasing the priority of the *Smooth Manager* to make turns smoother.

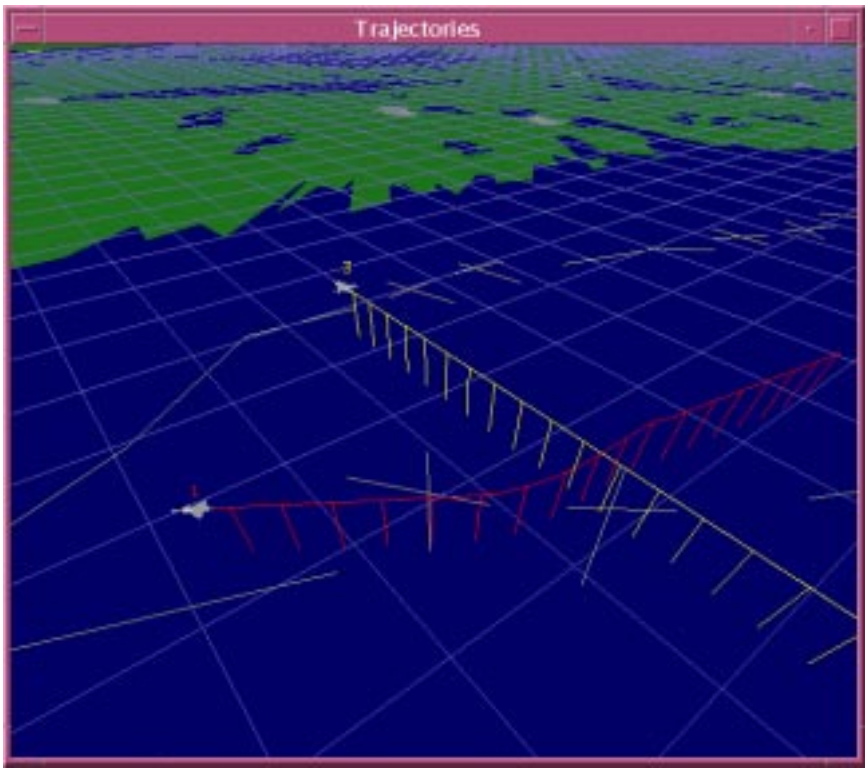

Fig. 5.33: *The scenario shortly after an aircraft was successfully avoided. The path of the EASE controlled aircraft (on left) is shown with a trail behind it. Notice the small curve near where the paths of the two aircraft cross showing the slight turn the aircraft made to avoid the obstacle.*

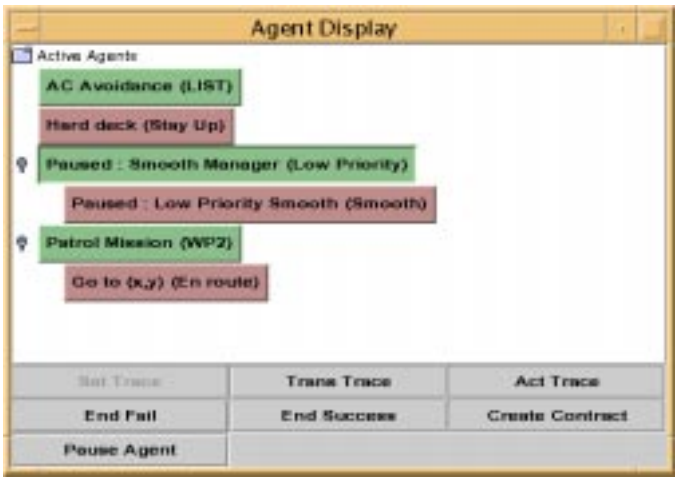

Fig. 5.34: *A snapshot of The Boss with the* Smooth Manager *agent suspended.*

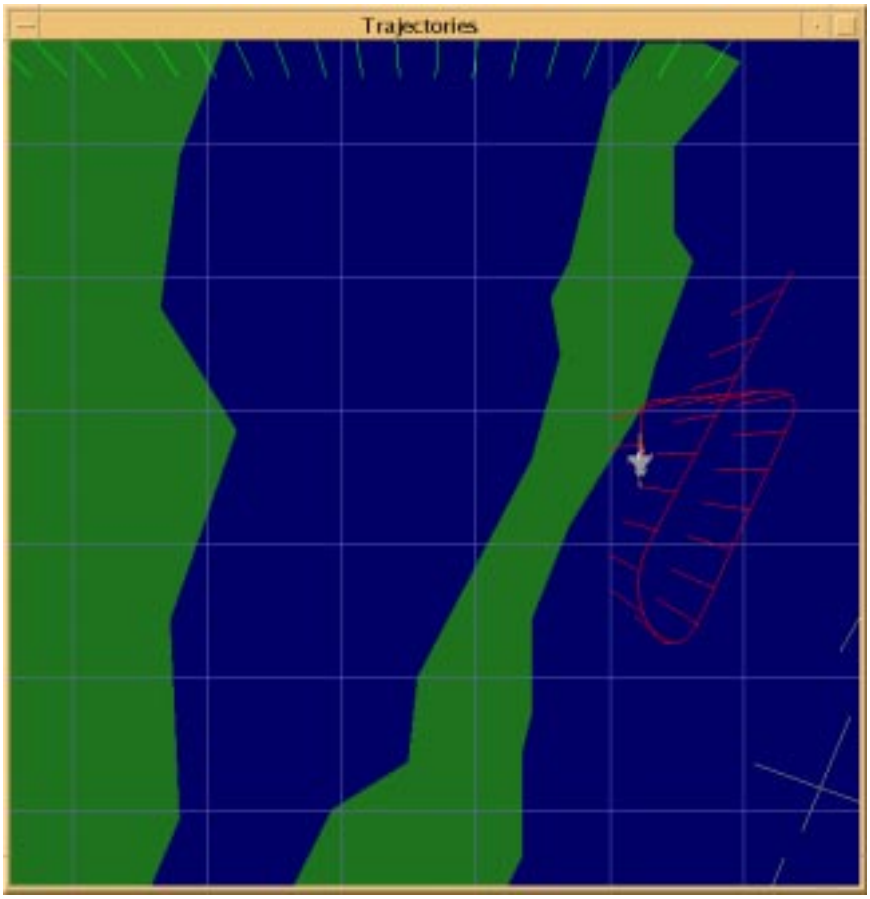

Fig. 5.35: *A trace of the aircraft's path during the patrol. The turn at the bottom (one long turn) was made while the* Smooth Manager *was active and the tighter turn at the top was made while the* Smooth Manager *was paused.*

5.7.5 Changing Generic Pool

Next we show how the generic pool can be manipulated. The Generic Pool Viewer is opened by clicking on the "Generic Pool" button on the Start Control. The starting state of the Generic Pool is shown in Figure 5.36. A second file of agent specifications is opened, using the "Open file" button at the bottom of the viewer, and the Generic Pool changes to the state can be seen in Figure 5.37. Among other things, the second file contains a second agent capable of getting the aircraft to a waypoint, called *Go To (x,y) - Fast*.

Fig. 5.36: *The starting state of the generic pool, showing the ten available agents.*

With two agents in the Generic Pool having the capability of getting to a waypoint, when the Capability Matcher (on request from the *Patrol Mission* agent) looks for an agent with that capability, either will be chosen. To ensure the "fast" agent is chosen, the "normal" one (i.e., the original *Go to (x,y)* agent) is made "unavailable" by clicking on it then unchecking the "Available" checkbox (see Figure 5.38). This effectively removes it from the Generic Pool (although it still appears in the window so it can be easily reinstated.)

Taking the generic agent out of the generic pool does not mean it also comes out of the organisation. Hence, the instance of the "normal" waypoint agent, i.e., *Go to (x,y)*, that is currently contracted by the *Patrol Mission* agent is still getting the aircraft to its current waypoint. However, when the aircraft reaches the waypoint the *Patrol Mission* agent breaks the current contract and, via the Capability Matcher, finds an agent to get it to the next waypoint. The *Go To (x,y) - Fast* is found and contracted. It is actually a

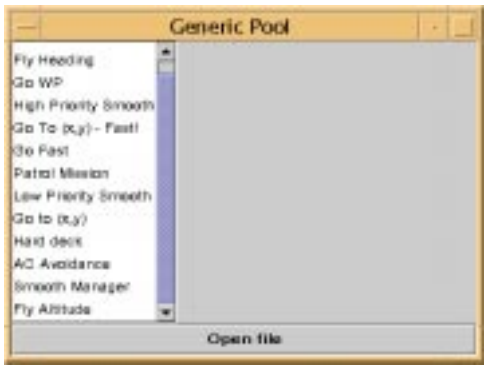

Fig. 5.37: *The state of the generic pool after a second file of agent specifications has been opened. The generic pool now contains 13 agents.*

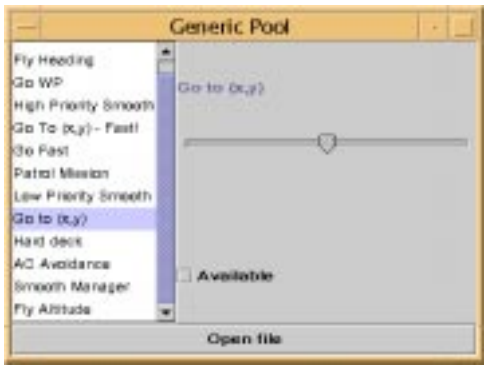

Fig. 5.38: *Making the agent* Go to (x,y) *unavailable to be contracted in the generic pool.*

manager agent and contracts more specific agents for flying towards the new waypoint and increasing speed. Notice, that the "normal" waypoint agent was an engineer and the new *Go To (x,y) - Fast* agent is a manager but this is OK. In fact, the *Patrol Mission* agent does not even know whether it has contracted a manager or an engineer only that it contracted an agent that can get it to a waypoint. Figure 5.39 shows The Boss after the new contracts have been made.

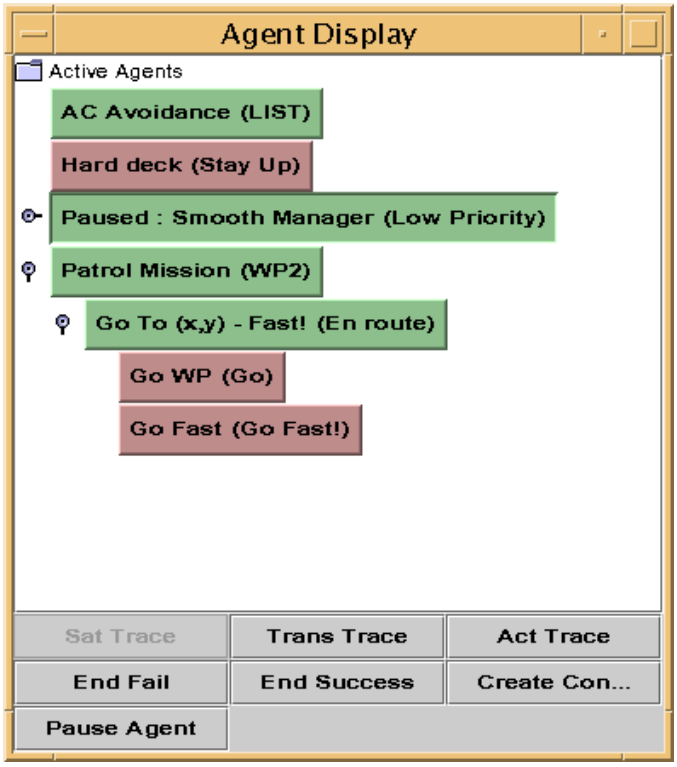

Fig. 5.39: *A snapshot of The Boss with the* Go To (x,y) Fast *manager and its contractees.*

5.7.6 Changing Constants

After the patrol has gone on for some time the fuel level of the aircraft should start to drop and the *Smooth Manager* should make a state transition to a state where being "smooth" is a high priority. However, after some time this transition has not occurred. The fact that the transition has not occurred

possibly indicates an error in the specification.

To investigate, the *Smooth Manager* is selected (after first being reactivated) and the "Trans Trace" (i.e., Transition Trace) button is clicked to see the calculation the manager is doing to decide whether to take the transition. Figure 5.40 shows the trace. The details of the calculation are not particularly clear in this case because the details of the calculation have been implemented in Java code (rather than the Condition Specification spreadsheet). The third line shows the name of the transition condition being checked. The fifth line shows the value (10.0) of the named constant (*Low Fuel Level*) that represents a low fuel value. Even with the limited information, the user can quickly work out that the problem is that *Low Fuel Level* is in the wrong units and should be 1000.0 instead of 10.0.

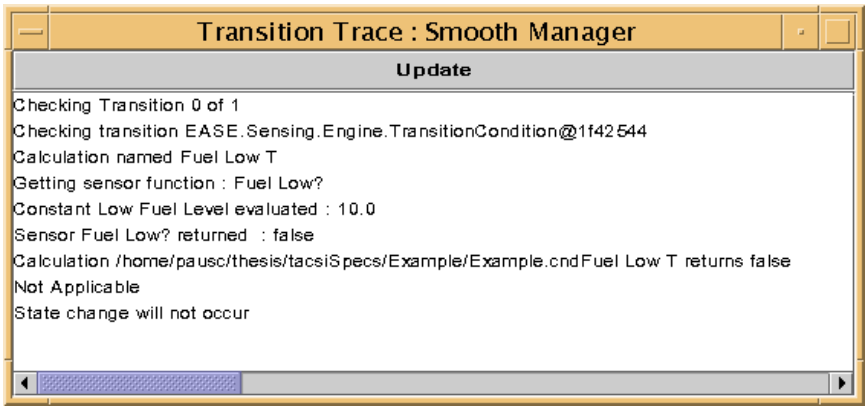

Fig. 5.40: *A trace of the calculations the* Smooth Manager *is doing to determine whether a state transition should be made.*

Fortunately, the *Low Fuel Level* was created as a named constant. The user opens up the Change Constants window (see Figure 5.41) and changes the value of *Low Fuel Level* to a more reasonable number. As soon as the value of the named constant is changed the *Smooth Manager* makes a state transition. Figure 5.42 shows the situation after the state change has been made.

Next, the user changes the value of one of the constants representing a waypoint coordinate. The resulting flight path is shown in Figure 5.43. The y component of the waypoint is changed to be very far north, so the aircraft immediately turns slightly to the left and heads (almost) directly north.

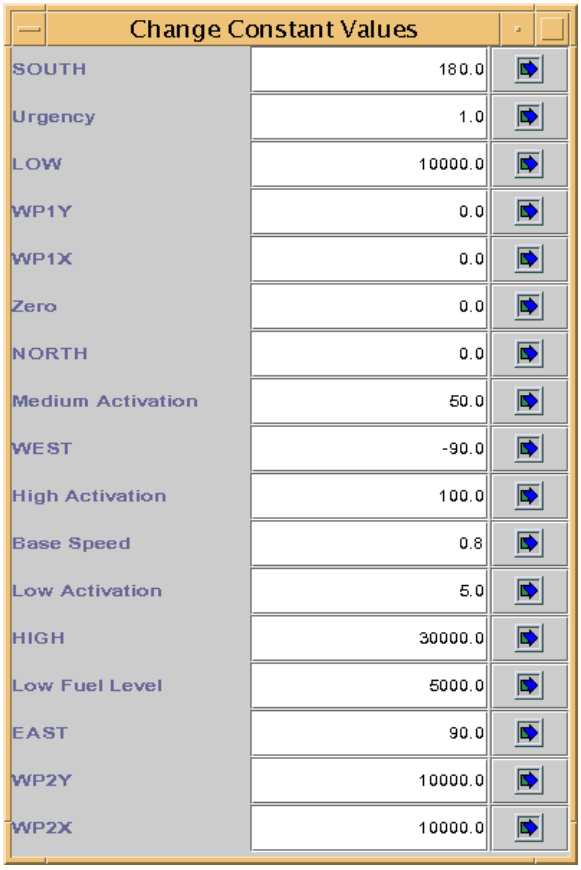

Fig. 5.41: *A snapshot of the tool for viewing and changing the values of named constants.*

Fig. 5.42: *A snapshot of The Boss after the* Smooth Manager *makes a state transition. (Notice that the user has also replaced the* Patrol Mission *agent with a* Circle *agent.)*

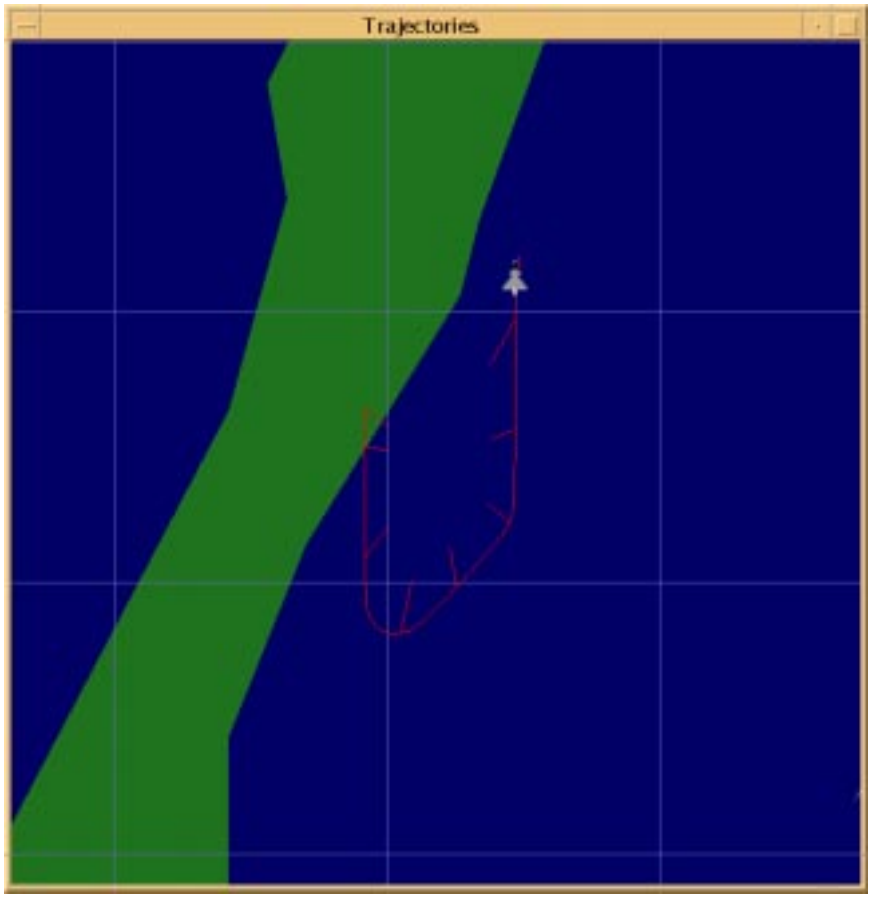

Fig. 5.43: *A trace of the path of the aircraft when the location of one of its waypoints is dynamically changed. The "turn" on the right is actually where the waypoint location was changed to be much further north, rather than north-east.*

5.7.7 User Taking Over High Level Decision Making

If the user wants to take over all the high level decision making of the actor they need to stop the high level decision making of the *Patrol Mission* agent. This is achieved by selecting the agent and clicking on either "End Fail" or "End Success" buttons. In this case it does not matter which button is used since the agent has no contractor to send a message to, anyway. Figure 5.44 shows a snapshot of The Boss after the *Patrol Mission* agent has been stopped.

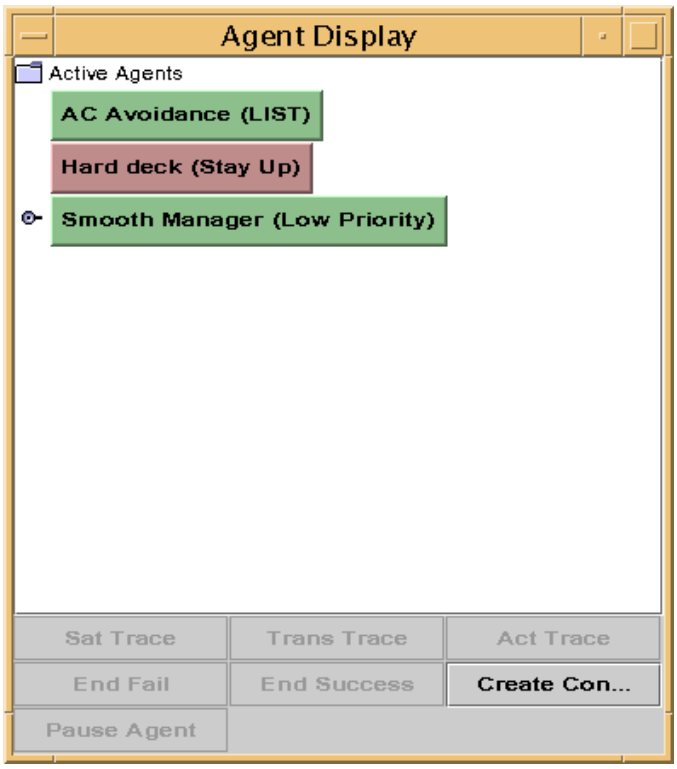

Fig. 5.44: *A snapshot of The Boss after the* Patrol Mission *agent is stopped.*

Next, the user can get the aircraft to follow a particular heading by creating a *Fly Heading* agent. *Fly Heading* is a parameterizable agent, where the parameter is the heading the aircraft should take. Figure 5.45 shows the user creating the agent and instantiating a heading. In this case, the required parameter existed and was quickly instantiated. In other cases, the

parameter might not exist and would need to be created using the Condition Specification tool (Figure 5.46). In the example, the user creates a simple constant, though arbitrarily complex expressions could have been created. In this case the user decides to make the aircraft go directly east (perhaps as the first stage of a defection to the east?). Figure 5.47 shows the resulting organisation in The Boss and Figure 5.48 shows the subsequent path taken by the aircraft.

Notice that, importantly, the user did not need to plan the low level details of the aircraft's behaviour. The *AC Avoidance*, *Hard Deck* and *Smooth* agents kept performing their functionalities so the user could focus only on the aspects of behaviour that interested them. For example, if another aircraft had been detected it would have been avoided by the *AC Avoidance* manager without further involvement from the user.

Fig. 5.45: *Creating a* Fly Heading *agent and instantiating the "Required Heading" parameter.*

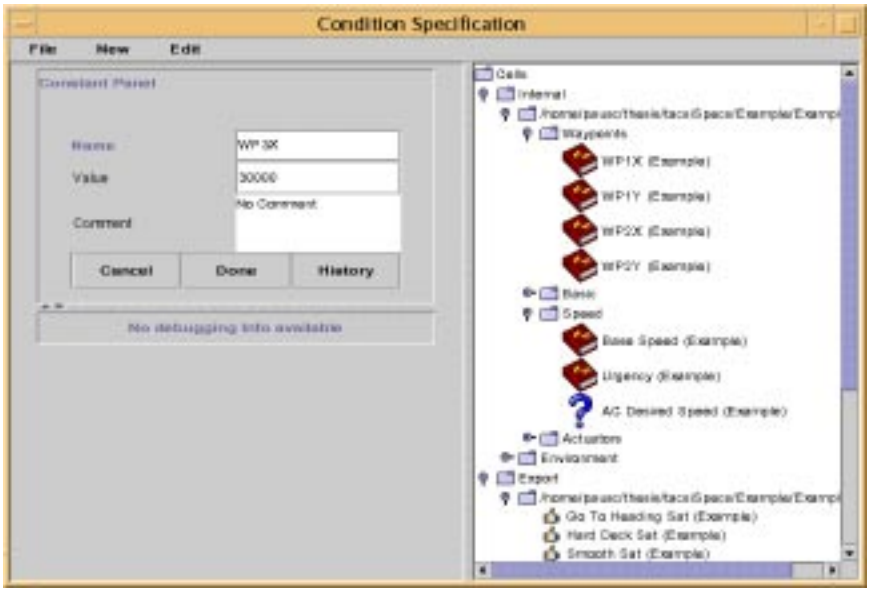

Fig. 5.46: *Condition Specification tool for specifying calculations to be performed by the actor.*

5.7.8 Detail Traces

Finally, we look at the tools that allow a detailed examination of the actor's behaviour.

Figure 5.49 shows in great detail how the *Go to (x,y)* engineer calculates that for a suggested heading of -146 degrees it is 99.8% satisfied. Each line in the calculation shows an intermediate result. For example, the waypoint positions, *WP1X* and *WP1Y*, can be seen to be both 0.0. A "complex" sensor *Head to (x,y)* takes the suggested heading, which is -146, and the required waypoint and returns a satisfaction value, i.e., 99.81. This "complex" sensor has been implemented in Java so the details of that calculation are not shown.

Figure 5.50 shows a small part of the trace of events that occurred in the agent organisation. This particular snapshot shows some state transitions that have occurred in the organisation. Notice that where agent names should appear "???" appears instead. This is because at the level of the state machines (from where this information comes) the agent's name is not known. This is a simple example of how not representing information internal to an actor can cause difficulties for AA (though it is a reasonable

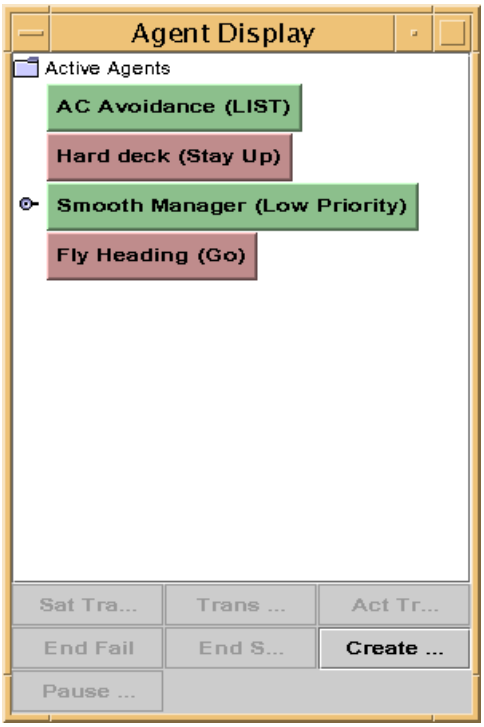

Fig. 5.47: *A snapshot of The Boss after the user adds the* Fly Heading *agent.*

Fig. 5.48: *The route taken by the aircraft after the user stepped in and started making high level decisions. Notice the turn to the east near the left of the screen.*

Fig. 5.49: *A trace of a* Go to (x,y) *engineer's reasoning for its satisfaction with the selected actor action.*

software engineering decision to decouple the agents for their state machines so other decision making mechanisms could be easily added).

The last snapshot in this example, Figure 5.51, shows a trace of the calculation the *Hard Deck* agent performs to calculate its environmental priority. The environmental priority part is a calculation using the named constant "Max", which evaluates to 100. The intrinsic priority component is 50. This agent has no contractor (i.e., parent) so there is no organisational priority component. The overall priority of the agent is calculated as 5000.0.

Summary of Example The example has illustrated some of the AA functionality in a simple patrol mission for a simulated pilot. We showed agents being added, removed, suspended and changed at run-time. The example aimed to show the functionality of some of the AA facilities rather than showing how they would be used in a real situation.

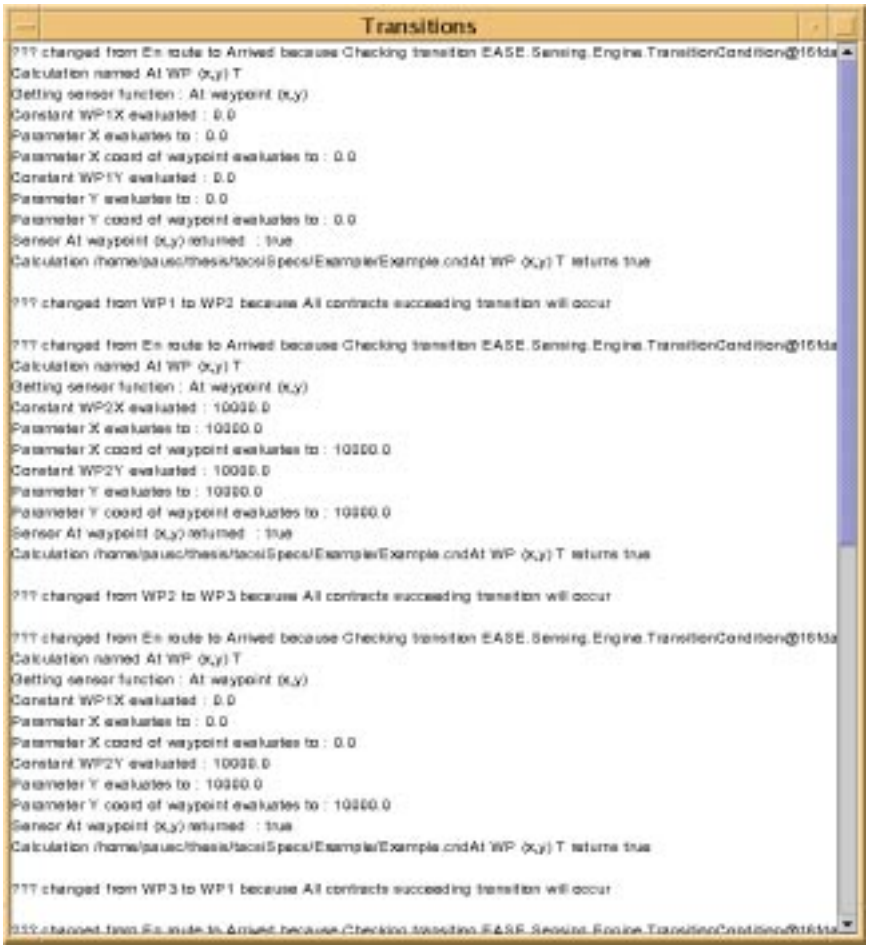

Fig. 5.50: *A trace of some of the changes in the agent organisation and the reason for those changes.*

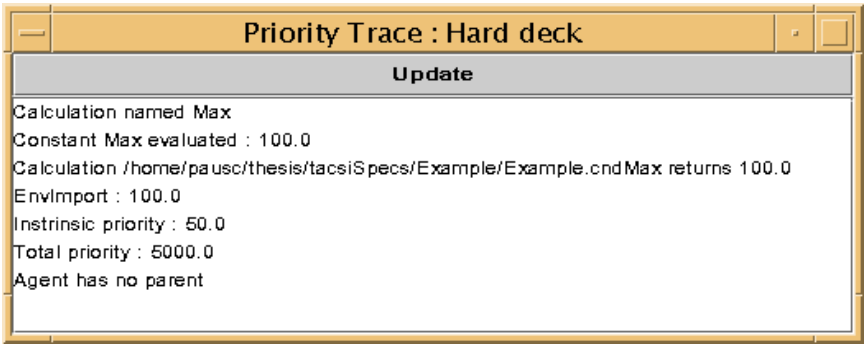

Fig. 5.51: *A trace of a* Hard Deck *engineer's reasoning for its priority.*

5.8 Limitations

EASE is a good architecture for straightforwardly building powerful AA functionality. Straightforward mappings exist for things like changing goals, changing capabilities, changing priority, etc. However, there are aspects of an actor's behaviour that are considerably harder to understand and change. We have presented some of these in the description of the system, e.g., the problems with a non-deterministic negotiation algorithm, and briefly look at some others here.

