A Decision-Mechanism for Reactive and Coordinated Agents

Silvia Coradeschi

—

Department of Computer and Information Science
Linköping University
Linköping, Sweden
E-Mail: silco@ida.liu.se

April 22, 1997

ABSTRACT

In this thesis we present preliminary results in the development of LIBRA (LInköping Behavior Representation for Agents). LIBRA is a rule-based system for specifying the behavior of automated agents that combine reactivity to an uncertain and rapidly changing environment and coordination. A central requirement is that the behavior specification should be made by users who are not computer and AI specialists. Two application domains are considered: air-combat simulation and a simulated soccer domain from the perspective of the RoboCup competition. The behavior of an agent is specified in terms of prioritized production rules organized in a decision tree. Coordinated behaviors are encoded in the decision trees of the individual agents. The agents initiate tactics depending on the situation and recognize the tactics that the other team members have initiated in order to take their part in it. What links individual behavior descriptions together are explicit communication and common means to describe the situation the agents find themselves in.

Acknowledgments

First of all, I would like to express my thanks to my supervisor, Anders Törne, for having given me the possibility of working in an interesting and fruitful project and for support and guidance. I would also like to thank my second supervisor, Dimiter Driankov, for the useful comments on this thesis and support. I am specially grateful to Erik Sandewall for having given me the opportunity to be part of a stimulating research environment as the one in IDA, and for his guidance and support during all this period.

I would like to thank Patrick Lambrix for useful comments and discussions about this work and Ivan Rankin for proof-reading the thesis and helping to improve the English.

My partner in work and life, Lars Karlsson, has given a great contribution to this work, both co-authoring several of the papers that form the basis of this thesis and keeping me in the best mood while I was writing them.

This work would have not been possible without the collaboration of Saab. I would like to thank the personnel in Saab: Jennie Andersson, Johan Görsjö, and Gunnar Frisk for support and discussion. In particular I would like to thank Jennie Andersson for her patience and support in solving the technical problems during the interface between our system and the Saab system and for the discussions and comments about this work.

I am also indebted to Milind Tambe, University of Southern California, and Gary Eiserman, Decision-Science Application Inc, for the help in understanding two of the systems presented in this thesis: TacAir-Soar and Brawler. I am really grateful for their patience in answering all my questions.

Finally, I would like to thank my colleagues in RTSLAB and RKLLAB and the rest of IDA for providing a stimulating research environment. Special thank goes to the administrative and technical staff; particularly Anne Eskilsson, Lillemor Wallgren, Lise-Lotte Svensson, Leif Finmo and Gunilla Fåk.

Contents

Chapter 1

Introduction

Multi-agent systems constitute one of the current promising fields in Computer Science research. Their main application is in simulation environments where a number of agents can interact with each other using coordination, communication, and plan recognition. Applications of such environments are in areas as testing of products [1], training [20], internet [14], study of economic markets [27], large-scale battlefield simulations [37], education [24] and [13], and entertainment [3].

In this thesis we present a system, LInköping Behavior Representation for Agent (LIBRA), for the specification of behavior of agents that combine reactivity to an uncertain and dynamic environment, with coordination with other agents. The agent behavior should be robust in changing scenarios. We would like to specify behavior in such a general way as to make it applicable in different scenarios and situations.[1]

Another issue we are especially interested in is the user friendliness of the specification. Normally experts in the domain can specify the behavior of the agents, but this is then encoded by system experts. In our approach the experts in the domain are intended to specify the behavior of each agent directly, without the aid of a system expert. This has the advantage of letting the domain experts directly provide the specification the automated agents should follow, test the behavior, and change the specification in case the automated agents do not behave as expected. It also makes it possible to tailor the behavior of the automated agents for testing specific features

[1]By scenario we mean a description of the general characteristics of the agents involved in the simulation and of the environment, for example the kind of aircraft, the number of agents involved, the zone in which they are flying. By situation we intend the state of the world at a particular moment of the simulation.

of a product, or training for particular situations. Furthermore, it makes it easier to specify and change the behavior of the automated agent depending on the role they have in the simulation. On the other hand this puts special requirements on the user aspects of the system, both with respect to learning its features and working with it.

Our current target is behavior specification of automated pilots in battlefield simulation environments. Human pilots are in general guided in their actions by well-defined tactics, but are also able to react to unexpected situations and to adapt general tactics to specific cases. The challenge is to develop this same capability in automated agents. Thus these can react in a flexible way to uncertain and dynamic environments, but still follow predefined tactics just like human pilots.

We have also applied the system in a simulated soccer domain in the context of a world-wide competition of simulated soccer teams (RoboCup) [21], that will be held in IJCAI'97.

To give a better understanding of this thesis, in the next section we give a brief account of the context in which the project has been conceived and developed. Then we present an overview of LIBRA and the reasons for applying the system to the RoboCup application. Finally we describe the structure of the thesis.

1.1 History of and motivation for the project

This work has been developed in the context of a collaboration project between Linköping University and Saab Military Aircraft AB, Sweden with the aim of investigating the design and implementation of automated agents for the air combat domain. Both at Saab and at Linköping University there is great interest in the multi-agent domain. The department for Mission and System Analysis at Saab has been working since 1990 on a system known as TACSI (TACtical SImulation) [1] for simulating air combat, in particular beyond-visual-range combat.[2] The system was created mainly for evaluating aircraft, missiles and tactics and is currently used at Saab for evaluating their products. In 1995 the work on TACSI became more intense and two students at Linköping University started their master's thesis at Saab initiating cooperation between Saab and the university. The aim of one master's thesis was to investigate the use of state-hierarchies to describe the tactical

[2]In beyond-visual-range combat the aircraft can see each other via radar, but not directly.

behavior of an agent [2] and the aim of the other was to extend the master's thesis mentioned to deal with command and control problems [17].

From the perspective of Linköping University there was an interest in increasing an understanding of the formal framework needed to create realistic behaviors in the simulation of multi-agent systems. The domain offered by Saab was highly suitable for the study of multi-agent systems in complex and real-time environments.

The aim of the project of which this work is a part has therefore been the specification of behaviors of agents that can act in complex and real-time environments and that can be tested in the air-combat domain offered by Saab.

1.2 Overview of LIBRA

An agent in LIBRA acts in an environment, receiving information and interacting with other agents present in the environment. It has a state where it can store information needed for making decision and a decision-tree that specifies its behavior. A decision-tree consists of a hierarchy of decisions which leads to actions that the agent will consider for execution. At every step of the simulation the decision-tree is visited and as a result of the visit a list of actions is created. To this list are added the actions still in progress. Some of these actions are mutually exclusive and the actions that are actually being performed are chosen depending on dynamically changing priorities. Some actions can be performed concurrently and in this case the agent performs them in parallel. The agent can also decide to perform sequences of actions; in any case the agent is not committed to complete the sequence of actions as the sequence can be interrupted if other actions are more compelling.

The system was presented in July '96 at the 6th Conference on Computer Generated Forces and Behavioral Representation in Orlando [12] on the part regarding the air-combat application and in December '96 at the "RoboCup Workshop: Soccer as a problem for Multi-Agent Systems" at the Second International Conference on Multi-Agent Systems (ICMAS-96) [10] on the part regarding the soccer application.

1.2.1 Communication-poor coordination

In both the soccer domain and the air combat domain, coordination between agents is essential, but it cannot be achieved just by communication and negotiation [22]. In fact, giving the real-time constrains in these domains, the time for communication is limited and the presence of hostile agents makes communication risky and unreliable (jamming). Several techniques have been developed for communication-poor coordination (see sections 2.5, 2.2.4 and 2.4.3). In LIBRA the agent initiates tactics depending on the situation and recognizes the tactics that the other team members have initiated in order to play its part in it. The knowledge required for applying the tactics is coded in the decision-tree that specifies the behavior of the agent and the observations of the situation and of the actions of other agents influence the decisions about which tactics to adopt. Explicit communication is performed through communication actions present in the decision-tree.

1.2.2 Autonomous vs automated agents

The agents that can currently be specified in LIBRA are automated agents, that is agents that act following the specification that has been given to them. Our long-term aim is to specify autonomous agents, that is agents that can show a certain degree of independence and react appropriately to unexpected situations. To have agents that are really autonomous and that have a really robust behavior it would be necessary to add a deliberative component to the system, which could deal, for example, with planning, evaluation of situations and prediction, handling of errors and exceptions. In the context of the classical three-layer architecture for autonomous agents with one deliberative layer, one reactive, rule-based layer and one processing layer for the world interface, the work in this thesis is more relevant in the context of the rule-based layer. We consider this work as a starting point for examining a combination of reactive and deliberative features in multi-agent real-time environment.

When we use the world *agent* in the thesis, we mean *automated agent.*

1.3 The RoboCup application

Together with the air-combat application we have also considered an application related to simulated soccer player in the context of the Robot World Cup Initiative (RoboCup). RoboCup is an attempt to provide a standard problem, a soccer game, where a wide range of technologies can be integrated and examined. The RoboCup organizers provide a simulator for the movements of the players and the ball and each participant provides the “brains” of the players. The first RoboCup competition will be held at IJCAI-97

(International Joint Conference on Artificial Intelligence), Nagoya, Japan, August 23 - 29, 1997.

The reasons why we have considered this alternative application for our system are several. First of all we were interested in examining how easily our system could be applied to another domain. Furthermore RoboCup offered the opportunity to compare our system with other multi-agent systems currently developed and to make in contact with other researchers in the multi-agent domain. Finally, the soccer domain provided the opportunity to specify and test tactics in a domain well known to us, while specifying complex tactics in the air-combat domain requires the help of air-combat experts.

1.4 Structure of the thesis

The rest of this thesis is organized as follows:

Chapter 2 presents related work in the air-combat domain. The field is relatively new and need an overview. To give a general view of the area and to use it as a background for our work, we present the main characteristics of the three most relevant systems in the air combat domain. We also present work in multi-agent domains involving communication-poor coordination as this is mainly the kind of coordination we have been interested in LIBRA.

Chapter 3 presents the current implementation of the TACSI system and some future developments. The TACSI system has been our starting point in this work and we have been using the simulation part of the system to test LIBRA.

Chapter 4 provides a description of LIBRA. Our main contribution in terms of LIBRA is a decision-mechanism for automated agents that is relatively simple and easy to specify and that combines reactivity to a highly dynamic environment with coordination between agents.

Chapter 5 considers the air-combat application in more detail. An example involving six aircraft is presented. Further, an example of communication between agents is considered.

Chapter 6 presents the extension of LIBRA to the soccer domain. The RoboCup competition is presented and an example of coordination between agents is given.

Chapter 7 contains a concluding discussion and draws up the lines for future research.

Chapter 2

Air-combat applications and related multi-agent systems

2.1 Introduction

In this chapter we present three systems for aircraft combat simulations: TacAir-Soar, Brawler and SWARMM. The aim of all three systems is to model an automated pilot that acts like a human pilot would act in the same circumstances.

TacAir-Soar has been developed within Soar, a software architecture developed as a basis for modeling general intelligence. Its aim is to produce intelligent automated agents for battlefield-simulated environments whose behavior is indistinguishable from human behavior at least in the air-combat context.

The main characteristic of the system is an integration of goal-driven and reactive behavior. The automated agent should react quickly to change in the complex, dynamic and uncertain battlefield-simulated environment, but on the other hand cannot base its behavior just on reaction. It should also pursue goals via plans. The Soar architecture gives support for knowledge representation, problem solving, reactivity, external interaction, and learning. TacAir-Soar has been created by extending Soar with a perceptual and motor interface which allows the automated pilot to fly an aircraft provided by ModSAF in environments based on *Distributed Interactive Simulation technology* (DIS). ModSAF is a distributed simulator used by the U.S. military. Specific knowledge about the air-combat domain and strategies for making Soar more suitable for the air-combat application have also been added. TacAir-Soar addresses important issues in air-combat simulation such as coordination between agents, communication and agent modeling.

Brawler has been developed by Decision-science Applications Inc. and according to [19] has become a standard for high fidelity air-to-air engagement modeling within the U.S. Air Force, U.S. Navy, airframe manufacturing and avionics communities.

Brawler models both within and beyond-visual-range aircraft combat. It also has a high level of detail in modeling aircraft, weapons, avionics and countermeasures. It has the aim of providing a realistic model of the human decision processes using a *value-driven decision making* approach and an *information-oriented decision architecture*. In a value-driven decision making approach the decision among alternative courses of actions is taken on the basis of an evaluation of the situation that the simulated pilot believes will be the result if this particular alternative is chosen. An information-oriented decision architecture means that the flow of information into and from the mental model of the world of the pilot is explicitly modeled.

SWARMM has been developed by the Australian Artificial Intelligence Institute (AAII) to simulate the physics and the pilot reasoning involved in air missions. The reasoning of the pilot is modeled with dMARS, a commercial product that implements an agent-oriented distributed real-time system and that is an updated version of the Procedural Reasoning System (PRS) [16] developed by SRI International (California).

In SWARMM the agents are characterized by their beliefs, goals and intentions. The goals are described in terms of what tasks and behaviors the agents intend to execute and are expressed as conditions over some interval of time. An intention consists of an initial plan and of the subplans that are used to execute the plan. Only one intention at a time is active, but other intentions can be suspended or deferred, or can wait for some conditions to become true in order to be activated.

In sections 2.2-2.4 we present the three systems just mentioned and in section 2.6 we discuss them in general terms.

In both the soccer and the air combat domain coordination between agents is essential, but given the real-time constraints and the presence of hostile agents it cannot be achieved just by communication and negotiation [22]. In section 2.2.4 and 2.4.3 we consider how the problem of coordination in a team has been treated in TacAirSoar and in SWARMM. In section 2.5 we consider other techniques that have been developed for communication-poor coordination in other context than the air-combat domain. The techniques we consider are a physics-oriented approach to coordination [32], a focal points technique [15], and observation-based coordination [18].

2.2 The TacAir-Soar project

2.2.1 Introduction

The aim of the TacAir-Soar project is to develop intelligent agents that simulate pilots in battlefield-simulation environment. Most of the effort in the project has been concentrated on beyond-visual-range combat, i.e. combat between aircraft that can see each other just with the help of onboard instruments. Currently some other forms of combat are also studied, for example helicopter against ground vehicles [34], [35].

TacAir-Soar is being developed by three universities: the University of Southern California, the University of Michigan and Carnegie Mellon University.

The key requirement for an automated pilot in TacAir-Soar is that the other entities in the simulation should be able to interact with it in the same way they would interact with humans. This is particularly important as the TacAir-Soar pilots should be able to participate in simulated tactical exercises with human pilots. In 1997 TacAir-Soar pilots will be involved in a large-scale simulated tactical exercise, the Simulated Theater of War '97, that will involve 50,000 entities, partly humans and partly simulated.

The TacAir-Soar project has been developed within the Soar architecture. Soar is a general architecture used to model human behavior. In particular it aims to form a basis for general intelligence [30]. In the next section we present the Soar architecture. Then in section 2.2.3 we examine the decision-making process in TacAir-Soar and in particular the combination of goal-driven behavior and reactivity. Finally we consider how TacAir-Soar deals with coordination and communication between agents, agent modeling and teamwork. Most of the information given in this section is taken from [37], [34], [30] and [35].

2.2.2 The Soar architecture

In this section we sketch some of the characteristics of Soar that are important for understanding how TacAir-Soar works.

Soar is structured on three interrelated levels: memory, goal hierarchy and decision level. The aim is to achieve goals in a *problem space*. A problem space determines the set of states and operators that can be used to achieve a goal. The states represent situations and the operators represent actions.