A feature of behaviour based style architectures (like this one) is that some observed behaviour may be an emergent property of the interactions between several behaviours (in our case agents) and the environment. Such emergent behaviour does not have an explicit representation in the actor, hence is difficult for $\mathcal{I}$ to present to the user and $\mathcal{A}$ to (intentionally) change.

Some (often abstract) properties of the actor's behaviour are not represented, because they were never explicitly captured at the design stage. For example, the *aggression* of a simulated pilot might not be explicit in the specification, hence will not be easily available for presentation by $\mathcal{I}$ and changing by $\mathcal{A}$. Such information *could* be captured by appropriate use of named constants however, in a violation of the Design Information Guideline, it is likely to be fairly common that not *all* abstract aspects of behaviour will be represented. Those that are not represented are hard to view and change at run-time.

The ability of the user to predict what the actor is going to do is limited by the ability of both EASE and the user to predict changes in the environment. That is, the user can never know exactly what the actor will do

because they cannot know what will happen in the environment. EASE has no explicit model of what it expects to happen in the environment, so it is difficult to present to the user a picture of even what is *likely* to happen in the environment and hence to the actor's behaviour. Such a limitation might be important for safety critical applications where it is very important for the user to anticipate and counter any problematic behaviour.

Our use of the AA tools discussed above, has shown that the speed and dynamic nature of the domain sometimes makes it very difficult for the user to make changes in time to be effective. This is partially a problem with the interfaces which are not always very efficient to use but also a fundamental property of the environments. For example, by the time the user realizes a RoboCup goal-keeper is not going to come off the goal line to attack a striker with the ball it may be too late to make a change that will force it to. It is quite noticeable that it is much easier to make effective changes to the slightly less dynamic TACSI actors than in the more dynamic RoboCup ones. Since the underlying speed of the environment is fixed it may not be *possible* to build any system where the user has useful control at the lowest levels of abstraction.

5.9 Other Behaviour/Agent Based Actor Architectures

To conclude this chapter we very briefly review some of the literature regarding actor architectures with a similar philosophy to EASE. Other Directed AA applications were discussed in Chapter 3, here we look only at some background to the behaviour based style of architecture.

Actor architectures aiming for emergent intelligence without having a central reasoning centre primarily developed from the seminal work of Brooks on subsumption architectures (Brooks 1991*a*,*b*). Similar work by Minsky also described how a society of simple agents could produce intelligence through their interactions with one another and their environment (Minsky 1988).

Subsumption architectures were a radical departure from mainstream AI ideas that careful, abstract centralized reasoning was required for intelligent behaviour. Subsumption architectures use a set of fairly simple behaviours interacting via primitive controls (i.e., subsumption) and the environment. In such architectures an actor is decomposed in a *behaviour oriented* manner rather than a functionally oriented one (Steels 1994). The observed behaviour of the actor is an *emergent* property of simple behaviours interacting with each other and with a complex world. Subsumption architectures depart from the assumption that the agent should (or could) maintain

an accurate, abstract model of the environment with which to reason. Instead, each behaviour (at least conceptually) senses and acts in the world independently, sensing only what is required to achieve its specific task.

Early successes of the subsumption approach resulted in many others following the ideas. A good summary of the key properties of behaviour based systems can be found in Mataric (1992). Mataric (1994) developed groups of behaviour based robots for her PhD thesis work. Parker (1998) also worked with behaviour based robots, focusing on fault tolerance and team behaviour with hierarchical arrangements of behaviours. Blumberg (1997*b*) uses hierarchical behaviour based architectures to produce interesting character behaviour, usually of animals, for interactive theatre applications. Tyrell (1993) returned to the biological inspiration for behaviour based systems by sticking closely to ethological models in his realization of a behaviour based system.

Inspired sometimes, by Minsky, other authors have used simple *agents*, instead of behaviours, working together to produce complex behaviour (Gerber et al. 1999, Neves & Oliveira 1997, Nakashima & Noda 1998), although the basic ideas are essentially the same as behaviour based systems. The differences between agents and behaviours are subtle, the key difference being that agents are slightly more independent and autonomous than behaviours (MacKenzie 1996). Using agents instead of behaviours clouds the border between behaviour based agents and distributed AI because the agents need not be physically co-located. Ossowski's agent-based $ProsA_2$ architecture is a good example of an architecture that could be looked at either as a distributed AI system or a single behaviour based actor controller that uses agents instead of behaviours (Ossowski & García-Serrano 1999). Agents seem to provide a better metaphor for inexperienced users to understand, hence agent based architectures have been used as the basis for creating complex behaviour in several end-user development tools, e.g., AgentSheets (Repenning n.d.) and ToonTalk (Travers 1996).

EASE has the same basic ideas as all these architectures. We chose to think of our components as agents instead of behaviours because agents seem to provide a better metaphor for end users to understand the actor's behaviour. EASE is probably simpler than most of the architectures mentioned above, mainly because the simplicity aids the effective implementation of AA. This *may* mean that EASE cannot produce as complex behaviour as some other architectures but it also makes the architecture easier to work with, especially when changes are needed at run-time.

5.10 Summary

In this chapter we have described EASE, an end-user system for building actors for interactive simulations. EASE uses a hierarchical society of agents, explicitly represented in an actor, mirroring the goal hierarchies of the actor, for decision making. Agents at the bottom of the hierarchies negotiate with each other to decide on concrete actions of the actor. EASE has been designed with AA in mind from the earliest stages of development. At run-time, a user has access to a wide range of information about the reasoning of the actor and a wide range of controls to change the actor's behaviour while it continues to run. That powerful AA interfaces supporting Directed AA can easily be built is due to characteristics of the underlying architecture. Since the guidelines were followed, the simplicity of the AA interface implementation can be seen as support for the utility of the guidelines. The extended example showed that the user does indeed have a wide range of control over the actor's behaviour at run-time.

6. ADJUSTABLE AUTONOMY FOR PERSONAL ASSISTANTS

An exiting use of intelligent agents is to streamline everyday processes in human organisations (Tambe, Pynadath & Chauvat 2000). Agents, acting as personal assistants, can find information, make purchases, filter email, order lunch and perform a variety of other routine tasks on behalf of their human user (Maes 1994*a*). Furthermore, *teams* of agents can work together, each agent acting on behalf of one user, to streamline co-operative processes in a human organisation. Figure 6.1 shows the conceptual relationships between the agents and humans in a multi-agent system for human collaboration. Teams of agents can coordinate the activities of their users, balance schedules, arrange meetings and so on. The aim of the agents is to free up their user's time for more productive, knowledge level work.

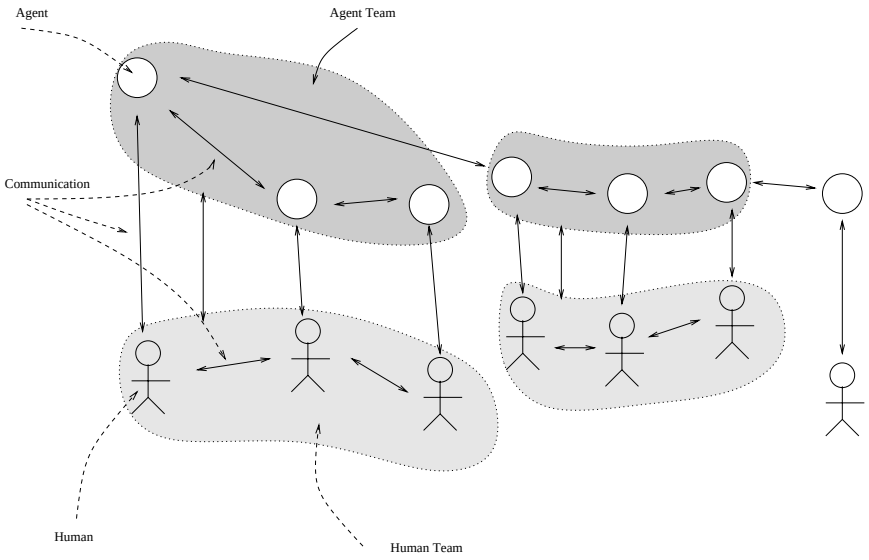

Fig. 6.1: *A conceptual visualization of the relationships between humans and agents, human teams and agent teams.*

A human organisation is an extremely challenging domain for even the most intelligent agent to *always* act appropriately. The complexity of human behaviour and the intricate and unpredictable social interactions between people provides a challenging environment for intelligent humans and an even bigger challenge for intelligent agents. Furthermore, despite an array of high tech sensors and communication devices (e.g., Palm Pilots, GPS, WAP-enabled phones), it is not always possible or convenient for agents to know the location of particular users and certainly not possible for agents to know their user's intentions. Even if an agent does have an accurate picture of the world and a good understanding of organisational procedures, choosing actions that actually improve the running of that organisation is far from trivial.

It is simply not reasonable to expect that agents can act autonomously in a human organisation without occasional, costly mistakes being made and without personal preferences being occasionally "bulldozed" by the autonomous agent. The dilemma is to try to take advantage of the strengths of agent technology without suffering for its weaknesses. The ideal situation is to have agents acting when their actions will be useful but leaving decision making to humans when agent actions are more likely to have undesirable effects. Introducing Adjustable Autonomy (AA) into a human collaboration agent system provides a mechanism that can allow the limitations of the agents to be avoided while letting the agents use their capabilities to help users with everyday tasks. Thus, the aim of the AA is to remove autonomy from agents when their behaviour is not useful, but give agents autonomy when their behaviour helps the running of the organisation.

In a human collaboration system, agents will be working around the clock in the service of their users. An important implication of this is that autonomy reasoning ($\mathcal{R}$) must be performed by software because the users will not always be around to make decisions. $\mathcal{R}$ needs to identify agent actions that are likely to be costly, annoying or otherwise undesirable and stop agents from taking those actions. When $\mathcal{R}$ is deciding whether an agent should be able to act, many factors need to be considered, including the potential costs and benefits of the action, potential mis-coordination with the team, trust and timing. However, $\mathcal{R}$ needs to be performed in the same complex, dynamic, inaccessible environment as the "core" agent decision making. Thus, $\mathcal{R}$ must also deal with things like uncertainty, incomplete information and unpredictability.

The remainder of this chapter is organised as follows. Section 6.1 gives an overview of the functionality of the E-Elves, a fully deployed agent system for human collaboration. The description of the E-Elves sets the context for

the rest of the chapter. Then we look at what AA means in such a human collaboration context, highlighting the important issues that arise for AA developers. Section 6.3 gives a high level description of the design of the E-Elves system. Section 6.4 looks in more detail at the implementation, in particular showing how the system design meets the guidelines from Chapter 4. Evaluation of the influence of the guidelines on the ease with which AA can be implemented is in the next chapter. Section 6.5 gives some results of the E-Elves in actual use. The chapter concludes with a look at some related work.

6.1 Electric Elves

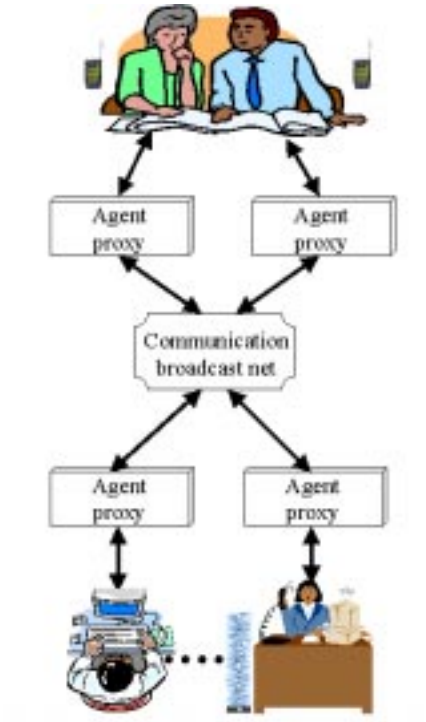

Fig. 6.2: *Basic architecture of the E-Elves. Agent proxies, supporting their users activities, communicate via* broadcast nets.

The E-Elves, is a fully deployed multi-agent system supporting human collaboration. The E-Elves has been in use at the University of Southern

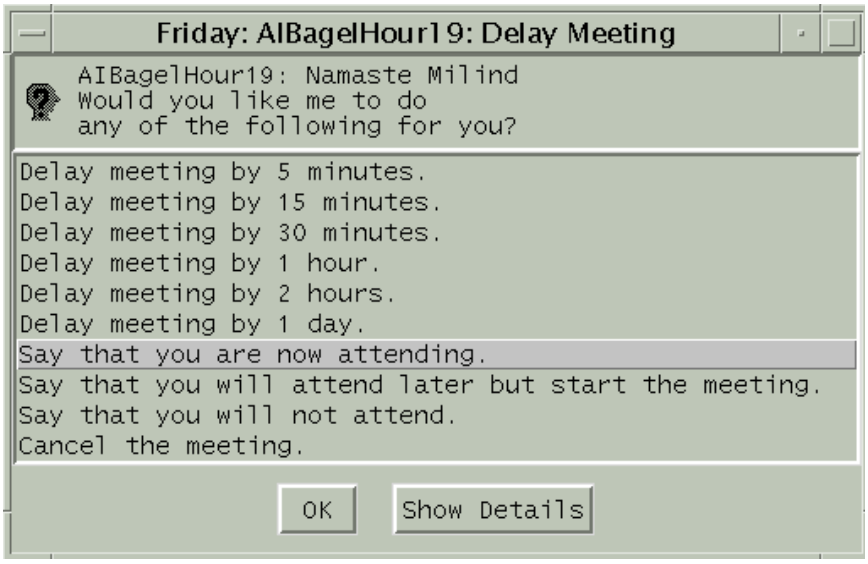

Fig. 6.3: *Dialog for delaying meetings. $\mathcal{R}$ has decided that the human should decide what action should be taken. Friday is asking the user, via a dialog on their workstation, what meeting delay action should be taken (if any).*

Fig. 6.4: *Friday, the user's agent proxy, reporting that is has ordered dinner from California Pizza Kitchen.*

California since June 2000. The E-Elves framework described below is due primarily to the efforts of Milind Tambe, David Pynadath and others at the University of Southern California.[1] The system consists of around 15 agents. Ten of the agents are *proxies* for human users, while the other agents perform a variety of auxiliary functions, such as interfacing to a calendar tool. Each proxy is called Friday (from Robinson Crusoe's servant Friday) and acts on behalf of a single user. Agents are members of various teams, mirroring the way their user is a part of different (human) teams. The agents communicate with other agents in their team via a *broadcast net* (see Figure 6.2). The agent team works around the clock, every day of the year to help facilitate the continuous smooth operations of the research group.

In the remainder of this section we describe the functionality of the system and discuss some of the problems that arise in everyday use. In subsequent sections we present our technical solutions to the problems, focusing on the AA services and their implementation.

The ultimate goal of the E-Elves project is to build agent teams that assist in all an organisation's activities, enabling coherent, robust attainment of dynamic mission goals and swift reactions to crises (Chalupsky et al. 2001). Results of the E-Elves work are potentially relevant to a variety of organisations, including the military, disaster rescue organisations, corporations and research institutions. For example, in a crisis, such as a natural disaster, E-Elves-like agent teams may help an organisation to urgently locate relevant personnel, coordinate their movement to the crisis site, coordinate the shipping of their equipment, provide them with the latest information, etc.

The current implementation of E-Elves includes a range of functionality for helping users with everyday tasks. Friday can delay meetings (see Figure 6.3), order meals for its user (see Figure 6.4) and volunteer its user for different roles, such as the role of meeting presenter. To help users find each other, Friday posts its user's probable current location on a web page (see Figure 6.5). Teams of agents can fill open roles in the group and reschedule meetings. The functionality of Friday, and the E-Elves in general, is continually being expanded, with new tasks being assigned to the agents as the capabilities become available.

If a user is delayed to a meeting, Friday can request that the other Fridays (i.e., proxies for other users) reschedule the meeting. If the meeting is rescheduled, each Friday informs their respective human user of the change.

[1] The author visited the group for six months in 2000 and worked with the group to improve the AA capabilities of the system.

Fig. 6.5: *A webpage showing that Ranjit Nair is not currently at ISI (as of 14:28 on 03/09/01).*

Friday uses mobile devices incorporating the Global Positioning System (GPS), as well as workstation activity, to reason about the user's current location. Due to limitations in these sensing abilities, Friday's reasoning about the user's location is somewhat uncertain. Simple spatial reasoning is used to reason about whether it is possible and/or likely that the user will get to upcoming meetings on time. Even if such spatial reasoning does not preclude the user arriving on time, Friday's model of the user's behaviour might suggest that the user *will* arrive late. A common example of such reasoning is for Friday to decide, just before a meeting, that a user not yet at the department for the day is likely to be late. Once Friday determines there is some reasonable probability that the user will be late, it needs to determine what (if anything) it should do to minimize inconvenience to all meeting participants, while maximizing the probability that the meeting will occur. We look at the details of this decision making process in Section 6.4.2.

The agent team acquires knowledge of upcoming meetings via communication with commercial meeting calendar software, wrapped with an agent proxy. Team *plans* require agents (and eventually their users) fulfill different roles in the meeting, e.g., the role of presenter. The team *auctions* off open roles to find suitable candidates. The auction mechanism allows the team to consider complex combinations of factors (Boutilier et al. 1999) and assign the best-suited user to the role. Fridays communicate bids to one other and, as a team, decide which is the best bid. In the current implementation a bid has two parts: the user's *willingness* to take on the role; and the user's *capability* for that role. A *capability matcher* (also implemented as an agent) manages data about the capabilities of the users to perform different roles. Friday looks up its user's (binary) capability to perform the role and submits a capability bid to the team. The user usually inputs the willingness part of the bid manually but Friday can make a decision autonomously if time pressure is high. Figure 6.6 shows the auction tool that allows human users to view auctions in progress and intervene if they so desire. Notice this is a *visualisation* of the auction, not an integral part of it. In the auction in progress in Figure 6.6, Jay Modi's Friday has bid that Jay is capable of giving the presentation, but is unwilling to do so. Paul Scerri's agent has the "best" bid (i.e., capable and willing) and was assigned the role.

Friday communicates with its user using wireless devices, such as personal digital assistants (PALM VIIs) and WAP-enabled mobile phones, as well as via pop-up dialogs on the user's workstation. Figure 6.7 shows a PALM VII connected to a GPS device, for tracking users' locations and allowing communication between users and agents, even when the users are

TEAMCORE20 | presenter

team-team

Agent	capability	willingness	Overall
Paul Scerri	1.0	1.0	1.0
David Pynadath	1.0	0.0	0.3
Milind Tambe	1.0	0.0	0.3
Jay Modi	1.0	0.0	0.3
Shriniwas Kulkarni			0.0
Hyuckchul Jung	0.0	0.0	0.0
Lei Ding		0.0	0.0
Takayuki Ito		0.0	0.0
Ranjit Nair		0.0	0.0
other-friday			0.0

Jay Modi

Assign

Fig. 6.6: *The E-Elves auction tool. The top left corner shows the meeting involved in the auction. To the right of the meeting name is the role being auctioned. Underneath the meeting name is the team that will attend the meeting. Empty spaces in the bid information are where information has yet to arrive.*

away from their desks. Communication via each of the different devices has different costs. For example, sending a message to a WAP phone is financially expensive and potentially quite annoying. On the other hand communicating via a workstation is financially free and, usually not a significant annoyance. The cost of communication needs to be taken into account by the agents and by $\mathcal{R}$.

Fig. 6.7: *Palm VII connected to GPS device.*

6.2 AA in Human Collaboration

AA is an essential part of a human collaboration system because it is responsible for minimizing the negative impact incorrect agent decisions may have on their users. The role of the AA is to remove agent authority when agent decisions are likely to be costly and wrong but allow the agent to

act otherwise. The basic aim of the AA is to maximize the autonomy of the agents while minimizing their harmful actions. Clearly, the more autonomous Friday is, the more time it saves its user. If Friday can correctly delay meetings without the user having to call or email other meeting participants user time is saved (or other participants time saved if the user was not in a position to inform the others). If Friday can, correctly, bid in an auction for a role without having to actually ask the user, time and effort of the user is saved. However, each time Friday makes a mistake the user may be badly effected.

In general, the AA is designed to have Fridays taking actions because this is what saves user's time – the user should only be consulted if there is a potential problem with agent decision making. Two types of problems potentially require autonomy to be taken from Fridays: *technical* and *social* problems.

Technical problems are caused by limitations in the technology the agent is using, including the limitations in sensing and actuation capabilities. For example, Friday may not always be able to detect the user's location or detect whether a meeting is actually occurring. Such technical limitations cause corresponding problems in Friday's decision making. For example, a meeting could be delayed because Friday wrongly believed the user was away from their office.

Social problems are those where agent decisions are wrong in that they contradict social conventions or norms. Such mistakes are caused by the differing, (often) unknown preferences and intentions of unpredictable users and the implicit, sometimes contradictory social conventions that govern their behaviour. For example, some people may prefer to inform other meeting attendees when it is likely they will be a few minutes late for a meeting while other users might believe that the notification is more inconvenient than the short delay. Other social conventions like, perhaps being more likely to attend a meeting on time with someone more senior than with someone more junior are not (and probably should not be!) written down in way agents can use them. Friday should try to respect such preferences, but if this is not possible, autonomy should be relinquished.

So $\mathcal{R}$ should look for both social and technical problems with the aim of withdrawing agent autonomy whenever either type of problem occurs. Although humans as well as agents will sometimes make such mistakes, we assume that it is always strictly better for the human to make a mistake than the agent. Notice also that the user might preempt Friday by explicitly requesting a particular action be taken. This is outside the framework of the AA, hence we do not consider the possibility in detail below.

6.2.1 Issues

A wide variety of different factors need to be considered when developing AA for human collaboration environments. Sometimes the factors are conflicting and need to be reconciled intelligently under different circumstances. In the following we consider some of the most important considerations when building AA for human collaboration systems.

Cost Friday has the potential to make costly mistakes when acting autonomously. For example, if Friday volunteers an unwilling user for a presentation, the user has to either actively retract the offer or do the presentation – either option wastes the user's and the organisation's time. Similarly, if Friday incorrectly orders lunch for a user, financial cost is incurred by the user.

It is not only autonomous actions that are potentially costly. When $\mathcal{R}$ decides that the human should have responsibility for a decision there are costs incurred by both use of the communication medium and in the disruption to the user. Furthermore, sometimes *inaction* can be costly to the organisation (see below). Hence, not only do we need to consider the costs of actions Friday or an agent team will make, but we also need to consider potential costs of $\mathcal{R}$ decisions.

Uncertainty Friday faces significant, unavoidable uncertainty when making decisions. Some uncertainty is due to incomplete physical sensing, e.g., is the user really at their workstation? Other uncertainty arises because human behaviour is unpredictable, e.g., does the user intend to attend the upcoming meeting? Hence, any decision Friday makes has some (often significant) probability of being different to the one the human would make if asked to decide.

The same sources of uncertainty are also present for $\mathcal{R}$, i.e., $\mathcal{R}$ cannot be sure that the information it is using is correct, nor can it be sure that its decisions are going to be without cost. Therefore, $\mathcal{R}$ must take into account the possibility that its information is wrong and that situations do not turn out as expected, hence it must consider the consequences of its actions if the environment is not as it believes.

For some decisions Friday, or the agent team, will have options which, on the basis of information available cannot be differentiated. A common example is when two or more users submit exactly equal bids to an auction for some role. Deciding which user to assign could be resolved randomly by

the software (and sometimes is) or it could be referred to human users, who may have more information to bring to bear on the decision.

Teamwork The problems presented by the human collaboration domain are compounded by the fact that agents and humans work in teams. The issues raised by involvement in a team can be divided into two groups. The first group of issues are those that are brought about because the team members must coordinate their actions. For example, an agent cannot wait indefinitely for a user to respond because other team members might be waiting. The second group of issues are related to group decision making. Because the agent team can make decisions as a group, the AA needs to deal with potential problems with group decision making. For example, the agent team decision about who will fill a certain role may be both wrong and costly in some cases, which implies that the particular decision should have been left to the human team. Hence, AA must be applied to group decision making as well individual decision making. We look in more detail at the two types of decision below.

The first type of team issues are those that arise because the team should maintain coordination. A challenge for individuals in team settings is to avoid mis-coordination with teammates when transferring autonomy, while simultaneously minimizing the risk of costly errors. In particular, Friday should not wait for user input, thereby inconveniencing other team members, if it could make a reasonable decision autonomously. For example, one approach to $\mathcal{R}$ uses uncertainty as the lone rationale for transferring decision-making control, relinquishing control to humans whenever uncertainty is high (Gunderson & Martin 1999). In a team setting, the agent cannot transfer control so simply. For example, consider Friday's responsibility to request the rescheduling of a team meeting if it thinks its user will be unable to attend on time. Rescheduling is costly, because it disrupts the schedules of the other team members, so the system can ask the user for input in an attempt to avoid making an unnecessary rescheduling request. However, while Friday waits for a response, the other users will wait in the meeting room – if the user does not arrive, other users will waste their time, while Friday sits idly by, doing nothing. On the other hand, if, despite the uncertainty, Friday is allowed to act autonomously and informs the others that its user cannot attend, then its decision may turn out to be a grave mistake if the user does attend. Thus, $\mathcal{R}$ must weigh possible team mis-coordination while waiting for a human response against possible erroneous actions as a result of uninformed decisions.

The second type of team issue is related to group decision making. Despite appropriate local decisions by individual agents, the overall team of agents can potentially make global decisions that are unacceptable to their counterpart human team. Because each agent has only local knowledge, when making their decisions they cannot know for sure the effect of their decision on team decisions. The most common example of this in the E-Elves occurs when individuals are bidding for roles. When the presenter role for the group's weekly research meetings comes up for auction, each user individually considers whether they have time, material, etc., to do the presentation. During busy times of the year, many users might be unwilling to present week after week. Maybe the same one (not busy) user will submit willing bids week after week and, hence, the agent team would select the same presenter week after week. To have the same presenter week after week is unacceptable at the team level because it defeats the purpose of the meetings (which is to keep the group informed of each others research). However, the agent team has few other options as it would have to be very bold to assign as presenter a human user who specifically said they were unwilling. So, while each agent acted benevolently and honestly, because its decision was based only on local knowledge, the result for the team was poor. The agent team can not make good decisions under all conditions, hence, for the smooth running of the organisation, it should not be allowed to. In such cases, the human team needs to be given responsibility for making the team decision. Thus, it is clear that AA is not only required for individual decision making but also for team decision making.

Learning Learning allows agents to adapt themselves to the changing preferences, work habits and environment of the user (Maes 1994*b*). Learning plays an important role in improving the ability of both $\mathcal{R}$ and Friday to improve the running of the organisation. For Friday, learning helps to improve core decision making, e.g., whether a delay or a cancellation of a meeting is most appropriate under particular circumstances. Learning in the AA helps improve its ability to make effective autonomy decisions. For example, users have different preferences about when Friday should refer decision making to them and when Friday can make a decision. In particular, some users will prefer Friday makes more decisions (and hence takes more risks) while other users, wanting more control, prefer Friday to ask them whenever there is any potential for a costly mistake.

However, when AA (and agents in general) are used in real-world environments it is important to protect users against temporary learning aber-

rations. Noisy data is a common learning issue, in any environment where agent sensing is not perfect or complete. However, in environments where mistake are potentially costly it is important that noisy data does not cause (even) temporary problems. For example, if the user cancels two Monday morning meetings the agent should (probably) not learn to cancel all Monday morning meetings because it is likely to be a simple coincidence that the meetings were cancelled rather than a user trait that should be learned. Notice, the problem is not that Friday will never learn the right thing, rather that it may do incorrect things, that negatively impact the user, while it learns. Because learning problems can cause real harm to users, mechanisms need to be employed to protect against them.

$\mathcal{R}$ in Software A fundamental aspect of using agents to streamline activities in organisations is that the agents will sometimes be acting even when their users are not available to provide input. This simple property means that humans cannot be relied on to do $\mathcal{R}$, i.e., *because we cannot rely on having a human around at all times we cannot rely on humans to interact with the agents when something goes wrong.* In fact, the system will not even know for sure whether the user could respond to a specific request in a timely manner, in part because its sensing of user location is incomplete and it will not know for sure where the user is. Hence, software, with all its inherent limitations, is responsible for deciding which actions Friday can take autonomously and for which the user should be consulted.

Having $\mathcal{R}$ performed by software constrains all other AA design decisions. In Directed AA, the human is often relied on to supplement AA information collection ($\mathcal{I}$) with things like "common sense" and an ability to draw implicit information from a data set. In E-Elves there is no such luxury because a human is not performing $\mathcal{R}$. For example, once, an earlier implementation of $\mathcal{R}$ in E-Elves delayed a meeting by five minutes – *nearly fifty times* in a row. This would not have happened if a user was performing $\mathcal{R}$ because clearly a sequence of fifty short delays is nonsensical.