Given a goal, a problem space and a state, operators are proposed, selected and applied depending on the system knowledge. Soar's knowledge is stored in the form of production rules, i.e. rules that perform an action given a satisfied condition. The *Operator-selection productions* generate symbolic preferences among the operators, given the current goal and state. Acceptability preferences determine which operators should be considered as candidates. Desirability preferences determine the relative desirability of candidate operators. Desiderabilty preferences can express that the operator is at least as good as any other operator, that it is worse than any other operator and that an operator is better or worse than another operator.

The *Operator-application productions* apply the selected operators. The application of the operators results in modification of the state and in some cases in generation of output commands. The *Operator-termination productions* terminate the operator if the state indicates that the operator has achieved its result. The productions fire in response to changes in the goals and in the states. All the applicable productions fire in parallel without conflict resolution.

The decision level has two phases. In the first phase, called the elaboration phase, all the applicable productions fire, thereby retrieving all the knowledge relevant to the current decision. When no more productions can fire, the second phase, the decision phase, follows. In this phase one of the actions retrieved in the first phase is selected on the basis of preferences generated by the productions.

If sufficient knowledge is not available for selecting the next operator to apply, an impasse occurs and a subgoal is generated. An impasse can occur in two cases. In the first case there are no alternatives that can be selected, and in this case the subgoal is to apply the operator selected in the previous cycle, for example decomposing it. In the second case there are multiple alternatives that can be selected but the preferences can not discriminate which one to choose. In this case a subgoal is generated to determine the operator to select. The creation of subgoals generates a goal hierarchy. The subgoals disappear when the associated impasse is solved. With each goal a problem space is associated in which the goal can be solved. Each problem space has its state which includes a representation of its supergoals and their states. The sensor data from the external environment are included in the top state and are therefore accessible in all problem spaces.

2.2.3 The decision-making process in TacAir-Soar

We explain how the decision-making process in Soar works in TacAir-Soar by means of an example. Figure 2.1 shows the problem-space hierarchy for

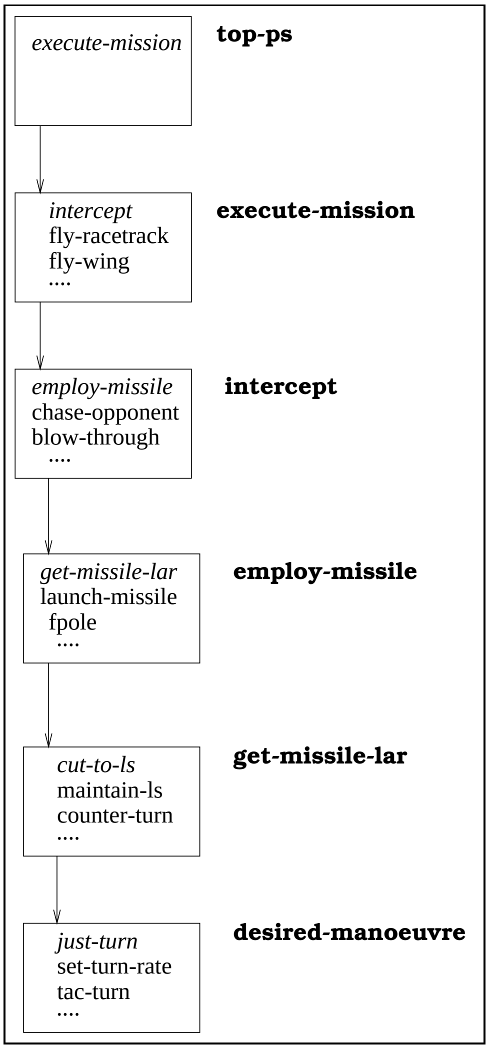

Figure 2.1: Example of Problem-Space Hierarchy

an automated pilot. **Top-PS** is the top-most problem space; in this problem space the *execute-mission* operator is applied. In this example the mission is to protected the home base. The execute-mission operator terminates after that the automated pilot has protected its home base for a prespecified time period. As this time period, in our example, has not elapsed, a subgoal is generated in order to complete the application of *execute-mission*. Associated with this subgoal the **execute-mission** problem-space is selected and in this problem-space the *intercept* operator is selected. The termination condition for this operator is the elimination of the opponent. As this condition has not been achieved yet, the subgoal of completing the intercept is generated, the **intercept** problem-space for this subgoal is generated and the *employ-missile* operator is selected. The generation of subgoals and the selection of operators goes on until the *just-turn* operator is selected that applies directly and issues a command that causes the aircraft to turn.

This example has mostly demonstrated goal-oriented behavior of TacAir-Soar pilots. However, as the battlefield environment is essentially dynamic and uncertain, reactive behavior is also required. This is obtained by constantly monitoring the external environment. Input from sensors are constantly tested by productions and inferences are drawn from other knowledge about collections of sensor data, current goals and operators being pursued and potential actions to take. Changes in the situation can provoke suggestions for new operators at any level in the goal hierarchy, suggestions for external actions, generation of preferences among operators and termination of operators. The decision-making procedure then determines the changes to be made to the goal and operator hierarchy given all generated suggestions and preferences.

2.2.4 Coordination and communication between agents

Coordination and communication between agents is in general a difficult problem, but it is simplified in this domain by shared knowledge of common doctrine and tactics and detailed mission specifications. The mission specification is given in the briefing session before the mission. In the briefing session mission-related parameters are specified such as the pilot's role, initial flight formation, numbers and types of missiles, rules of engagement and commit criteria. The rules of engagement are directives about the general conduct during the mission, for example that before firing against an aircraft this should be positively identified as a military enemy aircraft. The commit criteria specify whether to intercept an enemy aircraft.

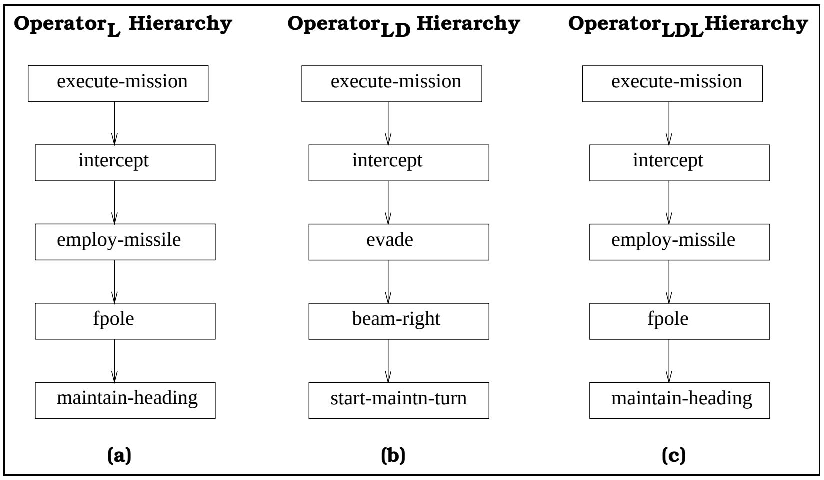

Figure 2.2: Operator Hierarchy for recursive tracking

There is still a need for communication between agents. Actually a fixed set of message templates is provided, but the intention is to use natural language in the future.

Coordinated behavior is achieved in TacAir-Soar using shared tactics and doctrine represented in the form of rules and explicit communication in the form of simulated radio messages. All the information relevant for the mission, given in the mission briefing, is entered via an editor. There is also a limited use of coordination through observed behavior. For example the wingman responds to small turns by the section leader.

The types of coordination implemented within TacAir-Soar are [22]: manoeuvring coordination (for example joining up in formation, flying in formation, changing formation), coordinated manoeuvring, tactical manoeuvring and control of the section (for example changing the leader or committing to an enemy). Radar coordination, targeting and sorting the enemies is also implemented.

2.2.5 Agent tracking

A very important challenge for an automated pilot is to try to infer the unobserved actions, the plans, the goals and the behaviors of the opponents by monitoring their observable actions. The key idea is that all agents possess similar types of knowledge, goals, and levels of complexity in their behaviors (see [33], [38], [39], [34]). The agent uses a copy of its own problem-spaces and operators to track the opponent's actions and behaviors. On the basis of this copy the agent checks the actions of the opponent and tries to infer unobserved actions, high-level goals and behaviors. The agent quickly commits to a single interpretation of the on-going actions and uses the interpretation to decide about future actions. Backtracking is used when the interpretation leads to inconsistency.

Agent tracking involves recursive tracking, i.e. the agent should consider that the enemy is tracking him and acts in consequence. In the case of recursive tracking the agent uses a first copy of its own problem-spaces and operators to track the opponent's actions and behaviors and a second copy for tracking the tracking of himself by his enemy. Figure 2.2 shows the operator hierarchies that the automated pilot L maintains for tracking his opponent D: (a) is the operator hierarchy of L, (b) is the operator hierarchy that L thinks that D has and finally (c) is the operator hierarchy that, in the opinion of L, D has of L. These operator hierarchies influence the decision that the automated pilot L takes. As an agent develops the operator hierarchies in real-time, it can be problematic to maintain operator hierarchies for tracking several opponents. This problem is addressed in [33] and optimizations involving sharing of near-identical models of opponents are considered.

The tracking of teams is considered in [34]. A similar technique to the one used for tracking single agents is used, but the operator hierarchies in this case also contain team operators. Teams operators are presented in the next section.

Agent tracking is still experimental and it is not fully used in the TacAir-Soar agents.

2.2.6 Teamwork

In [35] the problem of commitment in a team is considered. The central idea is that for effective teamwork, agents should have team plans and an underlying model of teamwork in which commitments and responsibilities of members of a team are explicitly outlined. This can help in increasing the robustness of the team behavior in the presence of unanticipated events.

The team model relies on the joint intentions framework of Cohen and Levesque [6]. In this framework the agents in a team have a joint goal of completing a team action. The joint goal holds if all team members believe

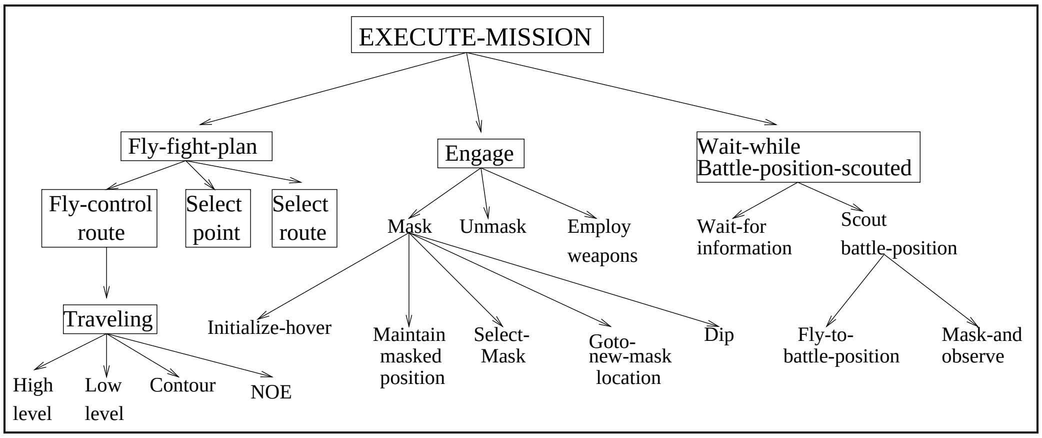

Figure 2.3: Operator hierarchy of a pilot with team operators (the ones in boxes)

that the goal is currently false and that it will be eventually true. If a member discovers that a goal is achieved, unachievable or irrelevant, he should share this knowledge with the other members of the team before the interruption of the joint commitment.

In figure 2.3 an operator hierarchy is shown where there are also team operators (those in boxes). To establish a joint intention each team member selects a team operator. The team operator can be terminated by updating the mutual beliefs of the team. This in general involves communication, but an evaluation of the cost and benefits of the communication is also performed.

The teamwork capability is currently fully used in the TacAir-Soar agents.

2.2.7 Comments

TacAir-Soar is a powerful system for air-combat simulation that includes capabilities such as coordination, communication, teamwork, and agent tracking. TacAir-Soar is part of Soar, a large-scale project for modeling general intelligence. This allows gives the possibility to have a large number of rules, as a lot of effort has been put in Soar to optimize the procedure for selecting rules. It also make it natural to add new capabilities in the system developed independently within Soar, such as learning, planning and natural language.

Several capabilities of TacAir-Soar such as agent tracking and teamwork are still not part of LIBRA. In the rest of the section we consider some features that are comparable between the two systems and that differentiate them.

The main difference between LIBRA and TacAir-Soar is that in the latter there is a hierarchy of goals and subgoals that is created during the execution by firing of rules. In our case we do not have a hierarchy of goals, but rules whose conditions are organized in a hierarchical fashion. There is a superficial similarity between the two hierarchies, but they are substantially different. In TacAir-Soar the agent has at all times a goal. This goal can be subdivided into subgoals by rules or can also lead to the execution of an action through the firing of rules. If the goal is subdivided, then one subgoal is selected which in turn can be subdivided or leading to actions. The agent is seen as being at any given time in one point of the goal hierarchy and acts accordingly.

In LIBRA all the rules of the agent are checked at each step of the simulation.[1] When a rule fires, it produces an action that becomes a candidate to be performed and that competes with the actions selected in other rules and the actions that are already going on. The compatible actions with higher priority are the actions that are going to be performed in the next step of the simulation. The LIBRA system reacts fast to any change in the situation as it reevaluates what actions to perform at each step of the simulation, while the TacAir-Soar is mostly oriented toward long term commitment. The intention is to introduce a higher level of commitment to a long-term course of actions in LIBRA without losing reactivity to changes in the environment (section 7.2.1).

In TacAir-Soar preferences between actions are performed mainly through a direct comparison between pairs of actions. In LIBRA the preference between actions is carried out on the basis of dynamically changing numerical values (priorities) attributed to the actions. The priority values are, of course, related to the preference of one action over another, but they do not require a direct comparison between pairs of actions. In LIBRA two methods can be used to assign priorities. In one method the values are attributed with the aim of creating an order between the candidate actions and this has some similarity with the what is done in TacAir-Soar. However, it is also

[1]This can be done efficiently given the hierarchical organization of the conditions in the rules, section 4.5.1. The number of rules in our system is also much less than in TacAir-Soar as we do not have rules dealing with the goal hierarchy.

possible to attribute the priority values on the basis of the importance of the action per se and not in relation to other actions that could possibly be candidates. In practice we have experienced that a combination of the two methods is used. The use of numerical values can also facilitate the modeling of the fact that the priority values are influenced by different aspects of the world. In LIBRA the actions have a basic priority value and numerical values can be added or subtracted from the basic priority value during the process of selection of the actions, given the satisfaction of conditions.

Another important feature that differentiates TacAir-Soar and LIBRA is the attention to the user aspect of the system. In TacAir-Soar the developers of the system specify the behaviors of the agents and thus the user aspect of the system has not received special attention, while in LIBRA the user aspect plays an important role in the development of the system. TacAir-Soar is a complex system in which a large number of rules seem necessary for specifying behaviors. In fact rules should be written, for example to construct and update the goal hierarchy, to establish preferences, and to execute actions. In LIBRA we have tried to keep the system relatively simple and to collect all the information necessary for the specification of the behavior, including preferences and dynamic changes of preferences, in a single structure, the decision-tree. The decision-tree structure is, in our experience, a natural and compact way to organize rules. We expect that the ease in specifying behaviors will further increase if graphical tools are used to write the decision-trees. We intend to add a graphical interface to the system in the near future.

2.3 The Brawler project

2.3.1 Introduction

The Brawler model has been developed by Decision-Science Applications Inc. under US Air Force sponsorship as a simulation tool for providing a detailed representation of air-to-air combat, both within- and beyond-visual-range.

It is used for combat exercises and tests, mainly in situations in which live tests are precluded by considerations of cost, time, safety or unavailability of hardware. A typical application would be performance studies for airframes, avionics, weapons, and tactics. It can also be used as a pilot training aid and to provide automated opponents in simulations involving humans and computer generated forces.