Everything the AA can do must be provided for in code. For example, software cannot reason about the impact of someone's house burning down on the likelihood of that person getting to a meeting on time unless it knows about houses, emotional effects of their loss, etc. All the information that Friday can reason with must be explicitly provided by $\mathcal{I}$. The same constraint applies to realization of autonomy changes ($\mathcal{A}$) – in Directed AA systems the user can be relied on to bring common sense to autonomy changes, in the E-Elves, details and consequences of autonomy changes need

to be properly handled by software. This is an especially difficult problem – Friday's decision making was wrong in the first place, why should the autonomy reasoning, faced with the same limitations, be any better?

Required Agent Services

$\mathcal{R}$ decides whether Friday's (or the team's) decision is in some way "safe enough" to take without user input. To make such a determination requires "meta" information about the action, not just the intended action. In particular, Friday needs to provide the following information to $\mathcal{I}$ in order for $\mathcal{R}$ to be effective:

- Friday's plan for dealing with the current situation
- The expected implications of the planned actions
- The potential costs of the chosen actions and the probability of those costs being incurred
- The potential benefits of chosen actions and the probability of those benefits being received
- The likelihood predicted outcomes are wrong

Analogous information needs to be passed between the agent team and human team. For example, the agent team needs to provide the plan for closing an auction to $\mathcal{I}$. The information passed constitutes the $\mathcal{I}$ services that Friday and the agent team provide to $\mathcal{I}$.

The service that Friday and the team need to provide to $\mathcal{A}$ is simple – hold off execution of an action until it is approved by AA, i.e., this allows removal of authority to act on an action by action basis. Using such mechanism $\mathcal{A}$ shifts autonomy away from Friday or an agent team, to their human counterparts, when $\mathcal{R}$ decides the circumstances and the selected action warrant such a change in autonomy.

6.3 Conceptual Design of the E-Elves

In this section we present the high level design of the E-Elves and describe how it meets the requirements presented above. The design, to a greater or lesser degree, meets the guidelines given in Chapter 4. We first present the workings of the individual systems, then describe the team infrastructure.

In subsequent sections we describe the details of the implementation. Evaluation of the usefulness of the system is provided towards the end of the chapter. The relationships between features of the architecture, particularly those features resulting from adherence to the guidelines, are summarised in Table 6.1 and discussed more critically in the next chapter.

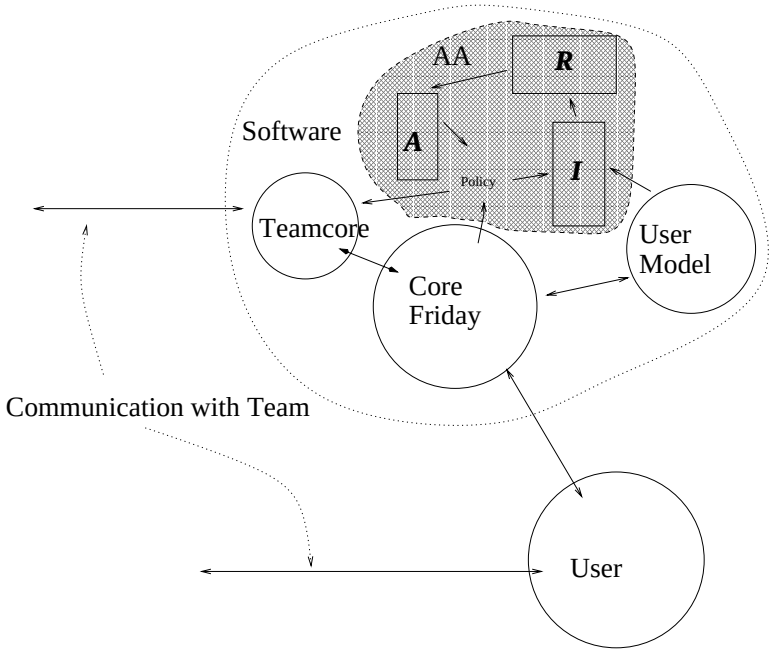

Fig. 6.8: *Conceptual design of AA for the Friday proxy agent. This figure shows more detail of one of the agents as shown in Figure 6.1.*

Individual Aspects Each user has a separate software system supporting their activities. The software system has four software components: "core" Friday, a model of the user, the teamcore proxy and the individual AA component. The basic arrangement is shown in Figure 6.8. Since, from the user's perspective the four components constitute a single system we sometimes refer to the whole system as Friday.

Both "core" Friday and the individual AA module rely heavily on the first part of the system, a shared, explicit model of the user and the environment. The model captures the user's location and likely movements, the importance and costs and benefits of different situations, simple spa-

tial information and so on. The model also explicitly contains uncertainty information about the probabilities particular state changes will occur, probabilities that the locations of users are correct, etc. This model effectively provides the common language with which Friday and the AA software communicate.

The core software module of the individual system is Friday. Friday reasons and acts on behalf of its user in the organisation's activities. Friday reasons from the *perspective* of its user, using only local knowledge, e.g., it does not know about the meeting schedules or locations of other users. Where possible, Friday observes basic social conventions, such as attempting to minimize inconvenience to other users. Friday reasons as if it has complete autonomy to take actions, i.e., it does not consider that the AA might allocate some decision making responsibility to the user. For every type of circumstance where its input might be useful Friday develops a complete *policy* which specifies what action it will take in each possible eventuality. For example, when dealing with a user's possible late arrival at a particular meeting, Friday will generate a policy that specifies what it will do if the user is late, on time, not coming at all, etc. For example, it might specify that if five minutes before the meeting the user is not at the department it will request a 15 minute delay. Once the policy is created, until the episode is over (e.g., the meeting takes place or is abandoned), Friday simply follows the policy, taking whatever action is dictated by the policy in the current, perceived situation. The policy is also made available to the AA module.

Friday uses Markov Decision Processes (MDPs) for its basic reasoning. (MDPs are also used for $\mathcal{R}$). The MDPs provide a natural representation of uncertainty and lookahead. The details of the MDP representation are presented in Section 6.4.2.

The third software component is the AA component. It takes a policy created by Friday and creates a corresponding *autonomy policy*. The autonomy policy sets out in advance which actions Friday will be allowed to take autonomously and for which responsibility will be given to the user. Notice that the AA never *changes* Friday's selected actions, it simply decides whether they will be permitted to be taken. The shared model provides the information required to determine for which actions the potential benefits of asking the user for input outweighs the expected benefits of letting Friday act autonomously. Notice, that the autonomy policy explicitly represents cases where the user does not provided timely input when requested and specifies whether to keep waiting for a response or take autonomous actions, if the user does not respond.

The final component of the individual software system is the STEAM-based team work module (Tambe 1997). The module provides the teamwork infrastructure, e.g., communication and team reasoning capabilities, that are required to allow the agent to work as part of a team.

A design decision was made that $\mathcal{R}$ should check individual Friday actions, rather than, say, deciding whether whole policies, strategies or principles were acceptable. So, for example, $\mathcal{R}$ decides whether submitting a particular bid for a particular role at a particular time is acceptable rather than whether Friday should submit a bid in general. The decision can be justified by noting that the high level intentions of the system are generally reasonable but it is the details that cause the problems, hence it is reasonable to accept all high level intentions and just look at the details. For example, the principle that Friday should minimize inconvenience to meeting participants from late arrivers is likely to always hold, but whether to delay, cancel or start the meeting is a difficult decision.

Furthermore, each meeting to be monitored, meal to be ordered, etc. is handled independently of all the others. So, for example, the impact on future meetings of one being rescheduled is not considered. The separation greatly simplifies the complexity of the reasoning required. The complexity of the reasoning is reduced due to a dramatically simplified state space and because interactions between episodes are ignored. In practice, we have not noticed this simplification to be significantly detrimental to the quality of agent reasoning (although in theory it might be). Because each situation is dealt with in isolation it is feasible to create a policy, *a priori*, that specifies Friday's actions in *all* the circumstances the might occur. Without the partitioning the state space of the resulting policy would be too large for a complete policy to be produced. The availability of this policy makes $\mathcal{R}$ simpler because the actions for each set of circumstances are known a priori.

Team Aspects Because agents work together there is an ability for team decision making and action (Tambe, Pynadath, Chauvat, Das & Kaminka 2000). Separate AA is used at the team level, sub-team level and individual level. The team takes input from individual Fridays and comes to some decision, using domain specific STEAM rules. As at the team level, the team produces an action policy and the team AA uses that policy to produce a team autonomy policy. The team AA looks at the potential costs and benefits of the action and decides whether to allow the team to act (see Figure 6.9).

The team members communicate with each other over *broadcast nets* to

make group decisions (Tambe, Pynadath & Chauvat 2000). A broadcast net is effectively a multicast network at the application layer. Hence, a message sent by any agent to a broadcast net is received by all the agents connected to that broadcast net. Messages are encoded using KQML (Finin et al. 1997), but only the syntax and not the full semantic power of the language is leveraged.

The team models each meeting as a *joint intention* (Tambe, Pynadath & Chauvat 2000). By the rules of STEAM, individuals keep each other informed of the progress of the joint intention (e.g., a meeting is delayed, cancelled, etc.). The STEAM rules provide a well known, clear and (relatively) simple framework for team behaviour. Using STEAM allows the team reasoning to be separated from the individual reasoning, which simplifies the reasoning required at both levels. Furthermore, STEAM *role relationships* model the important relationships among team members. For example, the *presenter* role is critical to a meeting since the other attendees depend on someone giving a presentation, hence the user fulfilling the presenter role is critical to the meeting's success. If the presenter cannot attend, the team recognizes a *critical role failure* that requires remedial attention. On the other hand a *passive participant* role is not critical, so if an agent assigned to such a role is failing to fulfill that role (i.e., its user is failing to fulfill their role) the agent team may safely ignore the failure. A common example of a passive role failure is ISI's (voluntary) weekly department lunch. Many users do not attend the lunch, effectively leading to them failing in their passive participant role. Fortunately, however, the team reasons that non-attendance by passive participants is not grounds for cancelling or delaying the lunch.

$\mathcal{R}$ Implementation $\mathcal{R}$ takes the policy generated by Friday (or the team) and decides whether to allow the agent (or the team) to take each of the actions proposed in the policy. The uncertainty and cost information, retrieved from the shared model, is used as the basis for deciding whether or not to allow the agent (or team) to go ahead with the action. $\mathcal{R}$ produces an *autonomy policy* that includes the actions that Friday can take autonomously and the states in which the user will be asked for input.

The most obvious thing for $\mathcal{R}$ to take into account is whether Friday is faced with potential high costs and reasonable probabilities of incurring those costs. However, $\mathcal{R}$ must also consider the possibility that the user does not provide input because they are occupied or out of reach. Transferring control to the user may actually make the situation worse in some

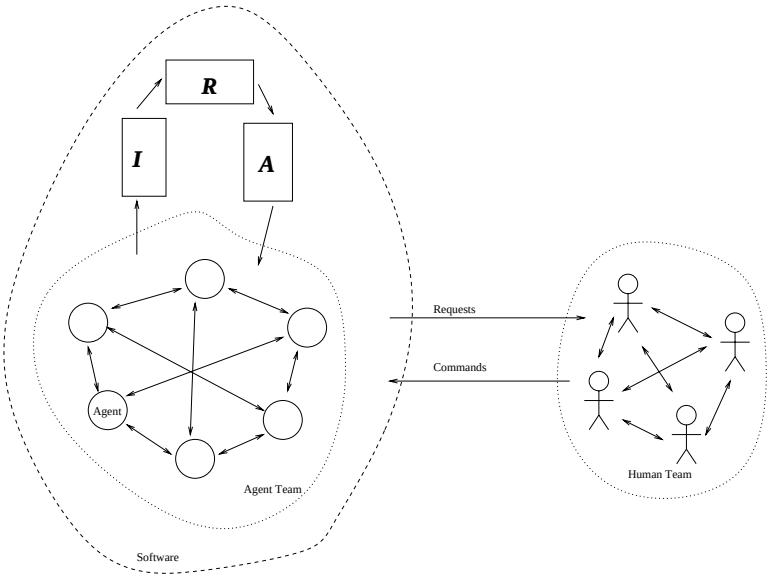

Fig. 6.9: *Conceptual design of AA for agent team to human team interaction.*

cases. Transferring control to a user who fails to respond, only delays the autonomous decision, potentially exacerbating the problem. For example, if the system waits for user input until five minutes after the meeting should have started and receives no response it is faced with the same decision it had five minutes before the meeting but now with guaranteed higher cost (because the other users have had to wait needlessly for five minutes). $\mathcal{R}$ also needs to take into account that asking the user for input may be annoying to the user.

Notice that while Friday and $\mathcal{R}$ use a shared model of the user and environment, they are looking at quite different things. Friday is trying to work out what actions to take to maximize the probability that desirable states will result, while $\mathcal{R}$ is trying to work out for which of Friday's actions the expected value of letting Friday act, outweigh the benefit of asking the user for input.

Learning Machine learning improves the shared model that provides input into the reasoning of both Friday and the AA. Improving how well the model represents the environment improves both Friday's decision making and $\mathcal{R}$. Although the rationale for including learning applies both to Friday's

decision making and to $\mathcal{R}$, here we are primarily interested in the impact on $\mathcal{R}$. As mentioned earlier, a critical aspect of the learning is to protect the user from temporary learning aberrations. Noisy data, which will occur frequently in domains such as the human collaboration one, should not lead to a system which makes unreasonable decisions, even temporarily. We refer to this problem as the *safe learning challenge*. This is a well known challenge in the field of machine learning and various approaches have been proposed for solving it (Kendrick 1981, Schneider 1995).

We take a two-pronged approach to safe learning challenge: (i) building substantial knowledge into the model; and (ii) providing a safety net to prevent harmful policies being learned. The safety net, in turn, has two parts. The first is a strict limitation on the structure of the state space. Regardless of what probabilities are learned, there are certain actions that will never be taken in certain states. For example, one constraint does not allow any one Friday to instigate more than three delays to a particular meeting. The second part of the safety net is to check whether certain important properties of the policies hold *after* the learning algorithms have been run. For example, an agent that does *nothing* is not particularly useful so we check that the model is not such that the agent will never do anything bolder than watch the evolving situation.

Avoiding Mis-coordination

To illustrate how the different elements of Friday and the AA come together, we consider in detail how the system handles the problem of avoiding team mate mis-coordination. Figure 6.10 shows a simplified state space for the situation. The numbers along the top of the figure are the number of minutes relative to the meeting time. Circles represent states and are labeled with the location of the user given that state. Arcs between states are labeled with the probability that arc is followed. For example, the two arrows furthest to the right show that between the meeting time and five minutes after the meeting time there is a 1% chance the user will stay in their office and a 95% chance they will go to the meeting location (for clarity, other possible location changes are not shown).

Maintaining coordination with teammates when AA is involved is a non-trivial problem, because the risk of incorrect autonomous action needs to be traded off against the possibility a user cannot provide input. The E-Elves attacks the problem with a three-step approach: (i) weighing costs of waiting for user input and subsequent team mis-coordination against cost of possible erroneous actions; (ii) flexibly transferring control between agent and human

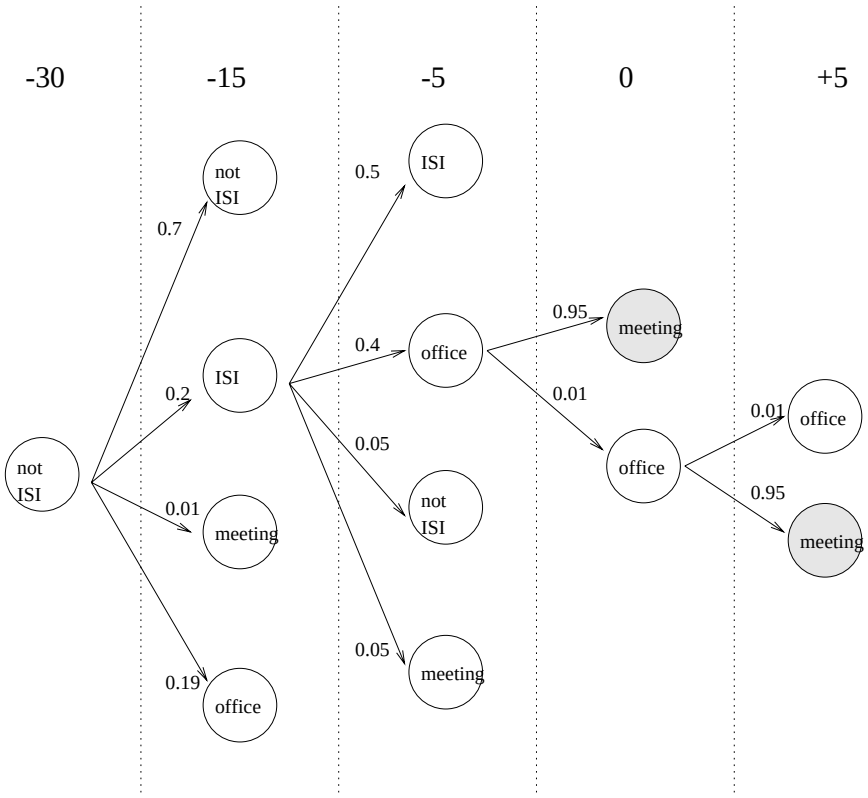

Fig. 6.10: *Part of the environment state space for an episode where a user should attend a meeting. Final states are shaded. Arrows between states show* some *of the transitions that might occur.*

rather than rigidly committing to an initial decision; (iii) electing to change the coordination rather than taking risky actions in uncertain states. The first step of this, i.e., the weighing of costs and benefits, is done via the MDP with input from the user model. The costs and benefits and likelihood of various situations are represented explicitly in the model and reasoned about in the basic decision making.

The second step of the approach to avoiding mis-coordination requires that agents avoid rigidly committing to changes in autonomy. For example, if autonomy is removed from the agent for a decision, the agent should not wait indefinitely for a user response, as a slow response could jeopardize the team activity. Instead, the agent must continuously reassess the developing situation, possibly changing its previous autonomy decisions. So, for example, if decision making control had been passed to the user but no response had been received, when the situation changes, the agent could be given back control and allowed to make an autonomous decision. The flexible approach is captured in the autonomy policy produced by $\mathcal{R}$. Figure 6.11 shows an example of some of the possible state transitions that might occur if the user is asked for input.

The third step of the approach arises because an agent may need to act autonomously to avoid mis-coordination, yet it may face significant uncertainty and risk when doing so. In such cases, an agent can carefully plan a change in coordination (e.g., delaying actions in the meeting scenario) by looking ahead at the future costs of team mis-coordination and those of erroneous actions. Such changes in coordination could, amoung other things, "buy time" to reduce the uncertainty or cost. An MDP is especially suitable for producing such a plan because it generates policies after looking ahead at the potential outcomes. For instance, an MDP supports reasoning that a short delay "buys time" for a user to respond to a query from an agent, potentially reducing the uncertainty surrounding a costly decision, albeit at a small cost. Thus, an agent might choose a 15 minute meeting delay to give time for an absent user to arrive, or respond, before cancelling a meeting. Figure 6.12 shows some of the effects a selection of Friday's actions have on the situation. For example, a transition near the middle of the figure shows Friday selecting a 15 minute delay which leads to a transition between a state where the user is in their office five minutes before the meeting to a state where the user is at the meeting location 15 minutes before the (new) meeting time.

The three steps, i.e., reasoning about costs and benefits, flexible autonomy changes and using coordination changes, are all required to ensure a flexible, robust approach to handling the problem of avoiding mis-

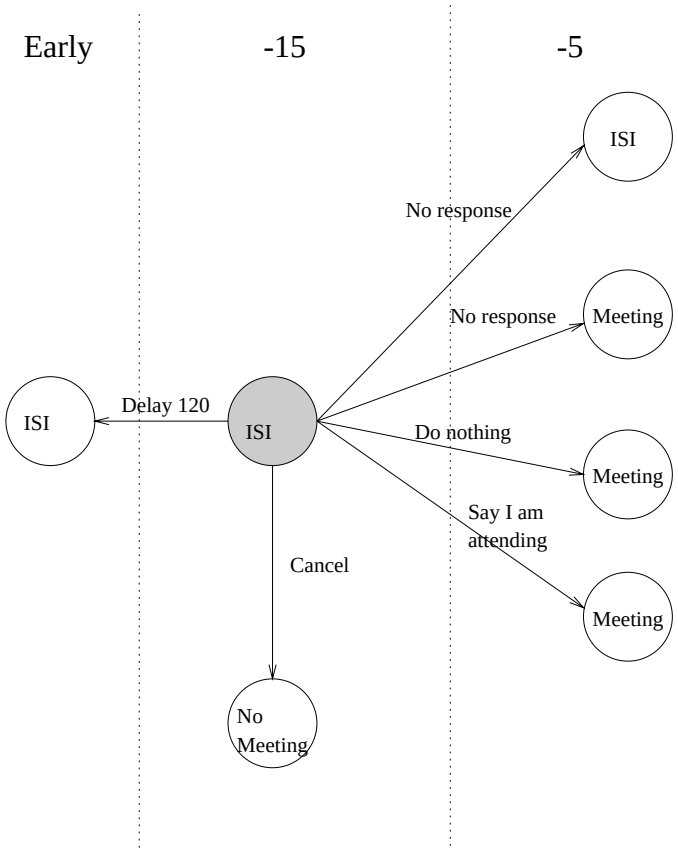

Fig. 6.11: *Part of the state space for the case where the user is asked for input. In the shaded state the user was asked for input. State transitions are labeled with the reply from the user.*

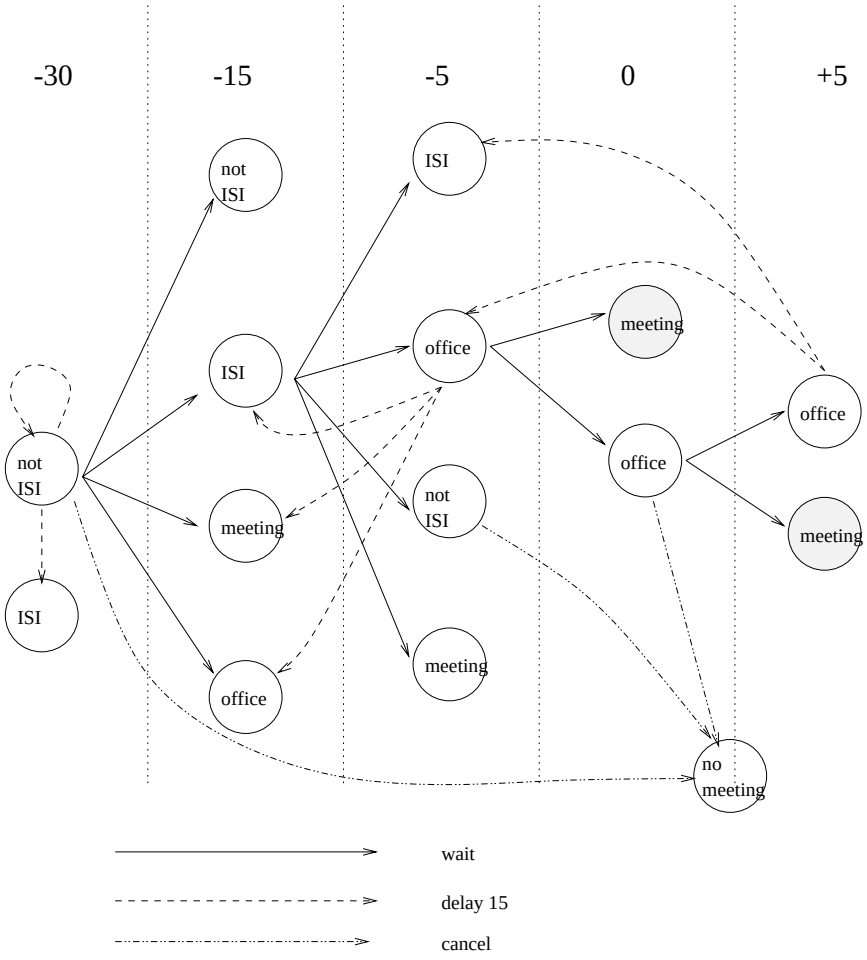

Fig. 6.12: *Part of the state space with some of the transitions that might occur if Friday acts. Arcs, the meaning of which is shown in the legend, show a selection of possible outcomes of different actions Friday might take.*

coordination. Later, we present some empirical results showing the use of the approach. Problems other than mis-coordination are handled in similar ways.

E-Elves in Terms of the Conceptual Model E-Elves maps to the framework described in Chapter 2 as follows. (We use the subscript f to denote Friday.) Friday will have some goal $g \in G_f$, e.g., the goal might be to have its user get to a meeting on time. The things Friday can do make up the set A_f. Friday will be authorized to pursue certain sub-goals, e.g., waiting for a changing situation, delaying the meeting, telling the other participants to start without the user, etc. These possible actions make up the set $A_f - C_f$ for the agent. Friday chooses some sub-goal, from $A_f - C_f$, that best fits the situation as it sees it. $\mathcal{R}$ takes the suggestion of the agent and decides whether to let the agent act or to try to defer authority to the user. When removing autonomy from the agent for a goal, g, g is removed from Friday's set G_f and put in the user's set G_u, as well as g being put temporarily in C_f. The team AA can be mapped to the definitions in an analogous manner.

Guideline	Agent Feature	AA Facility
Explicit Information Guideline	Model of the user and organisation Costs explicitly represented	Model used for $\mathcal{R}$ Straightforward assessment of risks
Software Engineering Guideline	Separation of team and individual reasoning	Simplification of $\mathcal{R}$
Design Information Guideline	Uncertainty explicitly represented	Information used in $\mathcal{R}$
Deterministic Execution Guideline	Behaviour policy execution is deterministic	Allows a priori creation of autonomy policy Enables lookahead to reduce risk
Explicit Behaviour Guideline	Behaviour policy is explicit	Used to create autonomy policy Allows risk reduction strategies
Building Blocks Guideline	Separate handling of each episode	Simplified $\mathcal{R}$
No Extra Mechanisms Guideline	None	
Design Expecting Failures Guideline	Team infrastructure	No detailed model/ information of other team members required

Tab. 6.1: *Summary of the E-Elves features resulting from adherence to each of the guidelines and the features of the AA that utilized the features.*

6.4 Implementation Details

This section describes the some details of the implementation of the E-Elves. The aim is to provide only enough detail for the implications of the design decisions and their impact on the AA to be clearly understood. Table 6.1 summarises some of the important features and their impact on AA.

Note In practice, both the "core Friday" decision making and AA are implemented as a single module. Potentially, the two parts could be separate modules, as per the conceptual design, communicating via some sort of interfaces. However, because everything is implemented in software it is simpler to combine the reasoning of Friday and the AA in a single, tightly integrated software module – even though we *think* conceptually about the parts separately. It is sensible to build them in the way that allows that interface to be as simple as possible. In the following we separate out the AA and Friday specific parts of the implementation, where possible.

6.4.1 Model of Organisation

A model of the user and the environment is at the core of the E-Elves. The model is state based with features representing important aspects of the environment, including the user's location, a meeting's importance and what the other team members have been told. Judicious selection of model features is important to the actual ability of the agents to make good decisions.

Time is represented discretely, i.e., to ensure a finite number of states only certain time steps are represented (e.g., *very early*, *one hour before*, *15 minutes before* and so on). Time is measured relative to the relevant event, e.g., times are measured relative to the time when a meeting should start.

Probability distributions describe the likelihood that the circumstances will move from one state to the next in the next time step. For example, a probably distribution captures the likelihood that the user will go from their current location to some other location by the next time step. The probabilities are instantiated individually for the specific situation that is being dealt with. Table 6.2 shows a sample of probabilities the user will arrive at a meeting at different times, given that they are not there five minutes early.

The probability distributions capture the likely effects of a particular action when taken in a particular state. For example, the action of delaying a meeting by five minutes has a certain likelihood of causing a transition from a state where a user will be five minutes late for a meeting to a state

	User current location		
Expected Delay	not ISI	ISI	Own office
On time	0.4	0.8	0.9
5 mins late	0.2	0.1	0.01
15 mins late	0.1	0.025	0.01
30 mins late	0.1	0.025	0.01
1 hour late	0.05	0.005	0.005
2 hours late	0.05	0.005	0.005
Never arrive	0.1	0.04	0.06

Tab. 6.2: *Hand-coded probabilities of all possible different lengths of time user will be late to a meeting, given their location five minutes before the meeting.*

where the user is on time for the meeting. The probability distribution of the transitions for a *wait* action (i.e., to have the agent do nothing) is also represented. Table 6.2 shows a sample of such probabilities.

The model also includes details of the costs and benefits of being in specific situations. For example, to have a user at a meeting, on time, has some benefits, but for meeting participants to be sitting around waiting for the user has some costs. The costs and benefits of being in a state are added up to determine the value of being in that state. For example, the worth of being at a meeting on time has the benefit of the meeting and no costs. Conversely, being in a state where others have been informed that the user will be 15 minutes late but has not arrived at the revised time has high costs because the users are having their time wasted *and* have been informed of a delay (which has a cost). Table 6.3 shows some examples of different costs and benefits to give an idea of their relative worth. Other costs and values are calculated via more complex functions and can not be so easily written down.

User Input An important part of the system is an explicit representation of the value of user input and the cost of getting that input. The value of user input is calculated by looking at how Friday's assessment of the possible costs and benefits would change given different possible user inputs. If the user input could potentially impact very positively on Friday's assessment of the situation then it is more likely to be requested. For example, say Friday's decision was to cancel a meeting. This is potentially a very costly

Item	Value	Cost
Role values		
Presenter	0.9	
Discussion leader	0.8	
Active participant	0.5	
Passive participant	0.1	
Action costs		
Delay 15 mins		0.00004
Delay 2 hours		0.0003
Cancel		0.5
Cost of others' time		
1 other		0.33
4 others		0.74
9 others		0.87

Tab. 6.3: *Example hand-coded costs and rewards for the meeting scenario.*

action – if the user *does* arrive, the meeting has been cancelled needlessly, though delaying the meeting for someone who does not turn up is even more costly. However, if the user will most likely say to "cancel" the expected impact of their input is fairly low (because it makes little difference to the expected costs and benefits). If there is some significant chance the user can make a decision that substantially reduces costs then the value of their input is higher. For example, if there is a significant chance the user will decide to just delay the meeting a short time instead of cancelling (which is Friday's decision) then the value of the user input is high.