Brawler achieves a high level of detail in the modeling of aircraft, weapons systems, avionics and countermeasures. Special emphasis has been placed on modeling human decision processes and it is this last aspect that we mostly consider in the following sections. The same techniques for modeling human decision processes have also been successfully used in modeling pilots for helicopters [4]. What we are presenting in these pages is taken from [19] and [23].

2.3.2 The decision-making process

The two principal features in Brawler's approach to modeling human decision processes are an *information-oriented decision architecture* and *value-driven decision making*.

The first consists of an explicit simulation of the flow of information into and out of each pilot's personal mental model of the situation. In figure 2.4 the conceptual representation of information flow is shown. The *central status arrays* represent the true physical state of the simulation, for example variables describing aircraft positions and velocity, quantity of fuel left, positions and orientations of missiles and so on. Via a consciousness event the information arrives at the *mental status array* that represents the "mind" of the automated pilot. A consciousness event includes the activities of situation assessment, decision making, and action execution. The mental status array mirrors the central status, but imperfectly. The pilot does not know, for instance, the exact velocity of the other aircraft and cannot in general detect missiles on the radar. Decisions of the pilot cause manoeuver events and weapons events.

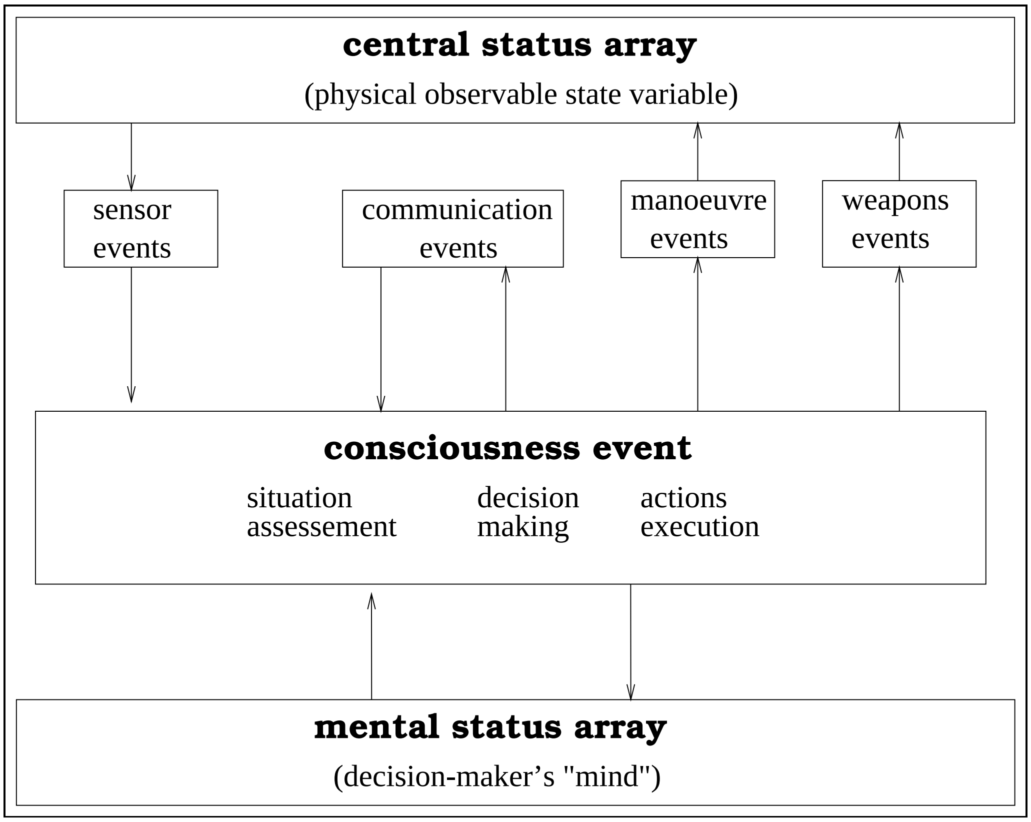

Figure 2.4: Representation of Information Flow

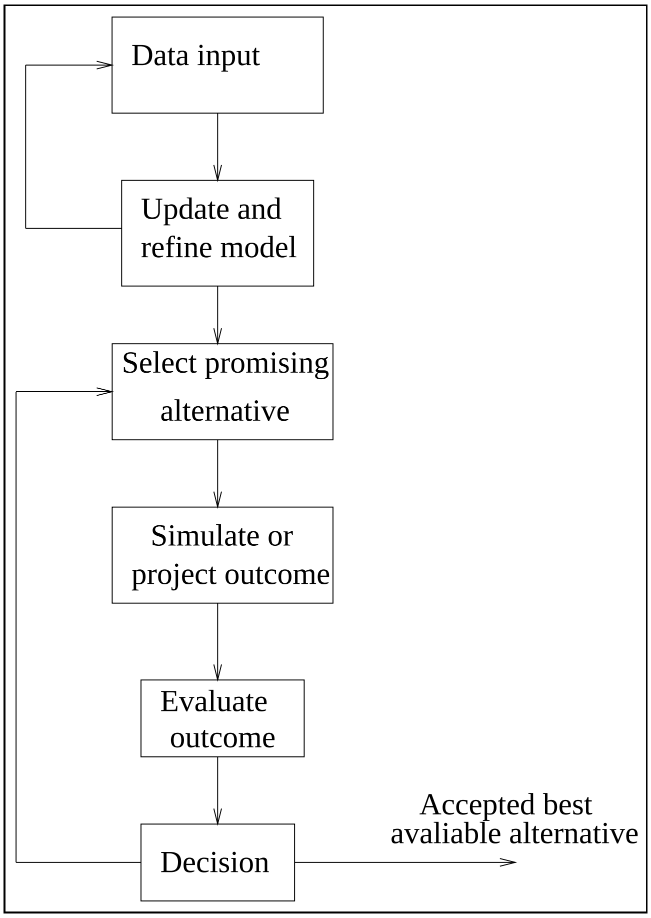

Figure 2.5: Value-driven decision-making process

In figure 2.5 the value-driven decision-making process is shown. The upper loop is executed when the pilot receives sensor or communication events. In this case values are assigned to variables that describe the situation. There is a difference in the frequency in which the variables are updated. The variables describing the physical state, aircraft trajectories for instance, are updated at least once per second. Higher level variables, for example variables describing to what degree another aircraft is perceived to be a threat, can be updated every three to five seconds.

The lower loop is executed when a decision is made. When what is called a consciousness event occurs, the automated pilot considers alternative courses of actions and traverses the lower loop for each of the alternatives. The loop consists of considering each of the alternatives, predicting in the mental model what the situation will be in the near future (two to ten seconds) if the alternative is taken and giving a numerical score to the situation. The alternative with the highest score is chosen. The scores are given by a *value function.*The value function is calculated as a sum of components, each of which reflects some aspects of the real world that can affect air combat outcomes. Each component is a product of a function that is sensitive to the real world feature and a dynamically changed *importance multiplier.* For example, the value function for manoeuver decisions evaluates the result of a manoeuver considering the ability to provide mutual support, to avoid collision with the ground, to see other aircraft and so on.

The decisions are reconsidered quite often, at least once per second for decisions that influence the physical state of the aircraft and at longer intervals for decisions made at higher level. Unexpected events can anyhow trigger early reconsideration of the decisions.

The decisions are hierarchical. High-level decisions influence the evaluation function of lower-level decisions and which alternative actions are considered at lower levels.

Figure 2.6 shows the Brawler decision hierarchy. The *flight posture decision* determines the general course of action and depends on a broad description of the situation and user-supplied combat priority. The *flight tactics decision* is made by the flight leader and it is sent to the other member of the mission. The message influences the values used by the other pilots for scoring alternative actions by incrementing or decrementing “importance” multipliers.

The *pilot posture decision* determines the general course of actions of the pilot, if he is covering the leader, attacking, evading and so on. It determines the radar mode and the weapon employed. The pilot posture decision can

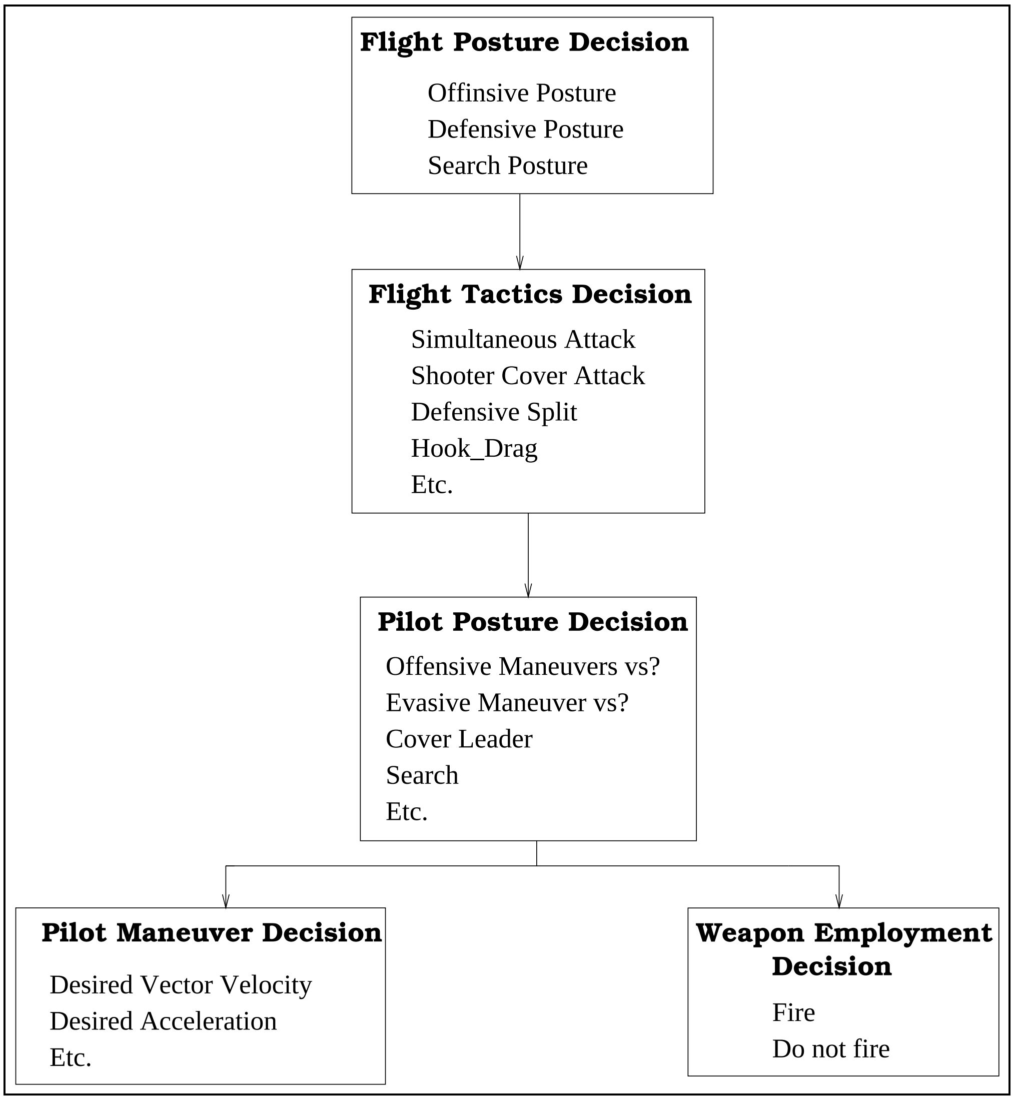

Figure 2.6: Brawler Decision Hierarchy

also simplify the scoring evaluation and reduce the number of manoeuvers that are considered, recognizing that some features are not important in the current situation.

At the lowest level of the decision hierarchy are the *pilot manoeuver decision* and the *weapon employment decision*. The former is responsible for choosing the actual maneuver to do, given all the previous decision, and the latter decides whether to fire a weapon in the present moment, given a prior decision about which weapon to use.

Orders are represented as increments and decrements of "importance" multipliers: they do not force an action upon the pilot, but they just influence the pilot mental model. This allows the pilot to hedge the orders against other considerations, for example even if it has been ordered to attack a specific aircraft, it will not miss the opportunity of shooting down another aircraft if he has this possibility.

The pilot, in addition to the decisions shown in figure 2.6, makes decisions related to information acquisition: *sector search decision*, which directs the pilot visual attention, and *information request decision*.

The automated pilot can incorporate visual information in his mental model if it is actually looking in the right direction. In order to model the searching process the visual field is divided into sectors. The radar and other sensors are considered alternate sectors. A priority value is dynamically assigned to each of the sectors. It depends on the intrinsic value of the sector, the time since the sector was last searched and the belief of the pilot that an aircraft is in the sector. The sector search decision determines which sector is searched on the basis of the priority values. Other factors that can influence the visual detection of an aircraft are considered, for example the cockpit masking, the contrast to background, the pilot visual acuity and whether the pilot is already aware of the existence of the other aircraft.

The information request decision determines whether to broadcast a request of information about an aircraft that has not been seen for a while. It depends on the time since the last observation and the time since a similar request was last sent and also whether the aircraft is a friend or an enemy.

2.3.3 Comments

The most interesting aspect of Brawler from our perspective is the high flexibility in making decisions. In fact preferences between several alternatives can be made at every level of the decision hierarchy. The preferences are not statically fixed, but depend on the interactions among important aspects of the current situation and potential future situations that the respective alternative can lead to. The way of modeling orders is also interesting, as it gives the automated pilot the possibility to evaluate the order, given its better knowledge of the situation with respect to the flight leader.

The system seems to reach a high level of details in modeling the hardware of the aircraft and the flow of information in and out of the mental model of the pilot and in the same time accurately models pilot and flight leader decisions.

The aspect that strongly differentiates LIBRA from Brawler is the fact that the decision-making process of Brawler includes prediction and evaluation of what the situation will be like in the near future if a particular decision is made. We consider the possibility of having prediction and evaluation of major interest and we intend to study the possibility of adding this feature by developing a deliberative layer in LIBRA.

2.4 The SWARMM system

The purpose of the Smart Whole AiR Mission Model (SWARMM) system is to simulate the physics and the pilot reasoning involved in air missions. The type of tasks considered are: air defense, attack, escort and sweep (the use of a fighter to clear a path for an incoming strike force). The SWARMM system has been developed at the Australian Artificial Intelligence Institute (AAII) interfacing a system for modeling the physical systems, Piloted Air Combat AUstralia (PACAUS), and the Distributed Multi-Agent Reasoning System (dMARS), a commercial product that implements an agent-oriented distributed real-time system. The PACAUS system models the aircraft, the radar, weapon and communication systems, and also the pilot's body, for example the sight and the effect of the manoeuvers on the pilot's physical state. The dMARS component models the reasoning of the pilot and it is described in the next section. In the context of the SWARMM project team modeling and plan recognition have been considered. We consider these issues in the last two sections of this chapter.

The information given in this section is taken from [28], [40] and [16].

2.4.1 Distributed Multi-Agent Reasoning System (dMARS)

The dMARS system is an updated version of the Procedural Reasoning System (PRS) [16] developed by SRI International (California).