The *cost* of communicating with the user depends on the device that will be used in the communication which in turn depends on where Friday thinks the user is. The cost of using a device has both a financial component, e.g., a WAP phone message is expensive, and a disturbance component, e.g., answering a Palm Pilot message takes a non-negligible amount of time. The costs of using the different devices influences how often $\mathcal{R}$ decides to pass over decision making control. For example, if Friday decides to delay a meeting by five minutes it is probably not worth contacting the user via a WAP phone (because the cost of the delay is low even if Friday is wrong and the cost of using the WAP phone is high), however if the user was at their workstation it may be worth passing autonomy to the user. Furthermore, there is an explicit model of the probability the user will respond to

a communication via some device. For example, the probability of a user replying to a WAP phone message may be lower than the probability of replying to a workstation dialog box – at least when it is thought the user is at their desk. $\mathcal{R}$ takes this probability into account when deciding whether autonomy should be transferred.

Notice, that the modeling of communication costs is not needed for *Friday* decision making, it is there specifically for the purposes of $\mathcal{R}$. The cost of communicating is important to Friday decisions because if a meeting delay is made, clearly the decision must be communicated to the team. However, Friday does not reason about this explicitly, rather it is built into the cost for delaying a meeting (along with many other factors, like inconvenience, frustration, reputation, etc.) Hence, an explicit communication model is an interesting example of when adding extra design information to an agent makes AA easier to implement.

Figure 6.13 shows how the number of states in which $\mathcal{R}$ will decide to ask the user for input decreases as the cost of asking the user increases. As expected, as the *cost* of asking increases, the number of states in which Friday relinquishes autonomy decreases. Figure 6.14 shows that as the *likelihood* of the user replying to a message is increased in the model, so does the number of states where $\mathcal{R}$ decides to ask. The figures show that $\mathcal{R}$ trades off intelligently between the wasted time if the user does not respond, the value of the information the user can provide, the cost of asking the user, and the likelihood of the user replying.

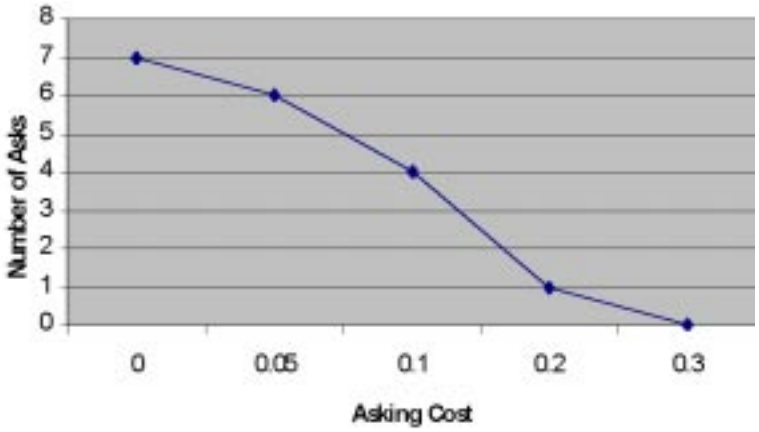

Fig. 6.13: *A graph of the number of states in the autonomy policy where a human is asked for input versus the cost of asking.*

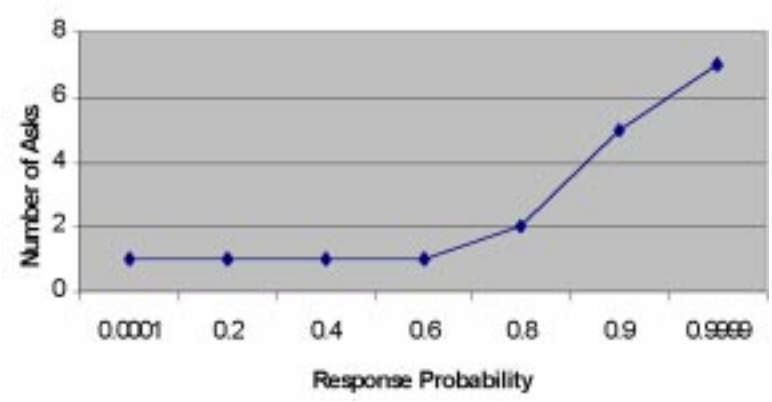

Fig. 6.14: *A graph of the number of states in the autonomy policy where a human is asked for input versus the probability of the user responding to the request.*

The cost of communicating with the user *should* also depend on how busy the user currently is. For example, a message to WAP phone while the user is on the train home is less costly than when they are in the middle of a meeting. Detecting how busy the user is is a difficult problem. Horvitz et al. (1998, 1999) are working on the problem of inferring how busy a user is from their workstation activities. Such technology would be a useful addition to the E-Elves.

6.4.2 Friday Decision Making

Markov decision processes (MDPs) (Puterman 1994) are used as the basic reasoning mechanism for both Fridays and $\mathcal{R}$. MDPs were a natural choice for addressing the issues identified for AA in the human collaboration domain, e.g., reasoning about the costs of actions, handling uncertainty, planning for future outcomes, and encoding domain knowledge.

An MDP takes a model of costs and benefits, a model of potential actions and a probability model of different environmental state transitions occurring and returns a "policy" consisting of the action with the highest expected value in each state. The basic cost and benefit information comes from the shared model, as does the probability information.

A reward function maps each state to a value representing how desirable it is to reach that state. The makeup of the reward function depends on the particular situation being handled. For example, when trying to achieve

the goal *get lunch*, desirable states will be those where the user is enjoying the lunch of their choice at lunchtime while for goals to get to meetings desirable states are those where the meeting takes place on time. Hence, in such states the reward would be high.

Most actions involve some sort of cost, e.g., there is an inherent cost involved in delaying a meeting because it means that the other meeting attendees are inconvenienced, however there is also a cost in having other meeting attendees wait for someone who is late. The costs of taking actions are captured in the reward function for the state. For example, there is a lower reward in states where the meeting has been delayed.

Given the state space, actions, transition probabilities, and reward function of the MDP, a standard value iteration technique (Russell & Norvig 1995) is used to compute an optimal policy, $policy^*(s)$:

$$policy^*(s) = \arg\max_a \Sigma_j M^a_{sj} U(j) \tag{6.1}$$

where $U(s)$, the utility of being in state s, is :

$$U(s) = r(s) + \max_a \Sigma_j M^a_{sj} U(j) \tag{6.2}$$

and $M^a_{sj}U(j)$ is the transition probability between state s and j given that action a is taken.

The value iteration results in a policy that specifies the action with the highest expected utility in each state. It is feasible to generate the entire policy because of the coarse granularity of the state space (e.g., there are only a small number of "times" represented) and the separation of the handling of different situations. The complete decision making policy, as well as additional information, is made available by $\mathcal{I}$, to $\mathcal{R}$ for the creation of an autonomy policy.

6.4.3 Partitioning

To overcome computational complexity costs, we rely on partitioning the MDPs. Thus, a Friday has a separate MDP for each episode that it encounters, e.g., each meeting, presentation or meal time where it might take some actions to save user time. The reason for this, that it simplifies reasoning, was discussed above.

For each *type* of situation Friday has a different *generic MDP*. The generic MDP captures the basic actions, states and costs that are important

in that type of situation. The generic MDP is instantiated at runtime with the details of the situation, e.g., the type of meeting. For example, if the meeting is important the cost of delaying the meeting (in the reward function) is higher than for an unimportant meeting and the probability of the user making the meeting on time might be higher. These values are instantiated in the generic MDP before the value iteration is performed. Using instantiations of a generic MDP, rather than using the changing factors as variables in the MDP, helps keep the computational complexity of the MDP within reasonable bounds.

6.4.4 Example – Delaying Meetings

In this section we present a detailed example of the handling of one particular situation type, namely meeting delays. We begin by describing Friday's decision making then describe the autonomy reasoning.

The delay MDP's reward function has a maximum in the state where the user is at the meeting location when the meeting starts. A component of the reward, denoted r_{user}, focuses on the user attending the meeting at the meeting time. The reward is given so Friday has incentive to delay meetings when its user's late arrival is possible. However, in isolation, r_{user} could drive the agent to choose arbitrarily large delays, virtually ensuring the user is at the meeting when it starts,[2] but forcing other attendees to rearrange their schedules – a very costly activity. The team cost is considered by incorporating a negative reward, denoted r_{repair}, with magnitude proportional to the number of delays so far and the number of attendees, into the delay reward function. The larger the number of delays the lower the reward. A pre-learning constraint (see Section 6.4.6) allows no more than 4 delays to a meeting.

However, explicitly delaying a meeting may benefit the team, since without a delay, the other attendees may waste time waiting for the agent's user to arrive. Therefore, the delay MDP's reward function includes a component, r_{time}, that is negative in states after the start of the meeting if the user is absent, but positive otherwise. The magnitude of this reward, like r_{repair}, is proportional to the number of attendees. The reward function also includes a component r_{role}, which, like r_{user}, is positive in states where the user is in attendance and zero otherwise. However, the magnitude of r_{role} also increases with the importance of the user's role (e.g., presenter vs. passive participant) to the success of the meeting, thus representing the value

[2] We assume that the user always wants to attend a meeting and, given enough time, will make it there. This assumption does not always hold.

of the user's attendance to the *team*. Finally, the reward function includes a component, $r_{meeting}$, which is positive once the meeting starts and zero everywhere else to deter meeting cancellation.

The overall reward function for a state s is a weighted sum of the components:

$$\begin{aligned} r(s) = & \lambda_{user} r_{user}(s) + \lambda_{repair} r_{repair}(s) + \lambda_{time} r_{time}(s) + \\ & \lambda_{role} r_{role}(s) + \lambda_{meeting} r_{meeting}(s) \end{aligned} \tag{6.3}$$

The precise values of the reward function components can be varied for different meeting types. For example, the value of r_{repair} will be bigger if the meeting involves the head of the department. Changing the value of different components changes the relative values of different states and, hence, the policy that results from the value iteration process.

Although taking into account team costs (mainly via the r_{repair} and r_{time} rewards), Friday's decisions are on behalf of its user only; the team may not concur (see below).

The delay MDP's state transitions are associated with the probability that a given user movement (e.g., from office to meeting location) will occur in a given time interval. The MDP designer encodes the initial probabilities, which the learning algorithm (described in Section 6.4.6) then customizes. In practice, transitions to states where the user arrives on time are highly likely.

Standard value iteration results in a decision making policy, dictating Friday's proposed actions, being created. Adding autonomy reasoning amounts to adding an extra action *ask* that results in the user being consulted for input. The *ask* action, through which the agent gives up autonomy and queries the user, has two possible outcomes. First, the user may not respond at all, in which case, the agent is performing the equivalent of a *wait* action. Second, the user may respond to the request, with one of the ten responses shown in Figure 6.3. The communication model, part of the shared model, provides the probability of receiving a user's response in a given time step. The cost of the "ask" action is derived from the cost of interrupting the user. The probabilities and costs vary with the communication medium. We compute the expected value of user input by summing over the value of each possible response, weighted by its likelihood (computed using the delay MDP as a model of the user's decision-making). We assume that if the user does respond they will do so accurately. For example, if the user provides input suggesting a five-minute delay, then the system knows that it will incur the cost of the five-minute delay but will then receive the maximum

reward when the user arrives at the (rescheduled) meeting on time, i.e., in the resulting state the reward function will have a negative component for requesting the delay but a (large) positive component because the meeting will occur.

Furthermore, the *lookahead* in MDPs enables effective long-term solutions to be found. As already mentioned the cost of rescheduling, r_{repair}, increases as more and more such repair actions occur. This provides a compact scheme for supporting some history dependency in the cost of future states. For example, in a given situation the user model might indicate that the user was most likely to arrive at a meeting five minutes late, with a much smaller chance they will be 15 minutes late. Without lookahead, the clear, best option for Friday is to ask for a five minute delay to the meeting. However, with lookahead it may become clear that a fifteen minute delay is the best option, for the following reason. If the user *did* end up being fifteen minutes late the costs would be *very* high because either two meeting delays would need to occur or other users would be firstly told of a delay *and still* made to wait. On the other hand, an initial 15 minute delay is only slightly more costly than a five minute delay and covers both the cases of the user being five and 15 minutes late. Clearly, more complex situations occur when more lookahead is required.

Lookahead is used in both the core Friday reasoning and in $\mathcal{R}$. For example, there are some situations where, although asking for user input at one point in time might be the best action, if the user fails to respond Friday ends up in a very bad position. Hence, it is better for Friday to take a (somewhat) risky action straight away than wait for user input that may not come.

Table 6.4 shows a portion of a sample policy for handling meeting delays. The *time* column gives the time relative to the meeting in minutes, so, *-60* is one hour before the meeting. The *User Location* column shows the user's inferred location. *ISI* and *not_ISI* are abstractions for *somewhere at the department but not their own office or the meeting location* and *not at the department* respectively. The *Told* column summarises what request the individual has so far made to the team, e.g., when the others have been asked nothing is written as *none* and when the others have been told of some delay is written as *delay*. The *Repairs* column shows how many times Friday has asked the other agents to change the time of the meeting. A pre-learning constraint limits the total number of requests to change a meeting one individual can make to three, i.e., the number in the *Repairs* column will never be higher than 2. The *Best Action* column shows the action Friday would take if allowed to act autonomously. The final column, *Autonomy*,

has "yes" if Friday will take the action and "no" if the user will be asked for input.

The policy is specialized to a meeting where the user is an active participant in a two person meeting. Generic transition probabilities are used, i.e., the MDP is straight "out of the box". Notice that in Table 6.4, states when Friday does not have autonomy ("no" in the final column) correspond well with Friday actions that we find intuitively "suspicious", i.e., without knowing more about the situation it is difficult to say whether Friday's action is appropriate. Conversely, when Friday does have autonomy, the actions it is proposing are fairly intuitively correct (though information not shown in the table and unbeknown to Friday may mean the actions are not appropriate). In the full policy, $\mathcal{R}$ will allow Friday to act autonomously in 720 of the 736 states. In 564 states it Friday decides the best action is to wait, in 72 it decides the user is not attending, in 40 it decides the user is attending and in 60 it decides on a delay of some kind.

6.4.5 Team Decision Making

Not all decision making responsibility is in the hands of an individual Friday, some decisions are made via a group decision making process. For example, decisions made by Friday to delay a meeting affect the whole team. So, when an individual software system decides (whether autonomously or not), e.g., that a meeting delay is required, the opinion is submitted to the team for a group decision. Figure 6.15 shows the basic idea. The decision whether to actually delay the meeting or not is a *group decision* made by the agent *team*. (The agent team potentially relinquishes responsibility to its human counterpart team for deciding whether or not to actually delay.) Hence, in reality, Friday is not making decisions about whether meetings should be delayed but only about whether it should request that the team delay the meeting. In the current prototype, the STEAM rules, by which the team decides, mean that delay requests will be approved if the requesting user is important enough (according to their role in the meeting) and/or the meeting is small enough, but rejects them otherwise. Just because the team will reject an unacceptable request does not mean that an individual agent can ignore the costs and benefits of particular actions to the team and make such decisions.

The team mechanism protects the team from incorrect, malicious and/or poorly informed decisions by individual Fridays. For example, one Friday cannot know that a small delay in a meeting will cause significant problems for other Friday users, so it might make a decision that to it seems reasonable

Time	User Location	Told	Repairs	Best Action	Λ
Early	Own office	none	0	wait	yes
-60	not_ISI	none	0	wait	yes
-60	ISI	not_attend	1	wait	yes
-30	Meeting location	none	0	wait	yes
-15	not_ISI	delayed	1	wait	yes
-5	ISI	none	0	delay_5	no
-5	ISI	delayed	2	wait	yes
-5	not_ISI	none	0	delay_60	yes
-5	not_ISI	delayed	2	wait	yes
-5	Meeting location	none	0	wait	yes
-5	Meeting location	delayed	2	wait	yes
-5	Own office	none	0	delay_5	no
-5	Own office	delayed	2	wait	yes
0	ISI	none	0	delay_15	no
0	ISI	delayed	1	delay_15	no
0	not_ISI	none	0	delay_60	yes
0	not_ISI	delayed	1	delay_60	yes
0	Meeting location	none	0	user_attending	yes
0	Meeting location	delayed	1	user_attending	yes
0	Own office	none	0	delay_15	no
0	Own office	delayed	1	delay_15	no
5	ISI	none	0	delay_15	yes
5	not_ISI	none	0	delay_60	yes
5	Meeting location	none	0	user_attending	yes
5	Own office	none	0	delay_15	yes
15	ISI	delayed	2	not_attending	yes
15	not_ISI	delayed	2	not_attending	yes
15	Meeting location	delayed	2	user_attending	yes
15	Own office	delayed	2	not_attending	yes

Tab. 6.4: *Part of the combined decision making and autonomy policy for meeting delaying. There are 736 states in the complete policy.*

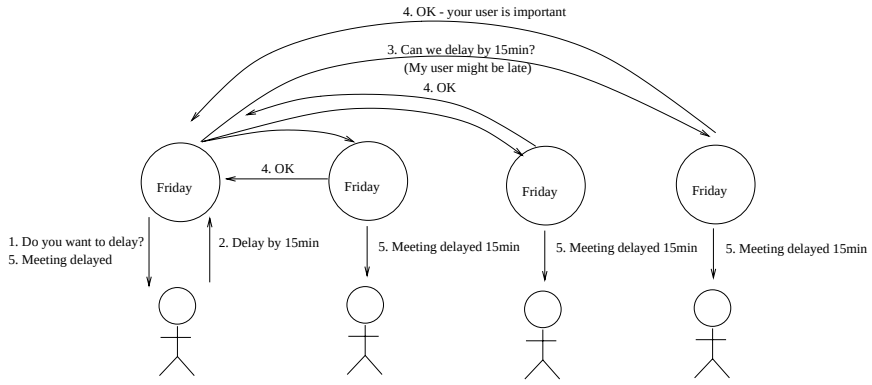

Fig. 6.15: *A high level view of the conceptual information flow for team decisions. The text next to the arrows shows the information passed. Numbers indicate the ordering of messages.*

but is very undesirable for other team members. The team mechanism allows such requests to be rejected, without having to give all Fridays intimate knowledge of other users. In the same way the team mechanism protects the team from uninformed Friday decisions, it protects the team from incorrect, malicious and/or poorly informed decisions made by individual users.

In a team context, the AA problem is for the team to decide whether to rely on its own reasoning or to relinquish the control to the user team. An important consideration for team level AA, like for individual AA, is to try to maximise the number of decisions the team makes autonomously because this maximises the amount of the team's time the system saves. Clearly, the costs of incorrect team action must be carefully weighed against the time saved. Thus, analogous to the AA at the individual level, the team-based AA needs to reason about uncertainty, cost, and potential developments in the world. Furthermore, most of the issues that are relevant at the individual level, e.g., reasoning about lack of response, are also relevant at the team level. Hence, we again chose MDPs for the team $\mathcal{R}$. The team MDPs compare the expected value of consulting the human team against the expected value of making an autonomous decision.

It is important to note that the decision of the agent *team* may be made autonomously and may reject the decision of the individual *user*. For example, when a *passive participant* in a large meeting tried to cancel the meeting, the agent team would override the user and allow the meeting to continue. This is a relatively rare example of software knowingly overriding

the desires of a human user but is a consequence of a having system where users can take actions that detrimentally effect other users. Of course, if the meeting should really be cancelled, but the reasons for the cancellation are beyond the comprehension of the agent team, the human team might assume team decision making responsibility and agree to the cancellation request. Hence, the users have not really lost control of the system, rather the system is ensuring the bad decision making by a single user does not impact the whole team.

Example – Auction Reasoning

During an auction, team members submit bids indicating whether they are capable and/or willing to perform some role for the team. The capability part of the bid is usually submitted autonomously by Friday after consultation with the capability matcher. The willingness part of the bid, on the other hand, is usually submitted by the user. Due to the nature of the decisions involved (*Am I willing to do a presentation next Thursday?*) the times when the willingness parts of the bids are submitted can vary greatly. Some users will know immediately that they would like the role and submit an unwilling bid as soon as the auction opens, others will know that other constraints prevent them from taking on the role and immediately submit a not willing bid. More often, however, users will put off decisions until their situation becomes clearer, e.g., until they know how busy they will be the day before the presentation.

The team-level decision to close an auction and assign a presenter for a talk has high uncertainty and cost, so the agent team will sometimes need to consult with the user team. Within reason, the sooner the role is assigned the better off the assigned person will be. For example, assigning someone to a presentation one week before the presentation is better than assigning them one hour before the presentation, because some time is needed to prepare.

When deciding when to close an auction at least three factors are important. The first factor is, obviously, finding a suitable, preferably optimal, candidate for the role. Second, the possibility of better bids coming in needs to be considered. Finally, the team must try to close the auction at such a time as the assigned person has enough opportunity to prepare for their role.

The team can take one of two actions, closing the auction and assigning the role or waiting. The *wait* action also allows a human to make an assignment (see Figure 6.6), so it is synonymous with asking the human team for

input (but less costly than actively asking the team to respond). We assume that once the auction is closed it cannot be re-opened. The states of the team-level MDP have abstract team features, e.g., *few*, *half*, *many*, or *most* bids have been submitted for a role. The agent team needs to know the probability distribution for higher quality bids arriving in the subsequent time steps, i.e., what is the chance that a better bid than the current best bid will be submitted in the next hour. This distribution is encoded a priori and adjusted autonomously from observations.

In each state, s, in the auction MDP the team receives the reward:

$$r(s) = \lambda_{bid} r_{bid}(s) + \lambda_{accept} r_{accept}(s) + \lambda_{time} r_{time}(s) \tag{6.4}$$

The reward function has a maximum when the team assigns a clearly superior, high-quality bidder to the role at the "optimal" time. The reward r_{bid}, is proportional to the quality of the winning bid and encourages the team to leave the auction open until a high-quality bid comes in. The reward r_{accept} is positive if there is a clearly best bid and negative if there is no clearly best bid, this discourages the agent team from making assignments when there is uncertainty surrounding which bid is best. The reward r_{time} is based on how appropriate the timing of the assignment is – too early and the team may miss out on receiving better bids, too late and the assigned user will not have sufficient time to prepare. The r_{time} reward captures any time pressure associated with the role, encouraging the team to make an assignment earlier (e.g., to give the assigned presenter more time to prepare). Table 6.5 shows part of auction closing policy.

As with the MDP delay rewards the relative values of the auction MDP rewards are adjusted depending on the role type being auctioned. For example, the optimal time to be assigned clearly depends on whether the role is to fly across the Atlantic and demonstrate a new product or if it is to collect the group's lunch from the local sandwich shop. Similarly, the importance of getting a "best" bid relative to the importance of assigning someone early will vary. For example, anyone who is willing could be assigned to go to San Diego to pick up a parcel, so as soon as the team gets a bid indicating a willing user, the user can be assigned the role.

6.4.6 Safe Learning

An accurate model of the user is critical to good $\mathcal{R}$. Improvements, via learning, to the shared model can lead to improved AA decision making, primarily by personalizing the reasoning to particular users and the organisations they are part of. For example, improved knowledge of the user's

Time	Bid Quality	% bids in	Difference	Decision
1month	Very_Low	None	Same	wait
1month	Very_Low	All	Different	wait
1month	Very_High	Many	Different	assign
1month	Very_High	Most	Same	assign
2weeks	Very_Low	Few	Different	wait
2weeks	Very_Low	Half	Same	wait
1week	Very_Low	Many	Different	wait
1week	Very_Low	Most	Same	wait
1week	Medium	All	Different	assign
1week	High	Few	Similar	wait
5days	Medium	All	Different	assign
5days	High	All	Same	assign
3days	High	Few	Same	wait
3days	High	Few	Similar	wait
3days	High	Few	Distinct	assign
3days	High	Few	Different	assign
1day	Medium	Half	Similar	wait
1day	Medium	Half	Distinct	assign
12hours	Medium	Half	Same	wait
12hours	Medium	Half	Similar	assign
12hours	Medium	Half	Distinct	assign
12hours	Medium	Half	Different	assign
4hours	High	None	Different	wait
4hours	High	All	Different	assign
2hours	Low	Half	Same	assign
1hour	Low	Few	Same	assign

Tab. 6.5: *Part of the auction closing policy. The "Difference" column shows the difference between the quality of the best bid and the second best bid. The total auction policy has over 1300 states.*

proclivity to make meetings on time or respond to a certain type of message can increase the confidence $\mathcal{R}$ has in a decision. By improving such knowledge, $\mathcal{R}$ can make more accurate judgments about when the situation warrants attempting to get user input.

In practice, we have observed that the small amount of data that is produced (most users have less than five meetings per week) does not provide enough of a sample to drastically modify the hand coded probability distributions. In most cases, however, the data has shown users to be *more* unpredictable than we estimated, e.g., they are late for more meetings than we thought (and encoded in the model).

Fridays learn via a simple reinforcement learning algorithm that periodically examines the extensive logs created during system execution and updates parameters in the MDP user model according to what has happened previously. For example, the probability that a user is late to a particular type of meeting is gradually updated based on the number of times the user is and is not late. Notice, the learning only improves the quality of the underlying model, not the reasoning done with the model, i.e., the state or action space does not change only the probabilities that different transitions will occur changes.

Noisy data is a significant problem for Fridays, e.g., they may be unable to correctly observe whether a user is actually attending a meeting on time, hence do not have a good indication of whether their decision to delay the meeting was a good one. The feedback the Fridays get is used to update their models and therefore influences future decision making. In a deployed system, anything learned has an immediate effect on the users — a phenomenon we have seen to be sometimes harmful in practice (Scerri et al. 2000). So, temporarily incorrect models, due to noisy data, cannot be tolerated (Schneider 1995).

An important first step in ensuring the agents do not cause too many problems, is to create an initial "generic" user model that encodes designer domain knowledge into an initial model which in turn allows agents to function well straight "out of the box". That is, the designers estimate and hand-code transition probabilities that are likely to be close to the "real" values that would be learned via extensive learning. By starting out with a well-developed model, agents avoid learning all decision-making details from scratch — one of the problems with the initial attempt to learn using C4.5 (Scerri et al. 2000). In turn this means that each new data point can be allowed to influence the model only slightly, so occasional noisy data is unlikely to perturb the model enough to cause very bad decisions. Over time, however, the (hopefully) large number of accurate data points slowly adapt

the model to the peculiarities of the organisation and user. Nonetheless, in order to gain the trust of the users, it is important to augment the learning algorithm with a *safety net* to ensure that that even in the presence of noisy data, agents are prevented from learning very harmful knowledge.

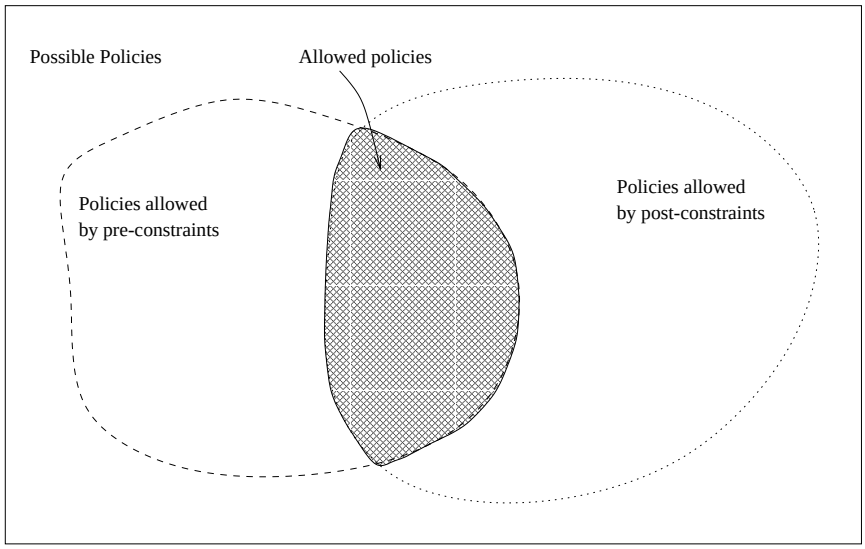

Fig. 6.16: *A Venn diagram showing the relationships between the policies allowed by pre- and post- learning constraints and the set of allowable policies.*

Safety Net

The learning safety net has two parts: pre-learning constraints and post learning constraints. Separating the constraints into two parts allows us to represent naturally and computationally efficiently the range of constraints the domain presents. Figure 6.16 shows how the two types of constraints limit the allowed policies.

Pre-Learning Constraints Pre-learning constraints put strict limitations on the actions that can be taken by Friday in certain states by removing those actions from the model, i.e., the MDP does even consider the possibility of taking some actions in some states. The actions cannot be taken even if learned transition probabilities would have led to the conclusion that those

actions had the highest expected utility. Pre-learning constraints naturally model the idea that some actions are simply non-sensical to take in certain states.

For example, Friday could learn from a limited, noisy data set that its user never attended Monday morning meetings. Given this (misleading) knowledge, Friday might cancel all Monday morning meetings as soon as it finds out about them. Introducing a pre-learning constraint that does not allow cancellations before, say, fifteen minutes prior to the meeting will allow us to avoid this problem. Closer to the meeting, when the cancellation constraint is not in force, Friday may still cancel the meeting, however it might also have other data which leads it to take a different action.

Ongoing work at ISI is looking further into the idea of constraints, including looking at ways a user can specify their own constraints.