The agents are characterized by their beliefs, goals and intentions. The beliefs are represented in a first-order logic and give information about the environment from the perspective of what the agent has perceived. The goals describe the desired tasks and behaviors of the agents and are expressed as conditions over some interval of time. The type of goals that can be expressed are the goals to achieve a condition, to maintain and to test a condition. The agent has an intention list which contains the tasks that have been chosen by the system for execution. Only one intention at a time is active, but other intentions can be present in the list. These intentions can be suspended or deferred, or can wait for some conditions to be true in order to be activated. Which intention is active at a time point depends on the beliefs of the agent and on the priority of the intention. An intention consists of an initial plan and of the subplans that are used to execute the plan. A plan is composed by a body, an invocation condition and a context condition. A plan can be executed if the invocation condition is satisfied. The body is represented as a graph. In the arcs of the graph are the sub-goals to be achieved in the

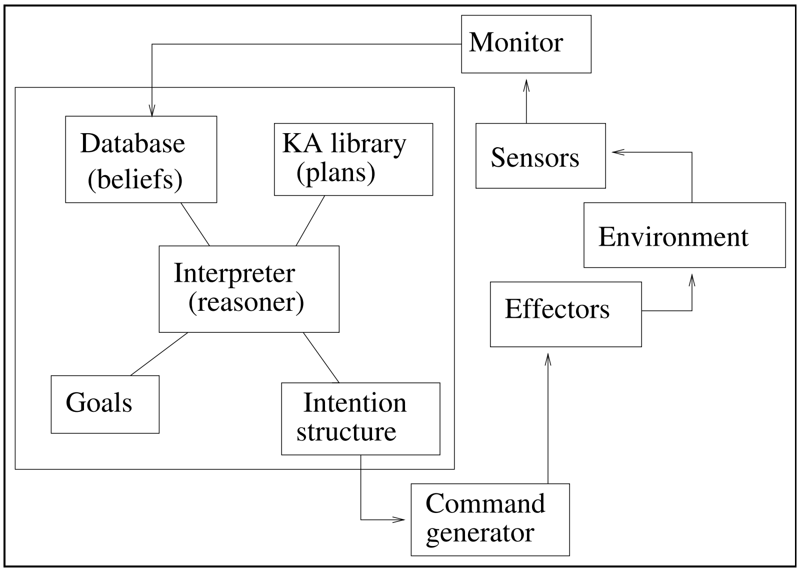

Figure 2.7: System structure

execution of the plan. The context condition contains information relevant for the execution of the plan. In the next section we consider the decision mechanism more in details.

2.4.2 The decision-mechanism

The structure of the system is shown in figure 2.7. On the basis of the present beliefs and goals the interpreter selects the plan to be executed and places them in the intention structure. An intention in the intention structure is selected for further execution. The intention which is executing can produce either new subgoals to be inserted into the goal structure or an action. The execution of an action can effect the external world and also the internal state of the system. An action can, in fact, operate on the beliefs, goals and intentions and can affect the system as a result of the knowledge acquired interacting with the world. The interpreter cycle then starts again. If new goals and beliefs have arisen, new plans can be selected. Otherwise, the system goes on trying to fulfill the intentions previously selected.

The selection of the plans are made by the interpreter following two methods. First in the absence of sufficient decision knowledge the interpreter executes fixed decision-making processes bounded in execution time. Second these decisions can be overridden by metalevel procedure specifications, *Knowledge Areas* (KAs). A KA consists of an invocation condition, which specifies when the KA is applicable and a body, which describes the steps of the procedure. The metalevel KAs are selected by the interpreter together with the other plans. They are selected, for example, in the case where multiple plans are applicable in the current situation, on failure to achieve a goal, or the awakening of some previously suspended intentions. The reason for having basic decision methods and metalevel KAs is to guarantee a response within a bounded interval. The decisions of the metalevel KAs take longer time and can override the decisions previously taken.

2.4.3 Team modeling

Team modeling in the SWARMM system has been considered mainly in two articles ([29] and [40]). Aircraft are teamed together before the start of the mission and a role is assigned to each member of the team. Team and roles can be reassigned during the mission. Two types of structures are imposed on the team: the organizational structure and the functional structure. The first type defines the command and control part of the team and the second type defines the functional expertise and the responsibilities in performing the task that the team is doing. The roles in the organizational structure are leader and wingman. The leader makes all the decisions for the team and informs the wingman. The wingman obeys the leader and provides information that can help the leader in making decisions. The choice of the functional roles depend on the task the team is performing. The assignment of roles is made during the team formation and depends on the capabilities of the sub-teams.

The team should work as a coherent entity. This can be achieved through centralized or distributed control and coordination. In the first case the leader makes all the decisions and gives orders. In the second case all the members should have an understanding of what the team intends to achieve, the roles in the team and how the team works together and cooperates in observing the world (implicit coordination) or through communication (explicit coordination). In both cases the team should rely on team tactics. Team tactics are modeled as plans in dMARS. Each agent executes the part of the plan proper to its role.

2.4.4 Mental-state recognition

In the dMARS system one of the main characteristic of the agents is the fact that they have mental states, that is beliefs, goals, plans and intentions. Recognizing mental-states in the air-combat domain has been considered in [28].

If an agent has the desire to recognize whether a certain state is being reached, for example it has the aim of recognizing the interception of an opponent, then it intends to recognize all the plans that can achieve this state of the world. This will result in the desire to recognize the first arc in the body of the plan and as a consequence the agent adopts further intentions towards all plans that recognize the desire. The case in which the recognition fails or is revised is not considered; the intentions maintain a suspended state until a successful observation takes place.

Of course, this method depends on a detailed knowledge of the plans that the opponent can perform. Actually the plans that can be recognized are the same plans that the agent itself has the possibility to perform.

2.4.5 Comments

In this section we have presented the SWARMM system, a system for air combat developed at the AAII institute. The SWARMM system is not a rule-based system and therefore differs quite radically from LIBRA. SWARMM is based on the selection of prespecified plans on the basis of the beliefs and goals of the agent. The selection can be made using a basic decision mechanism and through the execution of metalevel plans. This allows the system to react quickly and to then revise its decisions. It is not clear however how the two decision-mechanisms interact especially in consideration of the fact that while the metalevel part is deliberating, the actions selected by the reactive part are executed.

2.5 Multi-agent systems using implicit coordination

In this section we present three recently developed approaches to implicit coordination, that is coordination in the case in which the use of communication is limited. The techniques considered are: a physics-oriented approach to coordination [32], a focal points technique [15], and observation-based coordination [18].

Implicit coordination is very important in both the air-combat and the soccer domain and this is the reason of our interest in these works especially in consideration of the novelty of the techniques used. However, the examples presented are toy examples and it is difficult to judge how far the approaches can be applied to more complex domains.

2.5.1 The physics-oriented approach to coordination

In this approach methods from classical mechanics are applied to model large-scale agent-systems and physics-oriented methods are used to construct a cooperative goal-satisfaction algorithm to be used by the single agent within the system. The example used is a scenario where the agents should block holes of various sizes. Some holes cannot be blocked by a single agent and therefore cooperation is necessary.

Each agent is modeled as a dynamic physics particle and a goal is modeled by a static particle. Goal-satisfaction is modeled by the collision of dynamic particles with static particles. The advantage of this method is in its low complexity and in the fact that the model can rely on theoretical and experimental results from physics, but it seems difficult to apply in more complex cases involving different and conflicting goals.

2.5.2 The focal points technique

The focal points technique is a coordination technique common to communication free human interactions. The problem considered is how two agents can choose the same object out of a set of objects without communication. The idea behind the technique is that if two persons should choose an object where the only objective is to match each other's choice, they will tend to choose an object with special properties of rarity and extremity. An algorithm is presented that chooses an object following a formula that calculates the rarity and extremness of the properties of the object.

The algorithm is of quite low complexity and is independent of the domain and the agents' internal representation, but it relies on the fact that the agents do not have any personal preference about the object to pick. This can creates difficulties if the algorithm is used in choosing common plans in coordinated activities, one of the domains in which the paper claims the technique can be used.

2.5.3 Observation-based coordination

In this approach, agents coordinate performing plan and goal recognition. The agents observe the movement of other agents, infer their plans and goals and act in consequence.

The experimental environment used is very simple: it consists of a discrete time, two-dimensional, simulated grid world with two kinds of agents – bounding agents and overwatching agents. The bounding agents are given the goal of occupying a position and they move to this position. The overwatching agents try to infer the goal of the observed agent and move to a position related to the supposed position of the observed agent.

The goals are inferred employing a belief network. At each step the agent observes the actions of a bounding agent and adds this evidence to the history of previously observed actions and updates the beliefs in the Bayesian network. The belief in a possible destination increases in the presence of reinforcing evidence and decreases in the presence of contradictory evidence.