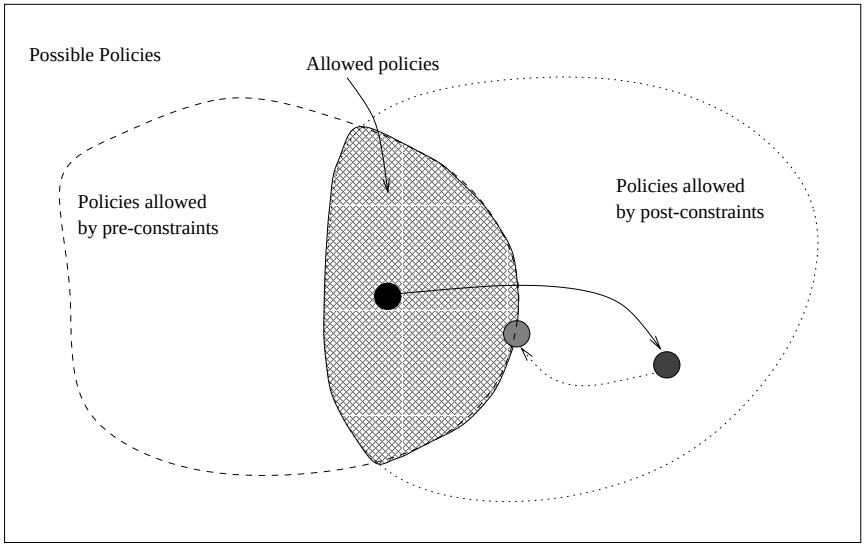

Fig. 6.17: *Diagram showing the effect of the post-learning check on the policies produced by a learned set of parameters. The dark circle represents the original policy. The solid line shows the change due to learning and the dotted line shows the changes due to the post-learning check.*

Post-learning Checking The post-learning constraints ensure that the policies produced by the MDP have certain desirable properties after the learn-

ing algorithm has run. Such constraints naturally represent limitations on certain "global" properties of the policy. For example, the delay MDP should not learn that all actions are too costly and thus generate a policy of complete inaction. In other words, there should be at least some states in which the agent will take some action other than *wait*.

Rather than exhaustively checking every state of every policy generated by all the possible MDPs (recall that some values are adjusted depending on the specific situation so there are many MDPs with potentially different behaviour), the post-learning checking algorithm uses *heuristics* to isolate a small number of states where the (non)existence of the property being checked should be most clearly seen. The algorithm can check only these states to determine whether it is likely that a property holds for a given MDP. For example, the heuristic *the user is not at the meeting room 15 minutes after the meeting should have started* isolates a small number of states where we can expect the meeting policy to specify an action (usually delaying the meeting) if it will take an action in any situation. In other words, if Friday does not take an action in a state where the need to act is very high we can be fairly sure that it will not take actions in any state. If the policy indicates that Friday will act in states that are checked then the constraint is adhered to. If Friday takes no action in the states that are checked it is unlikely it will take an action in any state. Hence, the heuristics give us a way to very efficiently check, with reasonable reliability, whether the policy has certain global properties.

If the post-learning check finds that a required property of the policy *does not* hold, the learning algorithm gradually adjusts the learned transition probabilities back toward their original values until it finds an acceptable set of values, i.e., ones that result in MDPs that have the required properties. Figure 6.17 illustrates this idea. This heuristic-based approach is our first step toward tackling the open research issue of providing post-learning guarantees within E-Elves.

6.5 Using the E-Elves

In this section we aim to show that the E-Elves is an effective AA system by presenting data collected during the system's day to day use. In the next chapter we evaluate the design in more detail, in particular examining the impact of the guidelines on the AA implementation.

6.5.1 General Observations

The E-Elves system has been operating in a research group at the Information Sciences Institute at the University of Southern California since the beginning of June 2000. The system runs around the clock, seven days a week (occasionally interrupted for bug fixes and enhancements). There are currently nine agent proxies (belonging to nine users *ramanan*, *ito*, *jungh*, *kulkarni*, *modi*, *pynadath*, *scerri*, *nair*, and *tambe*), one agent proxy for a project assistant, one capability matcher (with proxy), and an interest matcher (with proxy).

The general effectiveness of E-Elves is shown by several observations. Since the E-Elves deployment, the group members have exchanged very few email messages to announce meeting delays. Instead, the Fridays autonomously inform users of delays, thus reducing the overhead of waiting for delayed members. The amount of time saved is difficult to quantify – especially as we, the system's developers, would often spend time discussing whether the system did the right thing.

Second, the overhead of sending emails to recruit and announce a presenter for research meetings is now assumed by agent team run auctions. So, no user needs to take the time to send out email asking for interested parties, determine who should do the presentation from the replies and announce the winner – this is all done automatically.

Third, a web page, where Friday agents post their user's location, is commonly used to avoid the overhead of trying to track users down manually. For example, questions like *It is 4pm, is Jay here yet?!* can be answered without going to Jay's office.

Fourth, mobile devices keep us informed remotely of changes in our schedules, while also enabling us to remotely delay meetings, volunteer for presentations and so on.

Using Friday to order meals has also been a popular feature. In fact, Friday is so heavily relied on to order lunch that one local "Subway" restaurant owner even suggested marketing to agents: *"... more and more computers are getting to order food ... so we might have to think about marketing to them!!"*.

Most importantly, over the entire span of the E-Elves' operation, the agents have *never* repeated the catastrophic mistakes that previous implementations had been susceptible to (Scerri et al. 2000). Although the current agents do occasionally make mistakes, these errors are typically as serious as asking the user for input a few minutes earlier than may be necessary, etc. Thus, the agents' decisions have been reasonable, though not always

optimal.

6.5.2 Individual AA

Over the course of six months (June 1 to December 31) nearly 700 meetings were monitored. Figure 6.18 illustrates the number of meetings monitored for each user. Some users had less than 10 meetings monitored, while others had over 250 monitored. Most users had about 20% of their meetings delayed. Figure 6.19 shows that usually 50% or more of delayed meetings were autonomously delayed. In this graph, repeated delays of a single meeting are counted only once. The graph shows that the agents are acting autonomously in a large number of cases. Equally importantly, humans are also often intervening, indicating the critical importance of AA. Autonomously initiated delays were usually due to Friday detecting that the user was not at the department around the meeting time. Human initiated delays were often due to the user knowing in advance, sometimes well in advance, that they would be late (or absent) for a meeting and choosing to utilize the E-Elves framework to inform the other meeting participants and arrange a new meeting time.

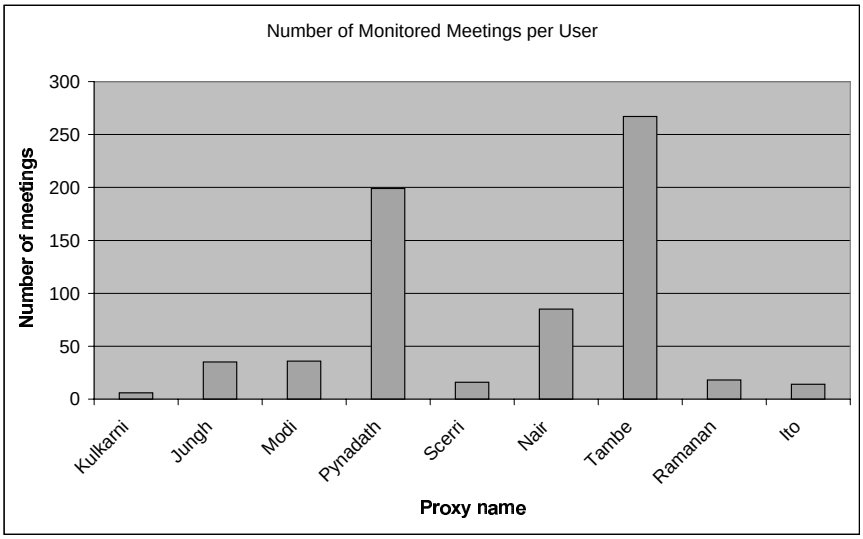

Fig. 6.18: *A graph of the number of meetings monitored by Friday for each user.*

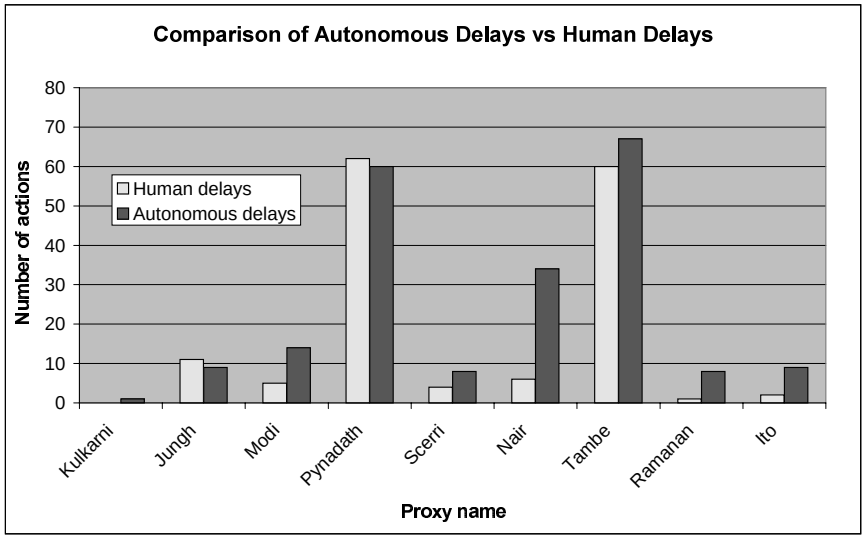

Fig. 6.19: *A graph of the number of meeting delays that were done autonomously vs. the number of delays initiated by the user.*

6.5.3 Team AA Evaluation

Figure 6.20 plots the number of daily messages exchanged by the proxies over three months (6/1/2000-8/31/2000). The size of the daily counts reflects the large amount of coordination necessary to manage various activities, while the high variability illustrates the dynamic nature of the domain.

The presenter role for the research group's weekly meetings is regularly decided using agent run auctions. Table 6.6 shows a summary of some of the auction results. The column headed "Date" shows the dates of the research presentations. While the auctions are held weekly, several weekly meetings were cancelled over the summer due to conference travel and vacations. The column headed "No. of bids" shows the total number of bids received before a decision to assign a user to the role was made. Notice that in several cases auction closure decisions were made without all nine users entering bids; in fact, in one case, only four bids were received before a user was assigned. This illustrates the tradeoffs being made between waiting for further good bids and giving the assigned user sufficient preparation time.

The column headed "Best bid" shows the winning bid. A winner typically bid $< 1, 1 >$, i.e., indicating that the user is both capable and willing to do the presentation – a high quality bid. When there was only one such bid,

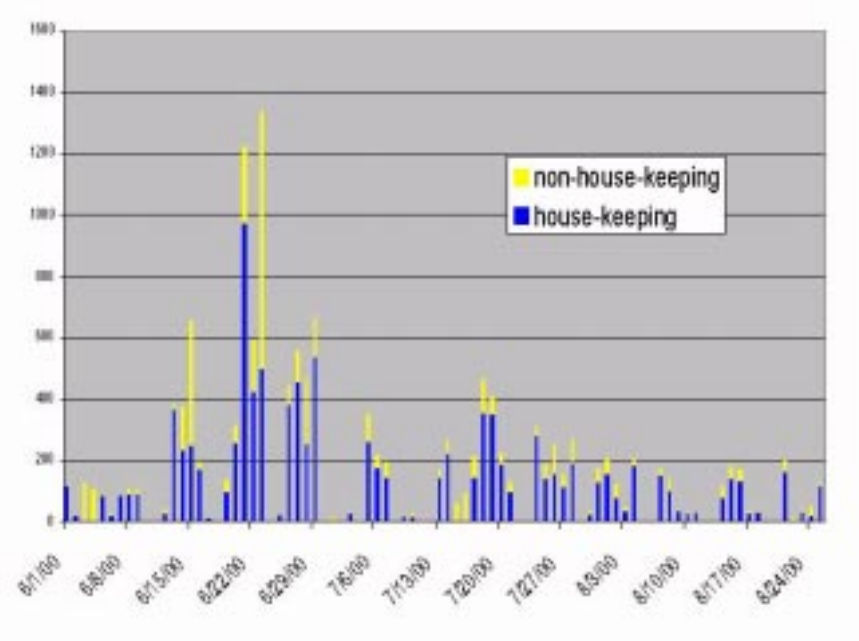

Fig. 6.20: *Number of daily coordination messages exchanged by proxies over three-month period. The y-axis shows the number of messages exchanged between proxies and the x-axis shows the date.*

the MDP could confidently choose the winner, otherwise it would wait for user input. Interestingly, the winner on July 27th made a bid of $< 0, 1 >$, i.e., not capable but willing. The agent team was able to autonomously select a user for the role despite the bid not being the highest possible, illustrating its flexibility.

Finally, the columns headed "Winner" and "Method" show the auction outcome. The final column indicates whether the auction was closed autonomously or manually by a user. In four of the six auctions shown, a winner was autonomously selected. The two manual assignments were due to exceptional circumstances in the group (e.g., a first-time visitor). Such exceptional circumstances are unlikely to be anticipated by even the most mature of systems, hence their existence highlights the need for AA at the team level.

Date	No. of bids	Best bid	Winner	Method
Jul 6	7	1,1	Scerri	Human
Jul 20	9	1,1	Scerri	Autonomous
Jul 27	7	0,1	Kulkarni	Autonomous
Aug 3	8	1,1	Nair	Autonomous
Aug 31	4	1,1	Tambe	Autonomous
Sept 19	6	N/A	Visitor	Human

Tab. 6.6: *Results of the team auction for the presenter role at the group's weekly meetings, during a three month period.*

6.6 Personal Assistants

Personal assistant agents have been applied to a variety of tasks, generally with the core goal of giving a user more powerful, flexible control over their interactions with their computer. We briefly review a small cross section of related work to show some of the directions different research is pursuing.

Some personal assistant research has focused on filtering information for users, e.g., (Collis et al. 2000, Horvitz et al. 1998). Faced with the huge amount of information the digital age produces, personal assistants filter information, attempting to find information relevant to their user's current tasks and interests. This type of information processing agent has become particularly interesting given the wealth and (relative) accessibility of the information on the Internet(Etzioni & Weld 1997).

Another type of personal assistant agent represents the user in some situation (Maes 1994*b*, Sen et al. 1997). For example, representing the user in negotiations over services or schedules. This type of agent reduces the user's workload by doing parts of their job for them.

Many different approaches have been taken to building useful personal assistant agents. A group at Microsoft have looked extensively at information type filtering agents (Horvitz et al. 1999). The agents used decision theoretic techniques to weigh the potential costs and benefits of bringing certain information to the user's attention.

The COLLAGEN project has looked at building software modules that encapsulate the functionality required for human-computer discourse (Rich & Sidner 1998). The hypothesis behind that work is that human-agent interaction should be similar to human-human interaction. A discourse between the agent and user is based on a formal model of the task being performed, though this formal model is separate from the inner workings of the agent.

Maes (1994*b*) have approached the problem of building intelligent personal assistants by creating many simple agents for simple tasks. Each agent learns when its behaviour is appropriate and activates itself accordingly. To the user, the emergent complexity of the simple agents and the complex task gives the impression the personal assistant is a cohesive, goal-directed entity (rather than the reality, which is simply a set of simple, reactive modules, not unlike EASE).

An approach closer to existing direct manipulation ideas is to explicitly delegate tasks to an agent for automation. In some cases, these "personal assistants" are effectively scripting languages, albeit with an intelligent interface. The approach is analogous to the way a human manager delegates tasks to human sub-ordinates. Milewski & Lewis (1997) points out that delegation is not always successful in the human world and appropriate care needs to be taken when mapping the idea to human-computer systems.

6.7 Summary

Living and working with the E-Elves has convinced us that AA is a critical part of any human collaboration software. In such systems, $\mathcal{R}$, implemented in software, transfers decision making responsibility between the humans and agents to best avoid the potential mistakes of agents without overburdening users. The transfer of autonomy occurs both at the individual and team level.

A key to the straightforward implementation of AA is the decision policy produced by the autonomous software (Friday or a team). $\mathcal{R}$ uses the

policy, which provides a complete description of the way a situation will be handled autonomously, to produce a corresponding autonomy policy, which specifies when the user will be asked for input. An explicit model of the user and environment, on which both the core Friday decision making and AA decision making rely, provides an efficient mechanism for sharing information. High level teamwork functionality eases the task of building the AA by allowing $\mathcal{R}$ to focus on a reduced subset of issues and letting the teamwork framework look after the team details.

In the next chapter we evaluate aspects of the design that are specifically related to the guidelines and assess their impact on the AA.

7. EVALUATION

In the two previous chapters we have presented two fully implemented Adjustable Autonomy (AA) systems. To a large degree both systems were implemented according to the guidelines in Chapter 4. The fact that the AA in both systems performs the task required of it and was fairly straightforward to build serves as a broad endorsement of the guidelines – especially since the systems are so different. That is, if the AA was straightforward to build when the agents were designed as per the guidelines then at worst the guidelines are not completely wrong.

Recall that in Chapter 4 we argued that guidelines were the appropriate way of extracting the experience we had gained building AA systems. The question remains, however, as to whether the particular guidelines presented actually capture the important aspects of the design, i.e., *have we correctly identified the aspects of the designs that made the AA simple to implement?* In this chapter we attempt to strengthen the case for the guidelines by looking at each guideline individually, determining the features that following the guideline brought to the agents and assessing the impact of those features on the amount of effort required to build the AA. By showing the generally positive effects of features resulting from the guidelines we show the utility of those guidelines.

Notice that we are not trying to claim that these systems have any special, unique or exciting features that made implementation of AA straightforward. In fact, many of the system features discussed below are quite normal and common, though not ubiquitous. The idea is to point out which of these features are useful when implementing AA and, conversely, which features hinder the development of AA and show that those desirable features are led to by the guidelines.

7.1 Method

This chapter aims to evaluate the guidelines in the following way. First, we identify a selection of *agent features* of the agents in the two systems that come about from following each of the guidelines. The guidelines and some

of the features they led to in each of the implementations are summarised in Table 7.1. Then we evaluate the impact of each of those features on the implementation of AA. By showing that the agent features encouraged by the guidelines made the AA easy to implement we show the utility of the guidelines.

We present the evaluation guideline by guideline. For each guideline, we show an EASE agent feature and an E-Elves agent feature that result from following the guideline under consideration. For each feature we show the AA facility that was built leveraging that feature. Finally, for each guideline, we present a counterexample which shows some feature, in one of the systems, that violates the guideline and discuss the implications of that feature on the AA.

Guideline	EASE	E-Elves
Explicit Information Guideline	Agents, negotiation	User organisation model
Design Information Guideline	Agent hierarchy	Potential costs and benefits represented in MDP, uncertainty explicitly represented
Software Engineering Guideline	Parameterization of agents	Separation between team and individual reasoning
Deterministic Execution Guideline	Agent organisation	Action policy
Explicit Behaviour Guideline	Agents and named constants	Action policy
Building Blocks Guideline	Agents	Separate handling of each episode
No Extra Mechanisms Guideline	None required	None required
Design Expecting Failures Guideline	Success and failure messages	STEAM "protects" others from individual decisions

Tab. 7.1: *Summary of the features in the two implementations led to by each of the guidelines.*

In some cases, due to other design constraints (e.g., efficiency), we have

violated the AA guidelines when designing the agents. In such cases, we can look directly at the implications of the violation on the implementation of AA, i.e., we can see whether the violation of the guideline did indeed hinder the AA development. In general, we found that when we violated the guidelines, AA facilities were more difficult to build on the resulting features.

This is not an exhaustive list of the AA facilities of the systems nor of the agent features underlying them. Rather, it is an illustrative selection of features that specifically highlight the ideas underlying the guidelines.

7.1.1 Scope of Evaluation

An ideal evaluation of the guidelines would be to have a number of separate teams of developers build agents for the same type of AA systems. Different teams could follow and violate different combinations of the guidelines in the design of the agents. We could then compare the effects of the different design decisions on the implementation of AA. Unfortunately, this sort of evaluation is beyond the scope of this work.

An examination of the literature provides some anecdotal evidence of the types of agent features that make AA easier or harder to build. (Some of these anecdotes are presented in Chapter 3 and others are presented where most relevant, either in the system descriptions or in this chapter.) Hence, one way to evaluate the guidelines would be to examine reported AA implementations to identify whether adherence to the guidelines was genuinely useful. However, the limited number of AA implementations reported, the diversity of their application domains and the lack of detailed design and evaluation information makes such an approach infeasible.

One weakness with our chosen evaluation method is that we ask that the reader take on faith that the AA facilities that we present are *easy to build* and are genuinely *useful*. In Chapters 5 and 6 we have highlighted some features of the systems which we believe would make them genuinely useful in real environments. Anecdotal evidence was presented in Chapter 5 which showed that EASE AA facilities were at least useful for the debugging and rapid-prototyping of some types of actors. In the E-Elves case, we have used the AA in a real environment and consider it an essential part of the human collaboration system. Furthermore, we have tried to show that the AA facilities of the systems were easily implemented given the features of the systems.

However, going further and *proving* that the AA facilities the systems provide are actually useful would require going from research prototypes

to complete implementations of AA (including complete Human-Computer interfaces and autonomy reasoning ($\mathcal{R}$)) then conducting extensive testing in real-world, unstructured environments. Without such use we cannot say, definitively, that the AA facilities are genuinely useful. Such testing is beyond the scope of this work. Equally infeasible is *proving* that the facilities were genuinely straightforward to build.

Any environment we "create" to perform structured experiments in will be "rigged" to invoke the AA facilities we provide. That is, it would be impossible to design an "artificial" scenario that we did not (consciously or sub-consciously) design with the AA facilities of the systems in mind. But AA is supposed to be useful for handling cases the designer did not consider, so the fact that we designed the experiment with our AA facilities in mind defeats the purpose of the experiment.

The problem of designing useful experiments to show the utility of an AA system is not limited to this work but is a general problem for all AA developers. AA is fundamentally used for doing things not anticipated by the software (i.e., the software designer). How can we (as designers) do useful, repeatable, controlled experiments on the ability of software to do things we did not expect. The only really effective tests of AA capabilities will come from real-world use of the system. The best we can do here is point out that the literature offers much motivation for AA with the sorts of AA facilities we have developed, hence we believe the AA we have developed would be genuinely useful in the real world.

Even if the systems were fully implemented and tested under real world conditions a problem would still remain for evaluation of this particular work. The problem is that it would be difficult to ascertain which aspects of the results were due to the agent design (and in turn to the guidelines) and which were due to the human-computer interface design or the skill of the user or the particular events that occurred, etc. For example, say we created a system that was found to work "perfectly". How can we properly ascribe credit to the agent design, the implementation of $\mathcal{R}$, the human computer interfaces, the simplicity of the domain, etc. for the (amazing) success? In fact, we would probably be back where we started, relying on qualitative evaluation to assess the utility of the guidelines. Given the lack of viable alternatives, we ask the reader to take our assessment of what facilities are useful for AA at face value.

This evaluation in no way shows that these guidelines are the only ones that will lead to good agent designs for AA. Other guidelines, following quite distinct principles might lead just as well (or better) to good agent designs. This is OK, as our contention is simply that these guidelines lead to good

designs for making AA simple to implement, not that this is the only (or even best) way.

One might argue that some of the points made below are contrived, i.e., that special cases have been chosen to show the strengths of the guidelines and cover their weaknesses. By showing selected examples we illustrate that the guidelines are useful in at least some cases and, at least, are not guaranteed to lead to agents that are hard to use in AA systems. We have tried to show the underlying reasons why the features, resulting from the guidelines, have made the AA easy to implement. The underlying reasons show, to some degree, why it is reasonable to expect that the guidelines will be more generally applicable.

7.2 Explicit Information Guideline

The Explicit Information Guideline advocates making as much of the agent's decision making processes as explicit as possible in that agent. Because it is important for $\mathcal{R}$ to understand what an agent is doing and why it is doing it, it is important for as much of an agent's decision making as possible to be made available via the AA information collection component ($\mathcal{I}$). The basic idea behind this guideline is that explicitly represented information is easier to extract and use in $\mathcal{R}$ than is implicitly represented or obfuscated information.

7.2.1 EASE and the Explicit Information Guideline

In EASE, agents explicitly capture the goals of the actor. Contracts between agents explicitly capture the relationships between goals. Hence, the goals and intentions of the actor can easily be presented to the user by a straightforward extraction of the explicit agent hierarchy from an actor. The Boss, as shown in Figure 5.19, displays the extracted hierarchy without any post-processing. Furthermore, the displayed agent organisation actually provides a useful, understandable metaphor which users can easily understand (Travers 1996). This means that translating or interpreting the organisation in a different way is not necessary. Thus a "good" AA facility has been built straightforwardly on the EASE architecture.

The negotiation mechanism explicitly represents the conflict resolution process of the actor. The details of the explicit negotiation can be easily extracted and visualised. A simple interface shows the status and history of the negotiation to the user (see Figure 5.32). From the interface the user

can identify conflicts between goals and see how the agents resolve those conflicts (see Section 5.5.2 for details).

Together the agent organisation and the explicit negotiation capture much of the agent's reasoning in an explicit manner. The visualisations of the goal-hierarchies of the actor and the goal conflict resolution process give a user a solid understanding of the actor's overall behaviour. If we assume that understanding the actor's behaviour is important to AA (Brann et al. 1996, Kortenkamp et al. 2000) then these visualisations are useful.

Moreover, building these tools is trivial *because* the goals and conflict resolution *are* explicitly represented. Consider an architecture where goals were not explicitly represented, e.g., a single layered behaviour based architecture. In such an architecture the "goal-directedness" of the agent is an emergent property of the interactions between simple behaviours and the environment. Hence, $\mathcal{I}$ needs to perform a far from trivial computation to get useful goal information to the user. This would be clearly more difficult than extracting and presenting the explicitly represented goals.

Therefore the explicit representation of the actor's behaviour, in the form of an explicit agent organisation and explicit agent negotiation, has led to easily giving the user an accurate, understandable picture of the current state of the actor's reasoning. Hence, we can argue that because in EASE useful visualisations were straightforward to build on actor features that resulted from following the Explicit Information Guideline, then that guideline is useful.

7.2.2 E-Elves and the Explicit Information Guideline

The E-Elves has an explicit model of the user and the environment upon which all reasoning is based. The model contains information about the user's current location, probability distributions of likely developments in different situations, the costs and benefits of being in different situations, the likely impact of different actions and so on (see Section 6.4.1). Much AA reasoning in the E-Elves is based on this model via straightforward MDP value iteration techniques.

$\mathcal{R}$ uses the explicit, shared model in its own reasoning. Being able to use this model includes being easily able to leverage the sensor data accumulated by Friday. Friday and $\mathcal{R}$ use basically the same information to reach decisions, though the types of decisions they reach are different. Thus, the information contained in the model is critical to $\mathcal{R}$. Because the model is shared, $\mathcal{R}$ does not need to create, maintain or update its own model of the user and environment. The use of an explicit model is clearly advocated by

the Explicit Information Guideline. If the model were not easily available $\mathcal{R}$ would need to have its own world model. Hence, the feature advocated by the guideline makes the task of building AA simpler.

Notice, that the use of an explicit model that could be shared between Friday and $\mathcal{R}$ was not an inevitable feature of a human collaboration system. In fact, an earlier implementation of the E-Elves used C4.5 (Quinlan 1993) which does not have an explicit representation model of the user or environment, hence the rules for autonomy decisions shared very little information with the rules for making decisions about actions. Hence, the use of an explicit model, as advocated by the Explicit Information Guideline, is a useful but not an inevitable system feature.

7.2.3 Violation of the Explicit Information Guideline

Not all aspects of the E-Elves design follow the Explicit Information Guideline guideline. In particular, the MDP does not explicitly capture the *high level strategy* that following a particular sequence of actions in the MDP corresponds to. That is, Friday's high level strategy is not explicitly represented as advocated by the Explicit Information Guideline. For example, Friday has an implicit high level strategy to *stall for time* when the location of the user is unknown shortly before a meeting. The aim of the strategy is to try to wait as long as possible before taking decisive actions, like cancelling the meeting, in the hope the user will turn up. This might be implemented by requesting a short delay, to give the user time to arrive, before requesting a *cancel* if they still fail to arrive.

Ideally, we might like to have $\mathcal{R}$ decide whether *stalling for time* is an acceptable strategy and potentially decide whether to ask the user to approve use of the strategy. However, because the high level strategy is not represented explicitly by Friday, $\mathcal{R}$ is forced to deal with the low level details rather than the high level strategy, i.e., $\mathcal{R}$ must decide whether the user should be asked about each individual action rather than be asked whether the high level policy is acceptable. Clearly, under some circumstances this is a less desirable approach.

It may be possible for $\mathcal{I}$ to infer the high level strategy that Friday is following from the low level actions Friday has planned. Clearly, however, this requires more development effort for $\mathcal{I}$ than if the strategies were explicitly represented, i.e., some development work would need to be done to "reconstruct" the policies. Hence, a potentially useful AA facility is more difficult to build due to a system feature resulting from a violation of the Explicit Information Guideline. In at least some cases, then, violating the

Explicit Information Guideline makes implementing AA more complex.

7.3 Design Information Guideline

The Design Information Guideline advocates representing design information in an agent regardless of whether it is (believed to be) necessary for agent reasoning. The design information gives $\mathcal{R}$ insight into the workings of the agent further than simply what the agent is doing and why it is doing it. Such insights are important for reasoning about autonomy changes.

7.3.1 EASE and the Design Information Guideline

Intelligent actors created with EASE have a hierarchical agent organisation, corresponding to their goal hierarchy, explicitly represented at run-time. The explicit representation of the hierarchies has no effect on the behaviour of the actor at run-time. Explicit representation of the hierarchy is a design decision – one which extracts a high computational price. It is possible to "compile" out the hierarchy into a "flat" system (which would be far more efficient), however by doing so important design information would be lost, although the observable actor behaviour would be the same.

The hierarchies capture the designer's abstractions and decompositions – essentially the design knowledge of the actor's behaviour. Compiling those hierarchies out would be similar to compiling object oriented programs into procedural code. Many of the design decisions which made the object-oriented code easier to understand would be lost.

At run-time, the explicit hierarchical representation leads to the ability to easily show goal hierarchies to the user. The *hierarchies* provide a built-in abstraction mechanism which mirrors the abstractions and decompositions the designer made at design time. Since abstractions are used to manage complexity and the abstractions are available at run-time, the user can leverage the same mechanisms for managing complexity as the designer did. That is, the same abstractions that helped the designer manage the actor's complexity at design time can allow the user to manage the actor's complexity at run-time.

The Boss, i.e., the tool that shows the hierarchy to the user, easily allows the expansion or hiding of whole goal hierarchies. This allows the user to view the goals of the actor at various abstraction levels, depending on the information they require for their decision making. Figures 5.30 and 5.31 show The Boss with the hierarchies expanded and collapsed, respectively.

Notice, that it is not possible for $\mathcal{I}$ to "reconstruct" the abstractions at run-time if the information is not embedded in the run-time actor, i.e., it is not possible for $\mathcal{I}$ to take a flat system and produce useful abstractions. Hence, without the explicit representation of hierarchies at run-time the user would either have no opportunity to use the more abstract design information or they would need to infer the abstract information themselves. Hence, the explicit representation of the abstractions at runtime, as advocated by the Design Information Guideline, leads to useful features for the straightforward creation of AA.

As a second example, notice that the names used by the designer at design time are, for the most part, retained at run-time. The designer's naming conventions provide invaluable information about the *intended functionality* of some aspect of the actor specification. Representing the names at run-time means they can easily be extracted for use by the user. Obviously, the "shape" of the goal-hierarchy is of little use without knowing what the goals are. For example, knowing goal A has sub-goals B and C is less useful than knowing the goal *be happy*, has sub-goals *drink beer* and *watch football*. How useful the names are to the user relies on how well the designer named the goals in the first place.

7.3.2 E-Elves and the Design Information Guideline

Friday is primarily responsible for making autonomous decisions that are expected to save the user time. It has a set of actions (including "wait") and chooses one of those actions to take. $\mathcal{R}$ needs to understand how safe, reliable and likely to cause harm an action is to decide whether Friday should take the selected action autonomously or not. Such decisions are not only based on the expected cost of an action, i.e., because some costs are or might be incurred does not necessarily mean that $\mathcal{R}$ should withdraw autonomy because some costs are unavoidable.

A variety of mechanisms could have been used to implement Friday's decision making but an MDP was chosen. One important reason the MDP mechanism was used is that it intrinsically provides explicit information about the potential costs and benefits of an action and the likelihoods of those costs or benefits coming about. This is essentially *design information* that is explicitly represented.

At the AA level the cost/benefit information and especially the associated uncertainty information allows $\mathcal{R}$ to make good autonomy decisions. The autonomy decisions are fundamentally based on the explicitly represented cost/benefit and uncertainty information provided by the MDP

mechanism. If Friday, via the MDP mechanism, could not supply the AA with this detailed information $\mathcal{I}$ would have to essentially repeat Friday's calculations, albeit in a manner which produced the required uncertainty information. Hence, if Friday used an architecture where costs/benefits and uncertainties were not explicitly represented, the task of building AA would be strictly more difficult.

Since the explicit representation of the design information is advocated by the Design Information Guideline and results in features helpful to the implementation of AA we can say that at least in some cases following the Design Information Guideline is advantageous.

7.3.3 Violation of the Design Information Guideline

An example of a violation of the Design Information Guideline can be seen in the EASE architecture. There may be some situations an actor can get itself into where the *designer* knew at design time that the actor behaviour was going to be unsatisfactory. This information is not represented at run-time, in violation of the Design Information Guideline. For example, the designer may know that when the enemy aircraft is directly above a simulated pilot, the pilot will get "confused". In cases where behaviour is weak or incorrect, the user, using AA facilities should step in and help with decision making. However, because the designer knowledge is not represented at run-time, $\mathcal{I}$ cannot inform the user at run-time of potential problems the designer knew would occur. Clearly, this handicaps the user's ability to make good decisions.

If $\mathcal{I}$ could "warn" the user that the actor was getting into a situation in which the actor designer knew the actor's behaviour would be unacceptable, the user could react more quickly (perhaps pre-emptively) and take over decision making. Adequate warning could only be given if the limitations of the actor were readily available to $\mathcal{I}$. Such information cannot be "reconstructed" by $\mathcal{I}$ at run-time, hence not representing limitation information puts a fundamental limitation on the capabilities of the AA.

Thus, we can conclude that there are at least some cases where the failure to represent design information in an agent hampers effective AA. In turn, this shows that in at least some cases, violating the Design Information Guideline is detrimental to AA.

7.4 Software Engineering Guideline

The Software Engineering Guideline advocates following good software engineering practices when building agents because those practices tend to lead to easier to understand and easier to change agents. Being easy to understand and easy to change are useful properties of an agent in an AA system as a large part of AA is understanding and changing agent behaviour online.

7.4.1 EASE and the Software Engineering Guideline

A variety of aspects of EASE are specifically designed to encourage good engineering practice. For example, the use of agents encourages modularity, the end-user system for specifying information processing encourages re-use of calculations and the use of managers encourages abstraction and hierarchical decomposition. For the purposes of this discussion we focus on two features which exemplify the difference good software engineering can make to AA: good naming conventions and agent re-use via parameterization.

$\mathcal{I}$ can use good designer names to give useful extra information about the actor to the user – provided those names are used (and available) at run-time. For example, if a particular satisfaction function was named *Avoid aircraft by flying underneath it*, the user, seeing an agent using the satisfaction function would know what the agent was trying to do (or at least what the designer intended it to do). On the other hand if the function was called, say, "S3" then either the user (or $\mathcal{I}$) would need to infer the aim of the agent from its actions – clearly a more difficult task (Carmel & Markovitch 1998). Hence, the software engineering principle of good naming conventions makes AA easier. Notice, that it is also important that those good naming conventions are preserved at run-time so the user has access to them (as per the Design Information Guideline).

Parameterization of a "computation" allows the "core" elements of the computation to be re-used in different parts of some software, with the specific required behaviour obtained by instantiating the parameters appropriately. Parameterization is possible with most parts of an EASE actor including satisfaction and activation functions, state-machines, agents and whole hierarchies of agents.

The task of implementing changes in autonomy and behaviour (i.e., $\mathcal{A}$) is, in general, difficult because changing software is intrinsically difficult. However, the task is made very much easier if there are parameterized components that can be instantiated to implement new behaviour or parameters that can be changed to modify existing components. For example, say the

user wishes a simulated pilot to attack a particular target. To explain from "first principles" how to attack a target would be very complex and prohibitively time consuming at run-time, but if a parameterizable agent with the right capability is available, the situation is far more manageable. This is similar to Bindiganavale et al. (2000) usage of UPAR's for instantiating natural language commands. It would not be feasible for $\mathcal{A}$ to "reconstruct" the parameterization at run-time, hence the limitation would be unavoidable.

Hence, using software engineering practices in an agent design helps both in the extraction of useful information and in the mapping of user requested changes to actual changes in an actor.

7.4.2 E-Elves and the Software Engineering Guideline

The E-Elves is a fairly well engineered piece of software, but one engineering feature stands out as very clearly leading to a simplified AA implementation. A basic design decision of the E-Elves was to separate team and individual decision making. Such modularity is in accordance with accepted good software engineering practice. Individual Fridays make local decisions based on local information, if those decisions affect the whole of the team then the decision is submitted to the team which decides as a group whether or not to implement the individual decision. The separation makes the implementation of decision making at both individual and team levels simpler because factors at the other level can be ignored. For example, when deciding on a meeting delay an individual can ignore the specific situations other users might find themselves in.

A separation into team and individual $\mathcal{R}$ components mirrors the separation in the decision making. The separation of $\mathcal{R}$ substantially simplifies the reasoning that needs to be done. For example, we described above how part of the autonomy decision at the individual level is based on the expected *value* of the user's input. The calculation of this value is relatively simple because it only takes into account local factors, i.e., how different Friday's and the user's decisions are likely to be, rather than team factors, e.g., what the effect on the team will be given possible differences in Friday and the user's decisions. Similarly, at the team level a decision about whether an auction should be closed autonomously is simple because it is based on abstract information such as the probability that the *team* would like to close the auction manually rather than whether each individual would like to close the auction manually. Notice that this separation is only feasible *because* there is a separation in actual decision making into individual and team components.

Moreover, the teamwork of the team is controlled by a general, relatively simple set of team rules which all agents understand and follow. This is a well engineered design because it encapsulates pieces of reasoning and keeps the connections between those pieces of reasoning at a high level. The high level teamwork means that $\mathcal{R}$ can reasonably ignore factors from other components because the team work model handles the low level issues automatically.

Hence, as with EASE, the good software engineering practices used in the design of the E-Elves made AA simpler to implement.

7.4.3 Violation of the Software Engineering Guideline

Above, in the descriptions of the EASE actor software engineering features that make AA straightforward to implement, we mentioned that EASE *encourages* rather than *enforces* such practices. As an counterexample of the Software Engineering Guideline we consider the implications of the designer not following software engineering practices as they should. In particular, take the case where, for efficiency (and perhaps simplicity), a designer may not parameterize an agent definition. For example, rather than building a generic *attack target* agent with the target as a parameter, a more specific *attack fighter aircraft* agent is created. At run-time, when the user requires that the simulated pilot attack, say, a ground installation, it may be very difficult to achieve the correct pilot behaviour because of the detailed description of the behaviour the user must rapidly provide. Hence, this is a clear example of a case where violating software engineering principles makes the realization of effective AA more difficult.

7.5 Deterministic Execution Guideline

The Deterministic Execution Guideline advocates using deterministic algorithms because such algorithms make an agent's behaviour predictable. Being predictable makes it easier for $\mathcal{R}$ to reason about the impact of autonomy changes as well as allowing it to work out what the agent will do next if allowed to act autonomously.

7.5.1 EASE and the Deterministic Execution Guideline

The workings of the EASE agent organisation are strictly deterministic. The action(s) that a manager will take when receiving messages from contractees

are clearly known in advance and work deterministically. The way an engineer will negotiate in a negotiation is also clearly known in advance (although the outcome of a negotiation is non-deterministic – see below). The impact of different environmental events can be calculated in advance. Because all the changes that will occur in the agent organisation are known in advance, it is straightforward to simulate the dynamics of the organisation.

The determinism means that at the AA level it was trivial to build a simple tool, called the Implications Viewer (see Figure 5.20) that mimics the behaviour of the organisation. The tool allows the user to investigate the impact of different events, including changes they might make, on the structure of the organisation. The structure of the organisation is very important because it captures the goal hierarchy of the actor. Thus, the Implications Viewer, allows the user to see the impact of potential changes on the goal hierarchy of an actor and therefore see the effects of their proposed changes on the behaviour of the actor.

Without the determinism of the underlying agent organisation, as advocated by the Deterministic Execution Guideline, it would not be possible to be sure what was going to happen when a change was made. Hence, any change to the organisation would carry with it an element of luck. In turn, this implies that the Implications Viewer could not have been constructed in a way that would give the user a consistently accurate picture of the changes that will occur in the organisation. The determinism of the organisation leads to an easy to build facility that provides valuable AA functionality. Thus, we can conclude that at least in some case it is advantageous when building AA to have agents with features resulting from following the Deterministic Execution Guideline.

7.5.2 E-Elves and the Deterministic Execution Guideline

When Friday first detects a situation where it might take some actions, e.g., a meeting that may have to be delayed, it develops a *policy* that dictates what actions it will taken in each possible state it finds itself in. This policy completely dictates Friday's behaviour for the rest of the episode. Hence, Friday's behaviour is deterministic and predictable and thus $\mathcal{R}$ knows well in advance what Friday will do in all situations.

$\mathcal{R}$ uses Friday's predetermined policy to create its own *autonomy policy*. Friday's policy takes some of the "guess work" out of creating an autonomy policy because $\mathcal{R}$ knows what Friday will do (within the bounds of the unpredictability of the user and environment).

Sometimes autonomy decisions will not only be based on the action Fri-

day intends to take immediately, but also on the action it intends to take next. For example, consider the case when some time before a meeting the user is not at the department. If $\mathcal{R}$ knows only that for now Friday intends to wait, it may conclude that there is significant likelihood that the user will want to ask for some delay (because doing nothing is the wrong decision) and decide to withdraw Friday's autonomy. However, if $\mathcal{R}$ also knows that Friday *will* request some delay to the meeting if the user has still not arrived closer to the meeting time, $\mathcal{R}$'s decision might be different. In particular, it might decide that Friday's *policy* had sufficiently low risk so as to justify allowing it to continue autonomously. Hence, the availability of Friday's entire policy to $\mathcal{R}$ improves its ability to make good autonomy decisions.

Without the predictability of Friday's actions, $\mathcal{R}$ could not know in advance what Friday will do and, hence, autonomy reasoning would be weaker (as more uncertainty is inherent). It is the predictability due to the determinism that allows the high quality $\mathcal{R}$, hence the Deterministic Execution Guideline is a useful guideline.

7.5.3 Violation of the Deterministic Execution Guideline

An examination of EASE provides an example of how violating the Deterministic Execution Guideline leads to difficulties implementing AA in a straightforward manner. The negotiation mechanism used by the engineers to come to a decision on what action an actor should take is stochastic, i.e., there is some degree of non-determinism. A factory, the entity administering the negotiation, picks some action from the action space, checks each engineer's *satisfaction* with that action, keeping the action if it is the best action so far and discarding it otherwise and then begins the cycle again. Actions are chosen according to an intelligent algorithm with a significant stochastic element. The eventual action taken by the actor is the "best" of those sampled during the negotiation cycle. For large action spaces only a small percentage of actions may be sampled in each cycle.

In some cases the basic options are indistinguishable to the actor and one is picked randomly. For example, when a simulated pilot avoids an obstacle by flying around it, it will often have two options – to go left or to go right around the obstacle. However, there may be factors outside the pilot's knowledge (like other obstacles) that make it interesting or important for the *user* to know whether the pilot will go to the left or right. However, because the decision will be made by the negotiation algorithm which is non-deterministic, the user cannot know what the actor will do.

In practice, actions taken by the actor are "fairly" predictable, but this

is primarily due to the fact that "global maximums" in the search space of actions for our particular domains are usually fairly large and fairly reliably found by the negotiation algorithms. However, basing interfaces on "rules of thumb" like this is likely to be a dangerous practice because unexpected and problematic situations, i.e., those where AA is most likely required, will be more likely to break the "rules of thumb". That is, almost by definition, unexpected and problematic situations do not follow the same "rules" as more "normal" situations, hence anything that relies on standard rules holding will break when unexpected situations occur. Hence, AA tools based on rules of thumb are an inherently bad idea, because the situations where they are most likely to be useful are those where they are most likely to fail. The key point is that non-determinism hinders the development of effective, reliable tools for AA because the future actions of an actor cannot be presented.

Thus, we can see that violating the Deterministic Execution Guideline can lead to significant problems when implementing AA.

7.6 Explicit Behaviour Guideline

The Explicit Behaviour Guideline advocates using architectures where behaviour is represented explicitly and in a language close to that of $\mathcal{R}$ so that autonomy changes can be easily implemented. The underlying idea is that the closer the behaviour representation "language" used by $\mathcal{R}$ is to that used by an agent the easier the translation between the "languages" of the two. Since a translation between the languages needs to be done in order to allow $\mathcal{A}$ to realize behaviour changes, making the translation simple makes the implementation of that aspect of AA straightforward.

7.6.1 EASE and the Explicit Behaviour Guideline

EASE represents goals explicitly in an actor via the use of agents and represents more abstract properties of actor behaviour with named constants. In a well designed actor the goal hierarchy and the named constants might explicitly represent most aspects of the actor's behaviour. The agents and the named constants constitute the "language" of the EASE actor. For example, a designer might have a named constant *desperation* which is used in various satisfaction calculations, activation functions and transition conditions of some RoboCup actor. Changing the value of the constant changes the behaviour of each of those functions. For example, the named constant

desperation might change the distance to the ball at which a RoboCup player will transition between states for *watching* and *chasing*.

The fact that the behaviour is explicitly represented is, in itself, useful because mapping changes in behaviour to an explicit representation is simpler than mapping to some mechanism where the behaviour is only implicitly represented (like a neural network). However, the key to the practical effectiveness of this guideline is whether or not the things that are explicitly represented in an actor are the things that the user wants to manipulate. For example, *dribbling* is more difficult to manipulate if represented as *small kick* and *run to ball* than if represented directly as *dribbling*. Similar issues are likely to arise in the JACK system where natural language input is to be mapped to internal constructs (Bindiganavale et al. 2000) – if appropriate internal constructs exist corresponding to the natural language statement the behaviour change is easily made, if not, achieving the correct result is difficult.

Effectively, the problem boils down to how difficult the "translation" between what the user wants and what the actor "knows" is. In our use of EASE we have usually found that most of the things we want to change are explicitly represented and are therefore easily changed. For example, it was straightforward to force our test RoboCup players to *chase*, *pass* and *shoot* because such behaviour had an explicit representation in the actor. There are times, however, when the mapping from the change we required of the actor, to the things that were explicitly represented was not straightforward. For example, a particular RoboCup actor would periodically turn to check the location of the ball while running back to position. This *periodic check* was not explicitly represented in the actor, rather it was an "emergent" interaction between the *know where the ball is* and *get to position* agents. Changing the periodicity of the *periodic check* involved the (relatively) difficult task of altering the environmental priority functions of the two simpler agents.

Thus, it can be seen that an explicit representation of behaviour, in at least one case, makes the implementation of AA simpler.

7.6.2 E-Elves and the Explicit Behaviour Guideline

The feature of the E-Elves that most clearly exemplifies the Explicit Behaviour Guideline is Friday's creation of a policy which $\mathcal{R}$ uses to create an autonomy policy. The value of this feature to the AA has been discussed above and will not be repeated here. In summary, the explicit representation of the policy substantially simplifies the design of the AA. Since such

a feature results from adhering to the Explicit Behaviour Guideline we can conclude that the guideline is at least sometimes useful.

7.6.3 Violation of the Explicit Behaviour Guideline

It has been noted above that the MDP mechanism does not have an explicit representation of high level strategies, e.g., it does not represent the strategy *stall for time*. Because Friday does not represent high level strategies, there is no point in $\mathcal{R}$ reasoning about them either because $\mathcal{A}$ could not make appropriate changes to a policy in any case. For example, say the user decides that *stalling for time* is an inappropriate strategy. $\mathcal{A}$ needs to decide whether a particular action in Friday's policy is part of the *stalling for time* strategy in order to know whether to allow it or not. Such a decision is not simple. A five minute delay in the policy may be an action taken when it is known the user will be arriving five minutes late or it could be part of the stall for time strategy (or a bit of both).

Hence, because it is not feasible to reason about high level strategies, $\mathcal{R}$ is forced to work at the level of individual actions – a clear limitation. Because this limitation comes about due to a violation of the Explicit Behaviour Guideline we can see the utility of the Explicit Behaviour Guideline.

Other architectures where the reasoning of the agent is far more sophisticated, e.g., Soar (Rosenbloom et al. 1991) or a first principles planner, e.g., (Pollack 1996, Veloso et al. 1995), will have a related difficulty. Planners use simple blocks to build up complex solutions to problems but will not, generally, "understand" the high level strategy of the overall plan. Likewise for Soar. For example, when faced with the problem of *getting bread* a planner might come up with a sequence of steps, e.g., *go to car*, *turn on engine*, ... , *take bread off shelf*, *pay cashier*, etc., that, if followed, will achieve the intended goal. We know that such a plan can be roughly described as *shopping*. If, instead, we want the agent to get bread via *baking*, it might not be straightforward to have $\mathcal{A}$ make appropriate changes because neither *baking* nor *shopping* is understood by the system. Thus, such architectures may not be good candidates for agents in AA systems, since they violate the Explicit Behaviour Guideline.

7.7 Building Blocks Guideline

The Building Blocks Guideline advocates building behaviour up from small units that are connected together in a well understood manner. It is our contention that such designs maximize the ease with which autonomy changes

can be made and the flexibility the system has to configure itself. The more flexibility $\mathcal{A}$ has the better $\mathcal{R}$ can configure the system for optimal performance.

7.7.1 EASE and the Building Blocks Guideline

The Building Blocks Guideline is clearly followed in EASE by having an actor broken up into many simple agents. Having many small blocks makes $\mathcal{A}$'s job of making some changes simpler. The smaller the "blocks" the more likely it is one or more "blocks" can be found corresponding to the aspects of the actor's behaviour that need to be changed. Conversely, larger blocks are more likely to cause problems as it will be less likely that a combination of blocks can be found corresponding exactly to the piece of behaviour a user is concerned with. For example, if complex behaviour is well decomposed into agents, then to stop pursuing some sub-goal might require stopping just one agent. Likewise, authority to pursue specific things can be given (or taken) more easily if the actor has a "block" corresponding to what the authority should be given for. Thus, a feature of the architecture advocated by the Building Blocks Guideline leads to greater flexibility and simplicity at run-time.

Having "small" building blocks is not an answer on its own. Goldman, Guerlain, Miller & Musliner (1997) noted that first principle planners were not appropriate as an agent architecture for mixed initiative systems precisely because the "building blocks", in this case planning operators, were too detailed and lacked the abstraction that users needed. The important difference between the building blocks of a first principle planner and those of EASE is that the building blocks in EASE are aggregated by the designer into more abstract blocks. These more abstract blocks allow the user to deal with more abstract "chunks" of behaviour when required but to also delve into the details. With the planner the user can *only* work with details.

7.7.2 E-Elves and the Building Blocks Guideline

In the E-Elves, each situation that Friday (or a team) is involved in is handled by a separate MDP. This means that each MDP needs to only consider a single situation at a time (the benefits of this have been discussed above). This breakup of decision making into "blocks" is in accordance with the Building Blocks Guideline. For the purposes of this discussion we are interested in how the modularity of the underlying decision making allows modularity in autonomy decision making. Without the underlying decision

making modularity, the AA would either have to separate out the situations itself or deal with several of them simultaneously.

At the AA level each decision making MDP in Friday is handled separately by the AA. So, for each policy Friday produces, $\mathcal{R}$ constructs a separate autonomy policy. As with the basic Friday decision making the computational complexity of AA for individual situations is substantially reduced by the separation. This makes $\mathcal{R}$ feasible because interactions between many different episodes do not need to be taken into account.

Notice that the breakdown into separate situations is similar to the philosophy of the breakdown for EASE. The underlying system has a modularity that allows the AA to deal with the system in a modular way. We show in the counterexample below, that if EASE actors are made up of big complex agents (instead of small simple ones) the AA task is more complex. Likewise for E-Elves, if the underlying system was handling everything at once then the task of making good autonomy decisions would be more difficult.

7.7.3 Violation of the Building Blocks Guideline

With EASE, it is possible to design actors where many aspects of the actor's behaviour are merged into a single engineer. For example, an obstacle avoidance engineer might be responsible for avoiding all other aircraft as well as the ground. In fact, all the behaviour of the actor could be summarised in a single engineer. The satisfaction function for that one engineer would need to be very complex, taking into account all the different objectives of the actor when calculating satisfaction values (something like a neural network might function in a similar manner). Although it is possible to create actors in this way, such designs, which clearly do not adhere to the Building Blocks Guideline are much more difficult to work with at runtime. It is far more difficult to manipulate a complex satisfaction function (or neural network) controlling a complex simulated pilot to make it, say, not avoid one specific obstacle than it is to remove a single agent from an agent organisation. This is because it is more difficult in a satisfaction function or neural network to isolate the data that need to be changed to elicit the appropriate behaviour change. Thus, in at least some cases violating the Building Blocks Guideline makes AA more difficult to implement.

7.8 No Extra Mechanisms Guideline

The No Extra Mechanisms Guideline was followed by both agent architectures. No extra mechanisms are used to implement any of the autonomy or

behaviour changes decided on by $\mathcal{R}$. In all cases, autonomy and behaviour changes are achieved via the same mechanisms that are used by the agents in their "normal" reasoning.

Both agent architectures were specifically designed with AA in mind so, no doubt, some aspects of the systems' designs were influenced by the needs of the AA. Some might argue this means the No Extra Mechanisms Guideline was followed only because we made the "extra mechanisms" part of the agent in the first place. However, this is still different to having mechanisms for doing "normal" agent reasoning and other mechanisms for implementing agent behaviour specified by $\mathcal{R}$ (and in fact thinking about AA from the earliest stages is precisely what we advocate). In the case of both EASE and the E-Elves the same mechanisms that do the "normal" reasoning of the agent implement the autonomy changes. Thus, the evaluation of this guideline is partly proof by existence – we have done it, therefore it is possible to do.

It is fairly clear that if we are free to choose between two systems that have the same functionality and one is more complex than the other, we would choose the simpler system. Since this will be the case if similarly functioning systems are built adhering and violating the No Extra Mechanisms Guideline, it seems clear that systems adhering to the No Extra Mechanisms Guideline are preferable. By implementing two effective AA systems without extra mechanisms we have shown it is possible, provided things are designed properly, to implement good AA without needing extra mechanisms. Thus, we can claim that the No Extra Mechanisms Guideline is a feasible and worthwhile guideline.

7.9 Design Expecting Failures Guideline

The Design Expecting Failures Guideline advocates building agents for AA systems with the expectation that any part of the agent could fail at any time. The guideline comes from the observation that often the effect of $\mathcal{A}$ on an agent is the same as if its components fail. The effect of AA is especially similar to a "normal" failure when $\mathcal{A}$ *reduces* authority or responsibility.

7.9.1 EASE Features

Behaviour based architectures are intrinsically robust to failure (Goldberg & Mataric 2000, Parker 1998). Since EASE is strongly based on behaviour based ideas its underlying principles ensure it handles failure robustly. Thus, EASE adheres to the Design Expecting Failures Guideline. Although the

basic nature of EASE provides a large degree of failure handling, for the sake of the evaluation of this guideline we concentrate on another feature of EASE which further increases its ability to handle failure gracefully.

When a contractee agent succeeds or fails in its assigned task, it reports that information to its contractor which can use the information in whatever way it likes. In particular, the contractor can use the information to handle the failure intelligently and gracefully. Recall, that when an agent is removed from the organisation by the user (i.e., by $\mathcal{A}$) it sends a message indicating either success or failure, just as if would have if it had *really* failed or succeeded. Thus, the mechanisms that allow the designer to easily implement graceful failure handling also, for free, provide graceful handling of autonomy changes.

An example of the usefulness of graceful failure handling for AA came up during the development of RoboCup agents for the 2000 World Cup. A bug in the low level information processing routines meant that sometimes a player did not realize it could no longer see the ball, which would mean the player would keep acting as if the ball was in the same relative position to itself as it was the last time the ball was seen. Strange effects, like the player chasing a "phantom" ball off the pitch or kicking a "phantom" ball over and over occasionally occurred. To get around the bug (during testing) we would often manually end the contracts of *chase ball* agents with failure messages. That is, we would assume decision making responsibility and specify that the player should assume its *chase ball* agent had failed. The contractor of the *chase ball* agent, perhaps a *striker* manager agent, would take actions to recover, perhaps by contracting another agent to (properly) locate the ball.

A nice feature of this example is that we were giving commands to the player at a fairly abstract level, i.e., "your behaviour of getting to the ball is failing". The player was dealing with the details of recovering from the problem in a context specific way, i.e., whichever contractor had contracted the *chase ball* agent would deal with the *chase ball* failure in an appropriate manner. The point is that the built-in failure handling effectively handles the "common sense" aspects of dealing with the requested change in behaviour. Once the actor knows that its behaviour is failing, it deals with it in an appropriate manner (provided the designer's specification is appropriate.) This is critical because it means the actor's behaviour is "sensible" when an AA change is made.

Thus, we conclude that adherence to the Design Expecting Failures Guideline, such as by the use of success and failure messages, is useful for a straightforward implementation of AA.

7.9.2 E-Elves and the Design Expecting Failures Guideline

The separation of the team and individual decision making has the desirable side effect of "protecting" the rest of the team from poor decisions by one Friday or its user (if the decision is not autonomous). That is, if a local decision is made that is not for the good of the team (whether for malicious reasons or because some information was not known) the STEAM rules protect the team by rejecting the decision. In effect, this is a failure recovery mechanism. If the team could safely assume that all the decisions that individual Fridays and their users were going to make were correct then the team could just accept all the decisions blindly.

From an AA perspective the protection afforded by the layering of the decision making makes $\mathcal{R}$ easier because problems with user input are easily and reliably handled. This makes $\mathcal{R}$ easier to implement because there is no need to question the user's requests, i.e., $\mathcal{R}$ need not be applied to the user. $\mathcal{R}$'s job at the individual level is just to decide whether or not a user should be consulted rather than also ensuring the user is not making dangerous decisions (which might require autonomy being taken from them).

Another example of failure handling in the E-Elves is the way that an autonomy policy specifically accounts for the possibility that the user will not provide input in a timely manner. Autonomy can be more confidently transferred to the user because it can be taken back later if the change does not produce the expected results. This makes autonomy reasoning simpler.

7.9.3 Violation of the Design Expecting Failures Guideline

As an example of a violation of the Design Expecting Failures Guideline we look at a very simple design decision in the EASE negotiation algorithm. The negotiation is designed to take advantage of a property of the domains we have so far used EASE in. In particular, the algorithm takes advantage of the property that whatever action is best at one cycle is often a good action at the next cycle. For example, if a RoboCup player is running in one cycle, the best option in the next cycle will often also be to run. Likewise, the heading of an aircraft in one cycle is likely to be similar to the best heading in the next cycle. To take advantage of this property, the negotiation algorithm is "seeded" at the start of each cycle with the action taken in the previous cycle. Recall that the algorithm keeps track of the favored action of the engineers and "executes" the best action at the end of the cycle. The action from the previous cycle is used as the starting value of the "best action" in the current cycle. An unintended consequence of this

is that if there are *no* engineers no *better* actions are found (the actor is equally ambivalent to all actions) and the previous action will keep on being selected and subsequently executed by the actor.

When a user takes over *all* decision making the first thing they do is remove *all* agents. Since the negotiation mechanism is seeded with the previous action, the actor will keep repeating the last action the engineers favored before they were removed. The resulting behaviour is often non-intuitive or even outright problematic, e.g., a RoboCup player with no agent organisation might repeatedly run (wasting stamina) or kick (regardless of the ball location) and a simulated pilot continues to dive (until it hits the ground) or turn (wasting fuel), etc. Such behaviour can be seen as a violation of the Design Expecting Failures Guideline because the "failure" of the agents, i.e., their removal, is not handled gracefully by the system. Further, it is clear that, in at least some cases, the resultant behaviour makes the AA harder to use because more care needs to be taken to avoid bad actor behaviour.

7.10 Evaluation Summary

In the above we have shown system features resulting from adherence to each of guidelines and the utility of those features to implementations of AA. Tables 7.2 and 7.3 summarises the EASE and ELVES system features respectively and the AA facilities those features support. We have also shown how violations of the guidelines can lead to system features that hinder the development of effective AA. Table 7.4 gives a summary of the example violations for each of the guidelines. While the evaluation cannot show definitively that the guidelines are worth following every time when developing agents for AA systems, the evaluation does show that in at least some cases, following the guidelines helps and violating the guidelines hinders, the development of effective AA.

Guideline	Agent feature	AA Facility
Explicit Information Guideline	Agents, negotiation	Goals, goal conflict resolution
Design Information Guideline	Agent hierarchy	Designer abstractions
Software Engineering Guideline	Parameterization of agents	Simple mapping of $\mathcal{A}$ changes
Deterministic Execution Guideline	Agent organisation	Implications Viewer
Explicit Behaviour Guideline	Agents and named constants	Goals and abstract aspects of behaviour
Building Blocks Guideline	Agents	Ability to identify blocks for a specific change
No Extra Mechanisms Guideline	None required	
Design Expecting Failures Guideline	Success and failure messages	"Common sense" handling of AA changes

Tab. 7.2: *Summary of the EASE actor features led to by each guideline and the AA facilities they support.*

Guideline	Agent feature	AA Facility
Explicit Information Guideline	User organisation model	Cost benefit analysis, sensing, common "language"
Design Information Guideline	Potential costs and benefits represented in MDP, uncertainty explicitly represented	Risk analysis
Software Engineering Guideline	Separation between team and individual reasoning	Simplified $\mathcal{R}$, low complexity
Deterministic Execution Guideline	Action policy	Autonomy policy, lookahead
Explicit Behaviour Guideline	Action policy	(as above)
Building Blocks Guideline	Separate handling of each episode	Simplified $\mathcal{R}$
No Extra Mechanisms Guideline	None required	
Design Expecting Failures Guideline	STEAM "protects" others from individual decisions	Simplified $\mathcal{R}$

Tab. 7.3: *Summary of the ELVES actor features led to by each guideline and the AA facilities they support.*

Guideline	System	Violation
Explicit Information Guideline	E-Elves	No high level strategies represented so $\mathcal{R}$ must reason with details
Design Information Guideline	EASE	Limitations on agent behaviour not represented so user cannot know when agent is likely to perform poorly
Software Engineering Guideline	EASE	Does not enforce good practices
Deterministic Execution Guideline	EASE	Negotiation in non-deterministic
Explicit Behaviour Guideline	E-Elves	No high level strategies represented
Building Blocks Guideline	EASE	Allows single agent designs
No Extra Mechanisms Guideline	None	
Design Expecting Failures Guideline	EASE	Negotiation uses previous action

Tab. 7.4: *Summary of the examples of violations of the guidelines and the problems the violations cause.*

7.11 Miscellaneous Agent Design Issues

We conclude this chapter with a brief discussion of some issues related to the design of agents for AA systems that are not covered by the guidelines. In general, these points are things that have come up during the development of one of the systems and may be of interest or value to other AA system developers.

7.11.1 The Guidelines are Flexible

The agent designs for the two implemented AA systems presented in this thesis are distinctly different. EASE agents have a reactive, behaviour based style architecture while E-Elves agents use a distributed, decision theoretic approach. The fact that there is a distinct difference between the architectures is important because it shows that the guidelines are not overly restrictive, i.e., they permit a variety of different designs. The guidelines require certain abstract *features* of an agent design but do not overly constrain *how* those features are achieved.

The range of domains where AA systems will be deployed is very diverse, hence, the types of agent architecture that are going to be required for the different domains are going to be very diverse. Thus, it is critically important that the guidelines do not overly restrict the designer's options. Consider an analogy with a guideline for making comfortable shoes which advises using a hard material on the bottom (to protect the foot) and a soft one on the top (to give flexibility). This guideline is met by a huge variety of different shoe designs that also meet other requirements, e.g., football boots, dress shoes, running shoes, hiking boots, etc. However, in some cases other requirements on the shoes' "performance" may mean that the guideline is violated, e.g., astronaut space walking boots are hard all over. When the guideline is followed comfortable shoes result and when violated uncomfortable shoes result. We believe that the AA guidelines we present have a similar property in that they permit a variety of designs all with the property that AA is subsequently more straightforward to implement.

The fact that we have implemented two very distinct architectures that basically meet the guidelines should *not* be interpreted as showing that any reasonable architecture will meet the guidelines. As the counterexamples in the previous chapter demonstrated it is very easy to violate the guidelines and make AA very difficult to implement.

7.11.2 Choosing between Directed and Reasoned AA

The difference between Directed AA and Reasoned AA is whether a human or software is responsible for $\mathcal{R}$. There appear to be two major properties of an application that will determine which type of AA is appropriate. The first property is whether a human can reasonably be expected to be available to make AA decisions when they need to be made and the second is the purpose of the AA.

Autonomy changes may be appropriate at any point in the operation of the system, i.e., there are not pre-determined times when AA might be useful. If a human is to do $\mathcal{R}$, i.e., Directed AA is used, then it is necessary for the user to be available the whole time the system is running (because the need for that reasoning might occur at any time). For some applications, like the ones EASE is used for and, for example, computer games, this is a reasonable expectation because the system is not intended to run without direct, continuous human involvement (for reasons other than AA). For applications where the system runs around the clock it may not be feasible to always have a human available to do $\mathcal{R}$, unless the criticality of the system is very high. For example, it is unreasonable to require humans to wait around to make autonomy decision in the E-Elves because the system is running around the clock (and requiring constant supervision would defeat the purpose), hence Reasoned AA needs to be employed.

The second important property that needs to be considered when deciding between Directed and Reasoned AA, is the *purpose* of the AA. Reasoned AA can only be used when software can identify the need for autonomy changes. The most obvious case of this is for optimizing system performance given *known* limitations and errors in constituent agents, e.g., as is the case with DAA (Barber, Martin & Mckay 2000) (see Section 3.3.1). If the purpose of the AA is to allow experimentation or entertainment then, clearly, it will be difficult or impossible for software to know when autonomy changes are required hence Directed AA must be used. The *purpose* of the AA is important to deciding whether Directed or Reasoned AA should be used.

7.11.3 Good Software Engineering is Essential – Unfortunately

Several of the guidelines from Chapter 4 advise practices which are well established in the software engineering field. Following software engineering principles leads to software that is straightforward to understand and change. In general, a very well engineered program might meet many of the

guidelines – *regardless of whether or not the designer attempted to meet our guidelines.*

The close correlation between good software engineering practice and good AA practice is perhaps not surprising. In some respects, AA can be seen as pushing an iterative development cycle to its extreme, i.e., the program is not even stopped in between development iterations. Each autonomy change can be seen as a single cycle in an iterative development process, going from one specification to another in response to the deficiencies of the previous program or changing requirements. By definition following good software engineering practices is critical to an effective software development process. So, if AA is viewed as simply a *very* tight iterative development process, we can reasonably expect that the same things that make "normal" software engineering easy to do will be critical for AA.

However, the reliance on good software engineering principles is a cause for some concern. Much software that is produced does not even come close to meeting the exacting principles advocated in software engineering. This makes the software hard to extend, understand, etc. For effective AA it seems necessary to observe software engineering principles *very well* because the whole development process is being taken to an extreme. If normal software engineering standards are difficult to maintain then it might be expected that high standards will be virtually impossible to maintain. But for systems that are not engineered well, AA will be hard to implement – which is disturbing because such systems are most likely to benefit from AA.

7.11.4 Ironically, Behaviour-based Systems are Very Appropriate

Behaviour based ideas were first presented in two seminal papers: *Intelligence without Representation* (Brooks 1991*b*); and *Intelligence without Reason* (Brooks 1991*a*). The idea given by the papers' titles, i.e., no representation and no reason, seem to imply an architecture going directly against most, if not all, the guidelines in Chapter 4. Yet, EASE and a variety of other projects (e.g., Blumberg & Galyean (1995), Perlin & Goldberg (1996)) have successfully implemented AA using behaviour-based agents. The reason for this apparent dichotomy is worth looking at.

Behaviour based agents do not do away with reason and representation, rather the reasoning and representation used is arranged in a radically different way. A critical characteristic of behaviour based agents is that the software is divided "vertically" instead of "horizontally" (Agre & Chapman 1987, Steels 1994). Brooks (1991*a*) explains the difference between conventional AI software and behaviour based software as follows. Traditional AI

software is broken into information processing or functional modules and intelligent behaviour is an emergent property of the interactions between the functional modules. In behaviour based systems, software modules are "behaviour producing" and intelligent functionality is an emergent property of the interactions between the behaviour producing modules. The idea is similar to that of *aspect oriented programming* (Kiczales et al. 1997) which aims to make development of all software easier, by separating out different functionalities and using automated combination techniques.

The decentralization into specialist behaviours actually makes the breakdown of the agent's behaviour *easier* to understand by an observer, provided not too much of the observed behaviour is due to interactions between behaviours. It also makes things easier to change because the breakdown makes it easier to identify and change specific aspects of behaviour. The key reason for this is that behaviour based architectures break down overall behaviour in a way similar to what a human observer would. Hence, when a user looks at the specification of a behaviour based agent they find the pieces they expect to see. For example, when a non-programmer observed the code for, say, a RoboCup player they might expect to see modules for *dribbling* and *shooting*. If the player is coded in a behaviour based style they will find such modules, if coded in a more traditional way they will find modules for planning and world modeling, etc. Hence, the behaviour based system's representation is closer to the expectations of the user, which results in them being easy to use.

So, behaviour based architectures turn out to be very appropriate for AA systems, despite not appearing so on first inspection.

7.11.5 Teamwork is a Key

There are few common features of the agent architectures developed for EASE and E-Elves. One feature, however, is common – the use of teamwork. In EASE, a very simple form of teamwork is used between the agents within an actor to co-ordinate the agent's activities. In E-Elves teamwork is used between Fridays and for group decision making.

Teamwork is useful in an AA system because it creates flexible, abstract connections between the parts of a system (Scerri & Reed 2000*d*). The task of changing the system at runtime is easier because of the flexible, abstract connections. The flexibility aspect means that single components can be changed and the general teamwork structures will look after the details of ensuring that the rest of the system's behaviour moves into line. The abstract relationships between team members enforced by the teamwork model

reduces the occurrence of complex inter-relationships between components. This eases the task of understanding and changing an agent because the parts of the software that need changing are more localized and better insulated from the rest of the system.

It is not necessarily the teamwork per se, that is good for AA it is that the features teamwork has are very useful for AA. The basic features of teamwork meet, at least, the Design Expecting Failures Guideline, the Building Blocks Guideline and the Explicit Behaviour Guideline. Other mechanisms with the same features, thus meeting the same guidelines, we would expect to be just as useful as teamwork.

7.11.6 Using AA During Development

Rapid prototyping (Connell & Shafer 1994), iterative development (Booch 1994), extreme programming (Beck 1999) and other software development methods where systems with reduced functionality are developed then improved have recently become increasingly popular. In such development processes, where working software is produced early on, AA has the possibility to be a very powerful tool for reducing development time.

During the development process for the Headless Chickens IV RoboCup football team (Scerri, Reed, Wiren, Lönneberg & Nilsson 2001), AA was extensively used. The information required for doing $\mathcal{R}$ is similar to that required for debugging, hence the same interfaces used to present information for $\mathcal{R}$ provided useful information for debugging. That is, $\mathcal{I}$ is very useful to understand the agent's behaviour for debugging.

However, the ability to see inside the agent for debugging was a small bonus compared to the ability to change the player's behaviour while it ran. The functionality provided by $\mathcal{A}$ was very useful for experimenting with the behaviour of the agent. The task of setting up specific test scenarios was made significantly simpler because the agent could be manipulated at runtime, instead of the more standard process of stopping the game and changing the player specifications. Furthermore, players with considerable gaps in their functionality, e.g., not stopping when a goal was scored or not listening to referee calls, could still be effectively tested because the developer could come in and "help out" via AA, when the player encountered a situation it did not yet have the functionality to handle.

Based on this experience we believe that AA functionality should be developed in parallel with the rest of the software, as the functionality it can provide can help the overall development process. The usefulness of AA capabilities will be greatest in development processes where functioning but

incomplete systems are produced early in the process.

7.12 Summary

In this Chapter we have evaluated two implementations of very different AA systems to identify features that resulted from adhering to each of the guidelines from Chapter 4. We then identified the facilities that have been easy to develop at the AA level utilizing those features of the agents. We have also examined aspects of the agent designs that have violated the guidelines and shown how this has made implementing AA more difficult. We can conclude that, (in at least some cases) adhering to the guidelines leads to agents that make AA easy to implement and that breaking the guidelines can make AA more difficult to implement.

This chapter also discussed various observations we made while implementing the two AA systems, e.g., we observed the good software engineering was critical. The observations might be helpful to other developers designing agents for AA systems.

8. CONCLUSIONS AND FUTURE WORK

In this chapter we summarise the contents of this thesis, emphasizing its contributions. Finally, we discuss some promising lines of future work that would build on the work presented here.

8.1 Summary

This thesis has examined the problem of how to develop intelligent agents for Adjustable Autonomy (AA) systems. We presented a conceptual model of an AA system which we used to clarify the requirements for each component. Chapter 4 presented guidelines for designing agents which will lead to agent features that make AA straightforward to implement. The guidelines are adhered to in the implementation of agents for the two AA systems which were presented in Chapters 5 and 6. The previous chapter analysed the systems with respect to the guidelines to show the utility of following the guidelines.

8.1.1 Conceptual Model of AA

In Section 2.1.2 we defined the autonomy of an entity, Λ, by the entities decision making responsibilities, authority to pursue goals and intrinsic abilities. We defined an AA system as one where the distribution of autonomy among the intelligent entities in the system could be dynamically changed at run-time. A conceptual model of such an AA system abstractly captures the basic components of most AA systems. The conceptual model has three parts. $\mathcal{I}$ is the component whose responsibility it is to extract relevant information from the intelligent system and transform that information so that it can be used for deciding on autonomy changes. $\mathcal{R}$ takes the information provided by $\mathcal{I}$ and makes decisions about how the autonomy of the system should be configured to achieve the best performance. $\mathcal{R}$ could be performed either by a human or by software. An AA system where $\mathcal{R}$ is performed by software is called a Reasoned AA system and one where $\mathcal{R}$ is performed by a human is called a Directed AA system. The required autonomy changes

are realized by the final component of our conceptual AA system, $\mathcal{A}$. $\mathcal{A}$'s task is to take the decisions about changes in responsibility and authority decided on by $\mathcal{R}$ and affect those changes in the intelligent system.

A core contention of the thesis is that the design of the intelligent agents in an AA system has a significant impact on how straightforwardly $\mathcal{I}$ and $\mathcal{A}$ can be implemented. We argue that if it is difficult to extract information from an agent it is also difficult to implement $\mathcal{I}$. Similarly, we claim that if it is difficult to change authority or responsibility of an agent, implementation of $\mathcal{A}$ is more difficult. It is clear that the limitations on the information provided by $\mathcal{I}$ and the limitations on the ability for $\mathcal{A}$ to implement changes limit the effective reasoning that can be done by $\mathcal{R}$. Hence, we conclude that it is important to design agents for AA systems carefully, so that the AA is not limited or difficult to implement.

8.1.2 Guidelines

Chapter 4 presented the central contribution of this thesis. Eight guidelines provide advice on how agents should be designed so that AA can be straightforwardly implemented for systems including those agents. The guidelines are intended to encourage agent features which make AA as simple to implement as possible. The guidelines should be followed from the earliest stages of agent development so that subsequent implementations of AA proceed more easily.

8.1.3 Implementations

Chapter 5 describes EASE, a Directed AA system for actors in interactive simulation environments. An EASE actor is made up of a hierarchy of simple agents which work together to make the decisions of the actor. The agent organisation represents a large amount of reasoning state information explicitly. That information is easy for $\mathcal{I}$ to extract and present to the user in an informative and usable manner. Other actor services provide mechanisms which allow the actor's behaviour to be changed flexibly at runtime, e.g., changing the goals of the actor by changing the agent organisation. Simple interfaces utilizing these services were shown to give the user a wide range of control over the actor at run-time.

Chapter 6 describes the E-Elves. The E-Elves is a multi-agent system for streamlining daily activities in a human organisation. Behaviour in the E-Elves is separated into individual and team behaviour. Individual agents use an explicit model of the environment which is shared with $\mathcal{R}$. Individual

agents create a complete policy for handling a situation. The policy is available to $\mathcal{R}$ which can use it to make informed autonomy decisions. $\mathcal{R}$ is implemented in software and reasons about the potential costs and benefits of allowing agents and agent teams to make specific decisions autonomously, removing authority from the autonomous system when the potential costs outweigh the potential rewards.

8.1.4 Evaluation

Chapter 7 looked at the impact of following each of the guidelines on the two implemented systems. The evaluation consisted of analysing the effects of each guideline on the two architectures, in particular looking at the agent features that resulted from following the guidelines. By analysing the AA facilities that utilized those features we found that following the guidelines had a positive impact on our ability to implement AA and the ease with which the implementation was done. Furthermore, in some cases the guidelines were violated because of other conflicting design objectives, e.g., efficiency. An examination of the negative impact of violations of the guidelines on the implementations strengthened the case for the utility of the guidelines. We concluded that, at least in the two systems analysed, following the guidelines often led to agent features that were useful for implementing AA, while violating the guidelines often hindered the development of effective AA.

8.2 Summary of Contributions

This work contributes to the AA field in three ways. The central contribution is a set of guidelines which lead agent designers to design agents which allow a straightforward implementation of useful AA (see Chapter 4). Secondly, this work provides an examination of the relationships between the features of an agent and the AA that can be implemented using those features (see mainly Chapter 7). Finally, this thesis contributes two prototype implementations of AA systems, one for actors in simulation environments and one for agents in a human collaboration environment (see Chapters 5 and 6).

8.3 Future Work

In this section we describe some interesting future lines of work. The ideas for future work described below are generally things that we would find interesting to pursue but are beyond the scope of this work.

8.3.1 Tradeoffs

It is not expected that all the guidelines can or will be followed to the letter in the implementation of any particular system. Even the systems presented in this thesis violate the guidelines in some cases. The guidelines will be violated (as they were in the presented implementations) when there are conflicting requirements with higher priority. A good example of a guideline that might be often violated is the Deterministic Execution Guideline. In some systems, e.g., computer games, unpredictability, i.e., non-determinism, is actually a very desirable property of the agents, hence many agents will be non-deterministic and, because this is a violation of the Deterministic Execution Guideline, problematic for AA developers. An interesting future line of work would be to make a more detailed examination of the impact of various violations on the ease with which AA can be implemented in order to help designers make more informed tradeoffs. For example, we might be able to conclude that violating the Software Engineering Guideline was "fatal" for an AA implementation, but violating the Design Expecting Failures Guideline was only a mild inconvenience. Alternatively, or as well, we could look at different methods of overcoming difficult agent features, such as non-determinism, in AA implementations. That is, we could look for effective techniques for dealing with undesirable agent features when implementing AA.

8.3.2 Other Domains

The two systems we have looked at in this thesis are used in quite different domains but still clearly cover only a small sliver of the spectrum of domains where AA systems might be deployed. Building AA for other domains may well unearth a range of other interesting issues. Two specific issues that are clearly going to be relevant to some AA systems are safety criticality and real-time response.

Although mistakes by agents in both simulation and human collaboration domains are annoying and financially costly in terms of wasted time, the affected users are not in danger of being hurt or property of being damaged. Because people cannot be seriously hurt by our systems, we had some flexibility in the implementations because no *guarantees* needed to be given that mistakes would not be made. When building AA for safety critical systems *guarantees* need to be given that mistakes will *never* be made, hence such systems will be constructed in a different way (hopefully). Much of the difference in the development process of safety critical systems will be

in the rigor with which the agent and AA behaviour is developed. As such we believe the guidelines will still be generally applicable because the ease of development is just as important (if not more so) in a safety critical development process as it is in a non-safety critical one. However, issues not considered when building our systems are sure to arise, e.g., what agent features make it straightforward to verify that human-agent interactions are safe (Schreckenhost 1999). Hence, the guidelines might need to be modified to be useful for such applications.

The potential need for guideline changes for safety critical systems may also occur for real-time systems. Although both EASE and E-Elves involved environments where time was a factor, neither presented extremely tight or critical time constraints. In some types of domain, e.g., industrial process control (Musliner & Krebsbach 1999), hundredths of a second can be critical and *guaranteeing* timeliness is important. It is not clear whether such real-time constraints would mean that agents needed to be designed in a different manner. We did notice that the fast reactions of the RoboCup players were harder to control than the slower behaviour of the simulated pilots, hinting that further increases in speed may cause our tools further problems.

8.3.3 Underlying Properties

This work focuses on the *external* features of an agent that makes AA harder or easier to implement. However, we have not tried to abstract further to find *underlying* principles that lead to the types of external properties we have shown to be useful. Tantalizing hints have emerged that there might be more fundamental properties that are important. The two primary candidates for underlying principles that may warrant further investigation are team work and software engineering. Both concepts appear in many of the agent features we have described.

8.3.4 AA Implementations

This thesis focusses on agent design issues for AA systems and does not deeply examine the design of AA mechanisms, i.e., we have looked at only one piece of the overall AA problem. A logical next step would be to apply the same type of analysis we have applied to agent design to AA design. The first stage of such an analysis would be to do a more careful implementation of the prototype AA functionality and perform extensive testing of that functionality. However, as noted above an effective, objective evaluation of the AA functionality is difficult, at best. Once complete AA interfaces have

been implemented and tested we could try to find the important principles underlying their design and capture those principles in a way that allowed others to leverage the knowledge.

8.3.5 EASE and E-Elves Extensions

Both EASE and the E-Elves have served to demonstrate some ideas but would clearly benefit from further development. The obvious area for future work would be to develop the systems' functionality and ability to achieve their tasks effectively. More capable systems are likely to open up more interesting AA challenges.

E-Elves is continually having its functionality extended. Primarily, Friday and the agent team are being applied to new tasks with the aim of further streamlining the human organisation's activities. The autonomy reasoning, $\mathcal{R}$, is also being continually refined and extended. Quantitative experiments which measure user satisfaction with different levels of autonomy might help guide this process. Currently, work is focusing around the idea of trying to provide stronger guarantees that the system will not take costly actions, thereby allowing the user to have more trust in the system.

EASE also shows a variety of promising paths for future development. EASE was originally developed to allow end user programming but no serious evaluation of this functionality has yet been carried out. The simple social structure between agents has been sufficient for the domains to which EASE has so far been applied, but more complex environments which demand more intelligence from actors might benefit from a more sophisticated team structure. Another area where more sophistication might be beneficial is in the negotiation. So far actions that are presented to engineers have been chosen at random. More intelligent algorithms that traverse the action space in a more principled manner may lead to the actor finding better actions faster. Finally, it would be of interest to investigate the development process of EASE actors further, especially the role that AA can play in that process.

8.4 Concluding Remarks

This thesis looks at the relatively new area of Adjustable Autonomy. Adjustable Autonomy is a technology that can ensure that humans retain responsibility for important actions in their environment while leveraging the power of autonomous systems. As we begin to rely more and more on intelligent software such technology is likely to become more and more critical.

With due diligence and some hard work we believe that we can leverage some of the fantastic potential intelligent agents offer without losing control of our environment.

BIBLIOGRAPHY

Agre, P. & Chapman, D. (1987), Pengi: An implementation of a theory of activity, *in* 'Proceedings of AAAI-87', pp. 268–272.

Alloyer, O., Bonakdarian, E., Cremer, J., Kearney, J. & Willemsen, P. (1997), Embedding scenarios in ambient traffic, *in* 'Proceedings of DSC'97 (Driving simulation conference)', Lyon, France, pp. 75–84.

Andersson, J. (1995), Plan oriented and rule-based system for generation of robust tactics in long range air combat with multiple targets, Master's thesis, Linköping University.

André, E., Rist, T. & Müller, J. (1998), Integrating reactive and scripted behaviors in a life-like presentation agent, *in* 'Proceedings of the Second International conference on Autonomous Agents', pp. 261–268.

Asimov, I. (1950), *I, Robot*, Bantam Books.

Badler, N. (1997), Real-time virtual humans, *in* 'Pacific Conference on Computer Graphics and Applications', pp. 4–13.

Banks, S. & Stytz, M. (1999), Considerations for the next generation of air force computer generated actors, *in* 'Proceedings of SimTecT99 - Advancing Simulation Technology and Training'. http://www.cse.rmit.edu.au/simtect/1999/papers/003.doc.

Barber, K., Goel, A. & Martin, C. (2000), 'Dynamic adaptive autonomy in multi-agent systems', *Journal of Experimental and Theoretical Artificial Intelligence* **12**(2), 129–148.

Barber, K. S. & Martin, C. E. (1999*a*), Agent autonomy: Specification, measurement, and dynamic adjustment, *in* 'Autonomy Control Software Workshop, Autonomous Agents 99', pp. 8–15.

Barber, K. S. & Martin, C. E. (1999*b*), Applying dynamic planning frameworks to agent goals, *in* D. Musliner & B. Pell, eds, 'Proceedings of AAAI

Spring Symposium Agents with Adjustable Autonomy', Stanford, California, pp. 1–8.

Barber, K. S., Martin, C. & Mckay, R. (2000), A communication protocol supporting dynamic autonomy agreements, *in* 'Proceedings of PRICAI 2000 Workshop on Teams with Adjustable Autonomy', Melbourne, Australia, pp. 1–10.

Bates, J. (1993), The nature of character in interactive worlds and the OZ project, *in* C. Loeffler, ed., 'Virtual Realities: Anthology of industry and Culture', Gijitsu Hyoron Sha.

Beck, K. (1999), *Extreme Programming Explained: Embrace Change*, Addison-Wesley.

Beizer, B. (1990), *Software testing and techniques*, Van Nostrand Reinhold.

Bernard, D., Dorais, G., Gamble, E., Kanefsky, B., Kurien, J., Man, G., Millar, W., Muscettola, N., Nayak, P., Rajan, K., Rouquette, N., Smith, B., Taylor, W. & Tung, Y. (1999), Spacecraft autonomy flight experience: The DS1 remote agent experiment, *in* 'Proceedings of the AIAA Space Technology Conference', Albuquerque, New Mexico, pp. 259–281.

Bindiganavale, R., Schuler, W., Allbeck, J., Badler, N., Joshi, A. & Palmer, M. (2000), Dynamically altering agent behaviors using natural language instructions, *in* 'Proceedings of the Fourth International Conference on Autonomous Agents', Barcelona, Spain, pp. 293–300.

Blumberg, B. (1997*a*), Go with the flow: Synthetic vision for autonomous animated creatures, *in* 'Proceedings of the First International Conference on Autonomous Agents (Agents'97)', Marina Del Ray, pp. 538–539.

Blumberg, B. (1997*b*), Old tricks, new dogs: Ethology and Interactive Creatures, PhD thesis, Massachusetts Institute of Technology.

Blumberg, B. & Galyean, T. (1995), Multi-level control of autonomous animated creatures for real-time virtual environments, *in* 'Siggraph '95 Proceedings', ACM Press, New York, pp. 295–304.

Bonasso, P. (1999), Issues in providing adjustable autonomy in the 3T architceture, *in* 'Proceedings of AAAI Spring Symposium on Adjustable Autonomy', pp. 11–16.

Bonasso, R., Firby, R., Gat, E., Kortenkamp, D., Miller, D. & Slack, M. (1997), 'Experiences with an architecture for intelligent reactive agents', *Journal of Experimental and Theorectical Artificial Intelligence* **9**(1), 237–256.

Booch, G. (1994), *Object-Oriented Analysis and Design*, Addison-Wesley.

Boutilier, C., Goldszmidt, M. & Sabata, B. (1999), Sequential auctions for the allocation of resources with complementarities, *in* 'Proceedings of the sixteenth international joint conference on AI', pp. 527–534.

Bradshaw, J. (1997), An introduction to software agents, *in* 'Software Agents', MIT Press, pp. 3–49.

Brainov, S. & Hexmoor, H. (2001), Quantifying relative autonomy in multiagent interaction, *in* 'IJCAI Workshop on Autonomy Delegation and Control: Interacting with Intelligent Agents', pp. 26–35.

Brann, D., Thurman, D. & Mitchell, C. (1996), Human interaction with lights-out automation: A field study, *in* 'Proceedings of the 1996 symposium on human interaction and complex systems', Dayton, USA, pp. 276–283.

Brooks, R. (1991*a*), Intelligence without reason, *in* 'Proceedings 12th International Joint Conference on AI', Sydney, Australia, pp. 569–595.

Brooks, R. (1991*b*), 'Intelligence without representation', *Artificial intelligence journal* **47**, 139–159.

Bryson, J. (1999*a*), Creativity by design: A character based approach to creating creative play, *in* 'AISB Symposium on AI and Creativity in Entertainment', pp. 9–16.

Bryson, J. (1999*b*), Hierarchy and sequence vs. full parallelism in action selection, *in* 'Intelligent Virtual Agents 2', pp. 113–125.

Bryson, J. & McGongile, B. (1998), Agent architectures as object oriented design, *in* M. Singh, ed., 'The fourth international workshop on agent theories, architectures and languages (ATAL97)', Springer Verlag, pp. 15–30.

Burgard, W., Cremers, A., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W. & Thrun, S. (1998), The interactive museum tour-guide robot, *in* 'Proceedings of AAAI'98', pp. 11–18.

Burt, A. (1998), Modelling motivational behavior in intelligent agents in virtual worlds, *in* 'Proceedings of the 1998 Conference on virtual worlds and simulation', Vol. Simulation Series, Volume 2.

Bye, A., Hollnagel, E. & Brendeford, T. S. (1999), 'Human-machine function allocation: a functional modelling approach', *Reliability engineering and system safety* **64**, 291–300.

Carmel, D. & Markovitch, S. (1998), How to explore your opponents strategy (almost) optimally, *in* 'Proceedings of the International conference on multi-agent systems', Paris, pp. 64–71.

Castelfranchi, C. & Falcone, R. (1998), Principles of trust for mas: Cognitive autonomy, social importance and quantification, *in* 'Proceedings of the International conference on multi-agent systems', Paris, pp. 64–71.

Cesta, A., D'Aloisi, D. & Collia, M. (1999), Adjusting autonomy of agent systems, *in* 'Proceedings of the AAAI Spring Symposium on Agents with Adjustable Autonomy', pp. 17–24.

Chaib-draa, B. (1997), *Readings in Agents*, Morgan Kaufmann, chapter Industrial Applications of distributed AI, pp. 31–35.

Chalupsky, H., Gil, Y., Knoblock, C., Lerman, K., Oh, J., Pynadath, D., Russ, T. & Tambe, M. (2001), Electric elves: Applying agent technology to support human organizations, *in* 'International Conference on Innovative Applications of AI', pp. 51–58.

Chang, J. & Chen, Y. (1998), 'Force control of a single-link flexible arm using sliding-mode theory', *Journal of Vibration and Control* **4**, 187–200.

Chapanis, A. (1965), 'On the allocation of functions between man and machine', *Occupational Psychology* **39(1)**.

Cohen, P., Greenberg, M., Hart, D. & Howe, A. (1989), 'Trial by fire: Understanding the design requirements for agents in complex environments', *AI Magazine* **10**(3), 32–48.

Collis, J., Soltysiak, S., Ndumu, D. & Azarmi, N. (2000), 'Living with agents', *BT Technology Journal* **18**(1), 66–67.

Connell, J. & Shafer, L. (1994), *Object-oriented rapid prototyping*, Yourdon Press.

Coradeschi, S. (1997), 'A decision-mechanism for reactive and coordinated agents', Licentiate Thesis. LiU-Tek-Lic 1997:16.

Craft, M. & Karr, C. (1996), Testing future weapons systems using cgf systems, *in* 'Proceedings of the sixth conference on computer generated forces and behavioral representation', Orlando, Florida, pp. 141–150.

Cremer, J., Kearney, J. & Papelis, Y. (1995*a*), 'HCSM: A framework for behavior and scenario control in virtual environments', *ACM Transactions on Modeling and Computer Simulation* pp. 242–267.

Cremer, J., Kearney, J. & Papelis, Y. (1995*b*), 'HCSM: A framework for behaviour and scenarion control in virtual environments', *ACM Transactions of modeling and computer simulation* **5**(3), 242–267.

Davidson, C. (1998), 'Agents from Albia', *New Scientist* **158**(2133).

de Carvalho Gomes, F., Lima, A., Olivera, C. & de Meneses, C. (1998), Asynchronous organizations for solving the point to point problem, *in* 'Third International Conference on Multi-agent systems', Paris, pp. 144–149.

Doherty, P., Granlund, G., Kuchcinski, K., Sandewall, E., Nordberg, K., Skarman, E. & Wiklund, J. (2000), The WITAS unmanned aerial vehicle project, *in* 'Proceedings of the 14th European Conference on Artificial Intelligence', Berlin, pp. 747–755.

Donaldson, T. & Cohen, R. (1997), A constraint satisfaction framework for managing mixed-initiative discourse, *in* 'Proceedings of AAAI Spring symposium on Computational models for mixed initiative interaction'.

Dorais, G., Bonasso, R., Kortenkamp, D., Pell, B. & Schreckenghost, D. (1998), Adjustable autonomy for human-centered autonomous systems on mars, *in* 'Proceedings of the first international conference of the Mars society', pp. 397–420.

Dorais, G. & Kortenkamp, D. (2001), *PRICAI Workshop Reader*, Vol. 2112, Springer Verlag, chapter Designing Human-Centered Autonomous Agents.

Dörner, R., Grimm, P. & Seiler, C. (2000), Agents and virtual environments for communication and decision training emergencies, *in* 'Proceedings of the fourth international conference on Autonomous Agents, Agents 2000', pp. 50–51.

Doyle, P. & Hayes-Roth, B. (1998), Agents in annotated worlds, *in* 'Proceedings of the Second international conference on Autonomous Agents', pp. 173–180.

EASports (2000), 'NHL2000', http://nhl2000.ea.com/.

Eaton, P., Freuder, E. & Wallace, R. (1998), 'Constraints and agents: Confronting ignorance', *AI Magazine* **19**(2), 51–67.

Eder, J., Kappel, G. & Schrefl, M. (1992), Coupling and cohesion in object-oriented systems, *in* 'Proceedings of Conference on Information and Knowledge Management', Baltimore.

Etzioni, O. & Weld, D. (1997), *Readings in Agents*, Morgan Kaufmann, chapter A Softbot-based interface to the internet.

Falcone, R. & Castelfranchi, C. (1999), Levels of delegation and levels of help for agents with adjustable autonomy, *in* 'Proceedings of AAAI Spring symposium on agents with adjustable autonomy', pp. 25–32.

Ferguson, G. & Allen, J. (1998), TRIPS : An intelligent integrated problem-solving assistant, *in* 'Proceedings of Fifteenth National Conference on Artificial Intelligence(AAAI-98)', Madison, WI, USA, pp. 567–573.

Ferguson, G., Allen, J. & Miller, B. (1996), TRAINS-95 : towards a mixed-initiative planning assistant, *in* 'Proceedings of the third conference on artificial intelligence planning systems', pp. 70–77.

Finin, T., Labrou, Y. & Mayfield, J. (1997), *Software agents*, The MIT press, chapter KQML as an agent communication language, pp. 291–316.

Fischer, K., Müller, J. & Pischel, M. (1994), Unifying control in a layered agent archiecture, Technical report, German Research Center for Artificial Intelligence, http://www.dfki.uni-sb.de/mas/interrap/TM9405.ps.

Fitts, P. (1962), 'Functions of men on complex systems', *Aerospace Engineering* **21(1)**.

Flanagan, J. & Huang, T., eds (1997), *NSF Workshop on human-centered systems: information, interactivity and intelligence.*

Fleming, M. & Cohen, R. (1999), Towards a methodology for designing and evaluating mixed-initiative AI systems, *in* 'Proceedings of AAAI Workshop on mixed initiative intelligence', pp. 130–134.

Fong, T., Thorpe, C. & Baur, C. (1999), Collaborative control: A robot-centric model for vehicle teleoperation, *in* 'Proceedings of AAAI Spring Symposium on Agents with Adjustable Autonomy', pp. 33–40.

Fox, J. & Das, S. (2000), *Safe and Sound: Artificial Intelligence in Hazardous Applications*, The AAAI Press.

Galitsky, B. (1999), Agents with adjustable autonomy for scheduling in the competitive environment, *in* 'Proceedings of AAAI Spring Symposium on agents with adjustable autonomy', pp. 41–49.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Georgeff, M. & Lansky, A. (1987), Reactive reasoning and planning, *in* 'Proceedings of the sixth national conference on artificial intelligence (AAAI-87)', Seattle, WA, pp. 677–682.

Gerber, C., Siekmann, J. & Vierke, G. (1999), Holonic multi-agent systems, Research Report RR-99-03, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

Ghedira, K. (1994), A distributed approach to partial constraint satisfaction problems, *in* 'Distributed Software Agents and their Applications. 6th European Workshop on Modelling Autonomous Agents in a Multiagent World, MAAMAW'94', Springer Verlag Lecture Notes in Artificial Intelligence, pp. 106–122.

Gilbreath, G., Ciccimaro, D. & Everett, H. (2000), An advanced telereflexive tactical response robot, *in* T. Fong & C. Thorpe, eds, 'Proceedings of Vehicle Teleoperation Interfaces Workshop, IEEE International Conference on Robotics and Automation', San Fransisco, CA.

Goel, A., de Silva Garza, A. G., Grue, N., Murdock, J. & Recker, M. (1996), Explanatory interface in interactive design environments, *in* 'Fourth international conference on artificial intelligence in design'.

Goldberg, D. & Mataric, M. (2000), Robust behavior-based control for distributed multi-robot collection tasks, Technical Report IRIS-00-387, Institute for Robotics and Intelligent Systems, University of Southern California.

Goldman, R., Guerlain, S., Miller, C. & Musliner, D. (1997), Integrated task representation for indirect interaction, *in* 'Working Notes of the AAAI Spring Symposium on computational models for mixed initiative interaction'.

Grand, S. (2000), 'Creatures: an exercise in creation', http://www.creatures.co.uk/Library/Science/sci_1exercise.htm.

Grand, S. & Cliff, D. (1998), 'Creatures: Entertainment software agents with artificial life', *Autonomous agents and multiagent systems* pp. 39–57.

Granlund, R. (1997), C3Fire: A microworld supporting emergency management training, Master's thesis, Department of Computer and information science, Linköping university.

Greenwald, A. & Kephart, J. (1999), Shopbots and pricebots, *in* 'Proceedings of the Sixteenth international joint conference on artificial intelligence', Vol. 1, pp. 506–511.

Grote, G., Weik, S., Wäfler, T. & Zölch, M. (1995), *Symbiosis of human and artifact*, Elsevier, chapter Complementary allocation of functions in automated work systems.

Gunderson, J. & Martin, W. (1999), Effects of uncertainty on variable autonomy in maintainance robots, *in* 'Agents'99 workshop on autonomy control software', pp. 26–34.

Haller, S., ed. (1997), *AAAI Spring Symposium on Computational models for mixed initiative interaction.*

Hayes-Roth, B. (1995), 'An architecture for adaptive intelligent systems', *Artificial Intelligence* **72**(1), 329–365.

Hayes-Roth, B., Brownston, L. & van Gent, R. (1997), *Readings in Agents*, Morgan Kaufmann, chapter Multiagent collaboration in directed improvization, pp. 141–147.

Heritage (1996), *The American Heritage Dictionary of the English Language*, Houghton Mifflin Company.

Hexmoor, H. (1999*a*), Adjusting autonomy by introspection, *in* 'Proceedings of AAAI Spring Symposium on Agents with Adjustable Autonomy', pp. 61–64.

Hexmoor, H. (2000*a*), Case studies of autonomy, *in* 'Proceedings of FLAIRS 2000', pp. 246–249.

Hexmoor, H. (2000*b*), A cognitive model of situated autonomy, *in* 'Proceedings of PRICAI-2000, Workshop on Teams with Adjustable Autonomy', Melbourne, Australia, pp. 11–20.

Hexmoor, H., ed. (1999*b*), *Workshop on Autonomy Control Software*, Autonomous Agents 1999.

Hexmoor, H. & Kortenkamp, D. (2000), 'Introduction to autonomy control software', *Journal of Experiemental and Theoretical Artificial Intelligence* **12**(2), 123–128.

Horvitz, E. (1999*a*), Principles of mixed-initiative user interfaces, *in* 'Proceedings of CHI'99, ACM SIGCHI Conference on Human Factors in Computing Systems', Pittsburgh, PA, pp. 159–166.

Horvitz, E. (1999*b*), 'Uncertainty, action and interaction: In pursuit of mixed-initiative computing', *Intelligent Systems* pp. 17–20.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D. & Rommelse, K. (1998), The Lumiere project: bayesian user modeling for inferring the goals and needs of software users, *in* 'Proceedings of the fourteenth conference on uncertainty in artificial intelligence', pp. 256–265.

Horvitz, E., Jacobs, A. & Hovel, D. (1999), Attention-sensitive alerting, *in* 'Proceedings of UAI'99, Conference on Uncertainty and Artificial Intelligence', Stockholm, Sweden, pp. 305–313.

Huhns, M. & Singh, M. (1997), *Readings in Agents*, Morgan Kaufmann, chapter Agents and Multiagent systems: themes approaches and challenges, pp. 1–24.

IEEE (1998), 'IEEE standard for application and management of the systems engineering process', IEEE Standard 1220-1998.

Ingrand, F., Georgeff, M. & Rao, A. (1992), 'An architecture for real-time reasoning and system control', *IEEE Expert* **7**(6), 34–44.

Jennings, N. (1999), Agent-based computing: Promise and perils, *in* 'Proceedings of the 16th International Conference on Artificial Intelligence', pp. 1429–1436.

Jennings, N. & Wooldridge, M. (1998), *Agent Technology Foundations, Applications and Markets*, Springer-Verlag, chapter Applications of Intelligent Agents.

Joseph, R. (1986), Visual analysis: An empirical evaluation of design guidelines for downhill ski trails and mountain support facilities, Masters thesis, Kansas State University.

Joy, B. (2000), 'Why the future doesn't need us', *Wired* **8.04**.

Kelso, S. (1995), *Dynamic Patterns: the self-organization of brain and behavior*, The MIT Press.

Kendrick, D. (1981), *Stochastic control for economic models*, McGraw-Hill.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. & Irwin, J. (1997), Aspect oriented programming, *in* 'Proceedings of the european conference on Object-Oriented Programming', LNCS 1241, Finland, pp. 220–242.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., & Matsubara, H. (1997), 'RoboCup: A challenge problem for AI', *AI Magazine* **18**(1), 73–85.

Kortenkamp, D., Burridge, R., Bonasso, P., Schrenkenghoist, D. & Hudson, M. B. (1999), An intelligent software architecture for semi-autonomous robot control, *in* 'Autonomy Control Software Workshop, Autonomous Agents 99', pp. 36–43.

Kortenkamp, D., Keirn-Schreckenghost, D. & Bonasso, R. P. (2000), Adjustable control autonomy for manned space flight, *in* 'IEEE Aerospace Conference'.

Lanier, J. (2001), 'One half a manifesto', http://www.edge.org/-3rd_culture/lanier/lanier_index.html.

Lashkari, Y., Metral, M. & Maes, P. (1998), *Readings in Agents*, Morgan Kaufman, chapter Coolaborative Interface agents, pp. 111–116.

Lee, S. (1995), 'Intelligent sensing and control for advanced teleoperation', *IEEE Control Systems Magazine* **13**(3), 19–28.

Lesser, V., Atighetchi, M., Benyo, B., Horling, B., Raja, A., Vincent, R., Wagner, T., Xuan, P. & Zhang, S. (1999), The UMASS intelligent home

project, *in* 'Proceedings of the Third Annual Conference on Autonomous Agents', Seattle, USA, pp. 291–298.

Luck, M. & d'Inverno, M. (1995), A formal framework for agent and autonomy, *in* 'Proceedings of ICMAS'95', pp. 254–260.

Luna, F. & Stefannson, B. (2000), *Economic Simulations in Swarm: agent-based modelling and object oriented programming*, Kluwer Academic Publishers.

MacKenzie, D. (1996), A design methodology for the configuration of behavior-based mobile robots, PhD thesis, Georgia Institute of Technology.

Maes, P. (1994*a*), 'Agents that reduce work and information overload', *Communications of the ACM* **37**(7), 31–40.

Maes, P. (1994*b*), 'Modeling adaptive autonomous agents', *Artificial Life Journal* **1**(1 & 2), 135–162.

Maes, P. (1995), 'Artificial life meets entertainment: Interacting with lifelike autonomous agents', *Special Issue on New Horizons of Commercial and Industrial AI, Communications of the ACM* **38**(11), 108–114.

Malin, J. & Fleming, L. (1999), Validation of adjustable autonomous control systems for production plants, *in* 'AAAI Spring Symposium on Agents with Adjustable Autonomy', Stanford, California, pp. 73–78.

Mataric, M. (1992), Behavior-based systems: Main properties and implications, *in* 'IEEE International Conference on Robotics and Automation, Workshop on Architectures for', Nice, France, pp. 46–54.

Mataric, M. (1994), Interaction and Intelligent Behavior, PhD thesis, Massachusetts Institute of Technology.

Milewski, A. & Lewis, S. (1997), 'Delegating to software agents', *International journal of human-computer studies* **46**, 485–500.

Miller, D. (1998), *Assistive Technology and AI*, Vol. LNAI 1458, Spring-Verlag, chapter Assistive Robotics: an overview, pp. 126–136.

Miller, D. (1999), Semi-autonomous mobility verses semi-mobile autonomy, *in* 'Proceedings of the 1999 AAAI Spring Symposium on Adjustable Autonomy', pp. 77–78.

Minsky, M. (1988), *The Society of Mind*, Simon and Schuster.

Musliner, D. & Krebsbach, K. (1999), Adjustable autonomy in procedural control for refineries, *in* 'AAAI Spring Symposium on Agents with Adjustable Autonomy', Stanford, California, pp. 81–87.

Nakashima, H. & Noda, I. (1998), Dynamic subsumption architecture for programming intelligent agents, *in* 'Third International Conference on Multi-agent systems', Paris, pp. 190–197.

Ndumu, D., Nwana, H., Lee, L. & Haynes, H. (1998), 'Visualization and debugging of distributed multiagent systems', *Applied Artificial Intelligence* **13**(1-2), 187–208.

Neves, M. & Oliveira, E. (1997), A control architecture for an autonomous mobile robot, *in* 'proceedings of the first international conference on autonomous agents', pp. 193–200.

Noda, I. (1995), Soccer server: A simulator of RoboCup, *in* 'Proceedings of AI Symposium'95', Japanese Society for Artificial Intelligence.

Ogasawara, G. (1993), RALPH-MEA: A real-time, decision theoretic agent architecture, PhD thesis, University of California, Berkeley.

Ossowski, S. & García-Serrano, A. (1999), *Intelligent Agents V : Agent Theories Architectures and Languages*, Springer, chapter Social Structure in Artificial Agent Societies: Implications for Autonomous Problem Solving Agents, pp. 133–148.

Parker, L. E. (1998), 'Alliance: An architecture for fault tolerant multi-robot cooperation', *IEEE Transactions on Robotics and Automation* **14**(2), 220–240.

Passino, K. & Yurkovich, S. (1998), *Fuzzy Control*, Addison Wesley.

Pell, B., Gamble, E., Gat, E., Keesing, R., Kurien, J., Millar, W., Nayak, P., Plaunt, C. & Williams, B. (1998), A hybird procedural/deductive executive for autonomous spacecraft, *in* 'Proceedings of the second international conference on autonomous agents', pp. 369–376.

Perlin, K. & Goldberg, A. (1996), 'Improv: A system for scripting interactive actors in virtual worlds', *Computer Graphics* **30**, 205–216.

Perzanowski, D., Schultz, A., Marsh, E. & Adams, W. (1999), Goal tracking and goal attainment: a natural language means of achieving adjustable autonomy, *in* 'Proceedings of AAAI Spring symposium on adjustable autonomy', pp. 93–100.

Pew, R. & Mavor, A., eds (1998), *Modeling Human and Organizational Behavior*, National Academy Press, Washington, D.C. National Research Council.

Pirjanian, P. (1998), Multiple objective action selection and behavior fusion voting, PhD thesis, Department of Medical Informatics and Image Analysis, Aalborg university.

Pollack, M. E. (1996), 'Planning in dynamic environments: The dipart system', *Advanced Planning Technology* pp. 218–225.

Puterman, M. (1994), *Markov Decision processes*, Wiley, New York.

Pynadath, D., Scerri, P. & Tambe, M. (2001), MDPs for adjustable autonomy in a real-world multi-agent environment, *in* 'AAAI Spring Symposium on decision theoretic and game theoretic agents', pp. 107–116.

Quinlan, J. (1993), *C4.5: Programs for machine learning*, Morgan Kaufmann.

Rajan, K., Shirley, M., Taylor, W. & Kanefsky, B. (2000), Ground tools for the 21st century, *in* 'Proceedings of the IEEE aerospace conference'.

Reed, N., ed. (2000), *Proceedings of PRICAI Workshop on Teams with Adjustable Autonomy*, Melbourne, Australia.

Repenning, A. (n.d.), *AgentSheets and Visual AgentTalk: Getting Started*, 1.3.0a2 edn, Department of Computer Science and Center for LifeLong Learning.

Reynolds, C. (1995), 'Authoring autonomous characters', Invited Talk, Distinguished Lecture Series, Georgia Institute of Technology.

Rich, C. & Sidner, C. (1998), COLLAGEN: when agents collaborate with people, *in* 'Readings in agents', Morgan Kaufmann, pp. 117–124.

Riekki, J. (1998), Reactive task execution of a mobile robot, PhD thesis, Infotech Oulu and Department of Electrical Engineering, University of Oulu, Oulu, Finland.

Rock, D. (1999), agent autonomy adjustment for midair collision avoidance, *in* 'Proceedings of autonomy control workshop at Autonomous agents'99', pp. 91–97.

Rosenblatt, J. & Thorpe, C. (1995), Combining multiple goals in a behavior based architecture, *in* 'Proceedings of 1995 International Conference on Intelligent Robots and Systems (IROS)', Pittsburg, PA, pp. 136–141.

Rosenbloom, P., Laird, J., Newell, A. & McCarl, R. (1991), 'A preliminary analysis of the Soar architecture as a basis for general intelligence', *Artificial Intelligence* **47**, 289–325.

Russell, S. & Norvig, P. (1995), *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Inc.

Rybski, P., Stoeter, S., Erickson, M., Gini, M., Hougen, D. & Papanikolopoulos, N. (2000), A team of robotic agents for surveillance, *in* 'Proceedings of the fourth international conference on autonomous agents', pp. 9–16.

Saab (1998), *TACSI - User Guide*, 5.2 edn, Gripen, Operational Analysis, Modeling and Simulation. in Swedish.

Saffiotti, A. (1997), 'The uses of fuzzy logic in autonomous robot navigation: a catalogue raisonne', *Soft Computing* **1**(4), 180–197.

Sawaragi, T. & Horiguchi, Y. (2000), 'Ecological interface enaling human-embodied cognition in mobile robot teleoperation', *Intelligence: New Visions of AI in practice* **11**(3), 17–19.

Scerri, P., Pynadath, D. & Tambe, M. (2000), Don't cancel my barcelona trip: adjusting the autonomy of agent proxies in human organizations, *in* 'Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents — the human in the loop', pp. 169–173.

Scerri, P., Pynadath, D. & Tambe, M. (2001), Adjustable autonomy in real-world multi-agent environments, *in* 'Proceedings of the Fifth international conference on autonomous agents (Agents'01)', pp. 300–307.

Scerri, P. & Reed, N. (2000*a*), Creating complex actors with EASE, *in* 'Proceedings of the Fourth International Conference on Autonomous Agents', pp. 142–143.

Scerri, P. & Reed, N. (2000*b*), The EASE actor development environment, *in* 'Proceedings of the Workshop of the Swedish AI Society, SAIS'2000'.

Scerri, P. & Reed, N. (2000*c*), 'Engineering characteristics of autonomous agent architectures', *Journal of Experimental and Theoretical artificial intelligence* **12**(2), 191–212.

Scerri, P. & Reed, N. (2000*d*), Making adjustable autonomy easier with teamwork, *in* 'Proceedings of PRICAI'2000 workshop on teams with adjustable autonomy', pp. 25–34.

Scerri, P. & Reed, N. (2001), Designing agents for systems with adjustable autonomy, *in* 'Proceedings of the IJCAI-01 Workshop on Autonomy, Delegation and Control: Interacting with Autonomous Agents'. to appear.

Scerri, P. & Reed, N. E. (1999), Adapting an agent to a similar environment, *in* 'Third International Conference on Autonomous Agents (Agents 99)', Association for Computing Machinery, pp. 420–421.

Scerri, P., Reed, N., Wiren, T., Lönneberg, M. & Nilsson, P. (2001), *RoboCup-2000: Robot Soccer World Cup IV*, Springer Verlag, chapter Headless Chickens IV, pp. 493–496.

Schneider, J. (1995), Exploiting model uncertainty estimates for safe dynamic control learning, *in* 'Neural Information Processing Systems', Vol. 9.

Schooley, L., Zeigler, B., Cellier, F. & Wang, F. (1993), 'High-autonomy control of space resource processing plants', *IEEE Control Systems Magazine* **13**(3), 29–39.

Schreckenhost, D. (1999), Human interaction with control software supporting adjustable autonomy, *in* D. Musliner & B. Pell, eds, 'Agents with adjustable autonomy', AAAI 1999 spring symposium series, pp. 116–119.

Sen, S., Haynes, T. & Arora, N. (1997), 'Satisfying user preferences while negotiating meetings', *International journal of humna-computer studies* **47**, 407–427.

Sengers, P. (1998), Do the right thing: An architecture for action-expression, *in* 'Proceedings of the second international conference on Autonomous Agents (Agents98)', pp. 24–31.

Sheridan, T. (1992), *Telerobotics, automation and Human Supervisory Control*, MIT Press, Cambridge, Massachusetts.

Shneiderman, B. (1998), *Designing the User Interface*, Addison Wesley.

Shoham, Y. (1998), *Readings in Agents*, Morgan Kaufman, chapter Agentoreinted programming, pp. 329–349.

Simmons, R., Krotkov, E., Hebert, M. & Katragadda, L. (1994), Experience with rover navigation for lunar-like terrains, *in* 'International Lunar Exploration Conference', San Deigo, CA.

Simpson, R., Levine, S., Bell, D., Jaros, L., Koren, Y. & Borenstein, J. (1998), *Assistive Technology and AI*, Vol. LNAI 1458, Spring-Verlag, chapter NavChair:An Assistive Wheelchair Navigation system with automatic adaption, pp. 235–255.

Smith, B. (2000), 'The Sims', The Computer Gamers' Gazette. http://-ourworld-top.cs.com/comgamesig/simssplash.htm.

Sommerville, I. (1996), *Software Engineering*, 5 edn, Addison Wesley.

Song, H., Franklin, S. & Negatu, A. (1996), Sumpy: A fuzzy software agent, *in* 'Proceedings of the ISCA Conference on Intelligent Systems', Reno, Nevada, pp. 124–129.

Sony (2001), 'www.aibo.com'.

Steels, L. (1994), *The artificial life route to artificial intelligence: Building situated embodied agents*, Lawrence Erlbaum Associates, New Haven, chapter Building Agents with Autonomous Behavior Systems.

Stone, P., Riley, P. & Veloso, M. (2000), Layered disclosure: Why is the agent doing what it's doing?, *in* 'Proceedings of Fourth International Conference on Autonomous Agents', Barcelona.

T. Fong, C. T. & Baur, C. (2000), Advanced interfaces for vehicle teleoperation: collaborative control, sensor fusion displays, and web-based tools, *in* 'Vehicle Teleoperation Interfaces Workshop, IEEE International Conference on Robotics and Automation', San Fransisco, CA.

Tadokoro, S., Kitano, H., Takahashi, T. & et al (2000), The RoboCup rescue project: A robotic approach to the disaster mitigation problem, *in* 'Proceedings of IEEE International conference on robotics and automation', pp. 4089–4094.

Takahashi, T. (2000), *RoboCup-99: Robot Soccer World Cup III*, Springer, chapter Kasugabito III, pp. 592–595.

Tambe, M. (1997), 'Towards flexible teamwork', *Journal of Artificial Intelligence Research* **7**, 83–124.

Tambe, M., Johnson, W. L., Jones, R., Koss, F., Laird, J., Rosenbloom, P. & Schwamb, K. (1995), 'Intelligent agents for interactive simulation environments', *AI Magazine* **16**(1), 15–39.

Tambe, M., Pynadath, D., Chauvat, C., Das, A. & Kaminka, G. (2000), Adaptive agent architectures for heterogeneous team members, *in* 'Proceedings of ICMAS'2000', pp. 301–308.

Tambe, M., Pynadath, D. & Chauvat, N. (2000), 'Building dynamic agent organizations in cyberspace', *IEEE Internet Computing* **4**(2), 65–73.

Tate, A. (1997), Mixed-initiative interaction in O-Plan, *in* 'Proceedings of AAAI Spring symposium on Computational models for mixed initiative interaction'.

Thorstensson, M. (1997), Situation analysis in air combat simulation using fuzzy logic, Master's thesis, Linköping University.

Travers, M. (1996), Programming With Agents: New mataphors for thinking about computation, PhD thesis, Massachusetts Institue of Technology.

Tunstel, E. (1996), mobile robot autonomy via hierarchical fuzzy behavior control, *in* 'Proceedings of 6th international symposium on robotics and manufacturing', Montipellier, France, pp. 837–842.

Tyrell, T. (1993), Computational Mechanisms for Action Selection, PhD thesis, University of Edinburgh.

Veloso, M., Carbonell, J., Pérez, M. A., Borrajo, D., Fink, E. & Blythe, J. (1995), 'Integrating planning and learning: The Prodigy architecture', *Journal of Experimental and Theoretical Artificial Intelligence* **7**(1), 81–120.

Veloso, M., Mulvehill, A. & Cox, M. (1997), Rationale-supported mixed-initiative case-based planning, *in* 'Proceedings of the fourteenth national conference on artificial intelligence and ninth innovative applications of artificial intelligence conference', pp. 1072–1077.

Wavish, P. & Connah, D. (1997), Virtual actors that can perform scripts and improvise roles, *in* 'First International conference on intelligent agents', pp. 317–322.

Webber, B. & Badler, N. (1993), Virtual interactive collaborators for simulation and training, *in* 'Third conference on computer generated forces and behavioral representation', Florida, pp. 199–208.

Wessberg, J., Stambaugh, C., Kralik, J., Beck, P., Laubach, M., Chapin, J., Kim, J., Biggs, S. J., Srinivasan, M. & Nicolelis, M. (2000), 'Real-time prediction of hand trajectory by ensembles of cortical neurons in primates', *Nature* **408**(6810), 361–377.

Wooldridge, M. (2000), 'On the sources of complexity in agent design', *applied artificial intelligence* **14**, 623–644.

Wooldridge, M. & Jennings, N. (1994), Agent theories, architectures and languages: A survey, *in* 'Intelligent Agents', Springer-Verlag, pp. 1–32.

Wooldridge, M. & Jennings, N. (1995), 'Intelligent agents: Theory and practice', *Knowledge engineering review* **10**(2), 115–152.

Yen, J. & Pfluger, N. (1995), 'A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation', *IEEE Transactions on Systems, Man and Cybernetics* **25**(6), 971–978.

Zhang, W. (1999), Adjustable autonomy for manufacturing cell control systems, *in* 'Proceedings of AAAI Spring Symposium on Agents with Adjustable Autonomy', p. 136.