The interesting aspect in this approach is the possibility to gradually change beliefs depending on observations of the world, but given the simple example used it is difficult to assess how the method scales up to more complex situations especially if the agents change their goals during the execution and if both agents are watching each other while they are pursuing their own goals.

2.6 Discussion

In this chapter we have presented three quite different systems. Brawler has been developed mainly in order to test air-combat "products", such as aircraft, weapons, tactics and so on, while TacAir-Soar and SWARMM have their main application in developing automated pilots for simulated tactical exercises.

Brawler seems to offer a very interesting approach to decision making that can allow a great flexibility in the pilot decision process. The level of detail of the model is quite amazing; consider for example the high level of detail with which the visual process of the pilot is modeled. The modeling of the acquisition of information in the pilot mental model is also interesting, an aspect this that is not taken so much in consideration by the other two systems.

TacAir-Soar is based on the Soar system, a very general approach to modeling human behavior. It has, therefore, the advantages of having most of the capabilities required for intelligent agents already implemented in the system, but the drawback of not having an architecture created for the specific domain. TacAir-Soar has also more systematicaly addressed capabilities such as coordination, communication and agent tracking.

The SWARMM system focuses on the mental state of the agents, their beliefs, goals, and intentions. An interesting point is that several intentions can be present at the same time, but as intentions are modeled as plans, it is not so clear how interruption or suspension of plans is handled when a new intention becomes active.

We believe that what mainly differentiates LIBRA from the systems just presented is closer attention to the user aspects of the system. In LIBRA the user of the system writes the complete specification of the behaviors of the agents, not the developers of the system. This is an advantage considering that the users have a practical knowledge of the domain and can adapt and change the tactics and the scenarios in the simulation depending of their needs. On the other hand this leads to constraints due to the fact that the system should be easy to use. Therefore, an important part in the future development of LIBRA will be a systematic testing and evaluation of the user aspects of the system.

Another feature that is not present in these systems and that we intend to consider in the future is planning and high-level handling of interruptions and exceptions.

In the last section we have presented three multi-agent systems that deal with implicit coordination. In all the three systems the ideas are interesting and innovative, but due to the simplicity of the examples it is not easy to see how the systems can scale up to more complex domains.

Chapter 3

The TACSI system

3.1 Introduction

The TACSI (TACtical SImulation) system has been designed by Saab Military Aircraft AB for simulating aircraft combat and in particular beyond-visual-range combat. A combat scenario is represented in terms of rules. The rules are defined by the user with the aid of a rule editor. The user is able to create his own scenarios and specify the behaviors of the agents present in the scenario in order to evaluate aircraft, missiles and tactics. The system has been created mainly for evaluating aircraft, missiles and tactics, but can also be used for training pilots. Typical users are retired pilots with elementary knowledge about computers. It consists of a physical simulator, a user interface for entering rules, scenarios and characteristics of aircraft and missiles and a decision-making mechanism.

The physical simulator includes models of aircraft and missiles and shows the aircraft moving and fighting in a virtual environment. The user can modify the aircraft and missiles characteristics using a *type editor* and can construct scenarios deciding which kind of aircraft he wants, the number of missiles, the quantity of fuel and so on.

The user can enter the rules according to which the agents, the simulated air combat pilots, should behave. The rules consist of several conditions and one or more actions that should be performed if the conditions are true. There is a fixed priority among the rules. The decision-making mechanism checks the rules in order of priority and fires the first rule whose conditions are true. The priority is defined by the order of the rules in the rule list. The simulation can be registered and the combat can be evaluated checking the rules that have fired.

The description of TACSI given in this section is based on the master thesis of J. Andersson [2], and also on discussions with staff at Saab and observations of the system itself.

3.2 Tasks, states and blocks

In TACSI there are three general tasks[1] which an aircraft can perform; these are: intercept, escort and attack.

Each task consists of a number of *states* that represent different phases of the task. For example in the escort task, the aircraft can be in the *escort state*, but can also be in the *attack state* in case an enemy aircraft becomes a threat, or in the *out state* when it is coming back after the mission. For each task there is a "natural" order between the states, that is the order in which the states are entered in a standard mission. The first state in this order is the starting state. The states of the task for escort are shown in figure 3.1.

For each of the aircraft involved in the simulation and for each of the tasks the aircraft can perform, the user defines the behavior of the automated pilot in terms of production rules. The user defines the rules for each state in each task. He is aided in creating the rules by the fact that each state is divided into *blocks*. In a block the user creates rules concerning one resource. There is one block for each resource that the user can make decisions about; for example the user can create a rule in the guidance block of the attack state that says that if a missile has been fired towards the aircraft, the aircraft should try to escape from it. The blocks execute in parallel and thus rules that concern different resources can fire in parallel. Examples of blocks are given in figure 3.1.

The rules do not deal with details of the actions; for example the conclusion of a rule can just be "escape from missile" and if the rule is fired the physical simulator takes care of changing the position of the aircraft trying to keep a predefined angle between the missile and the aircraft.

In every state there is the *Delta rule* block, that contains rules to decide which state to go to in the next step.

A special state is the *Beta block* state. This is present in every task and takes care, for each aircraft, of identifying the other aircraft and deciding which enemy aircraft should be the main target.

[1]Missions in TACSI terminology.

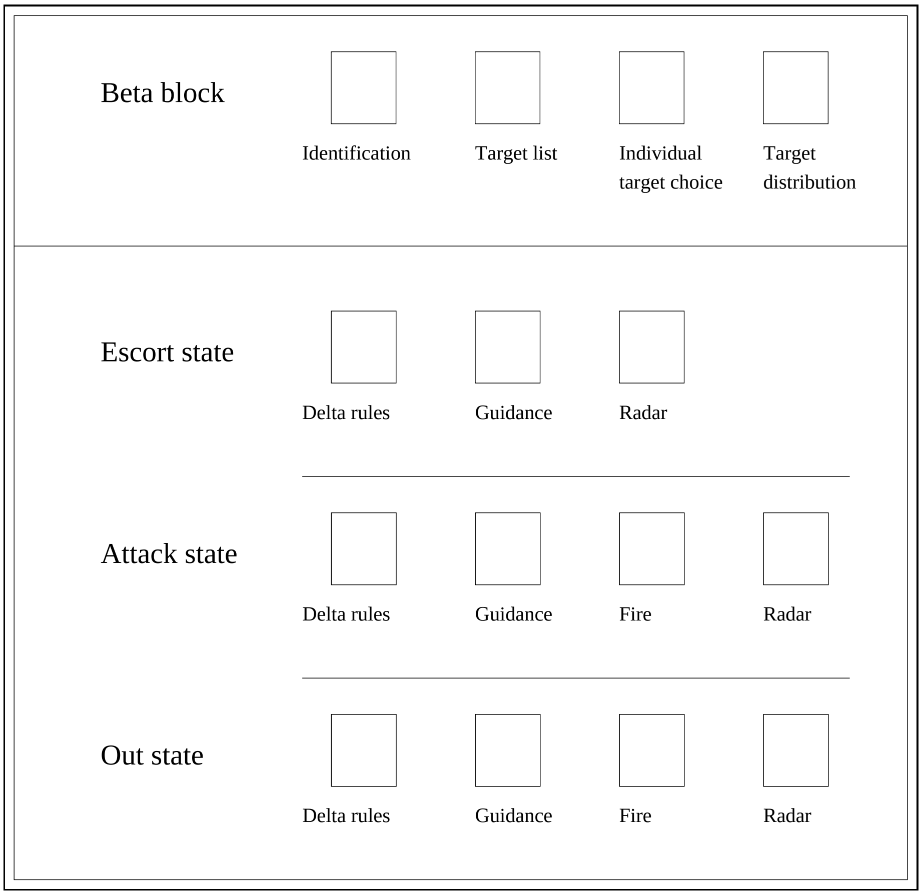

Figure 3.1: The states in the escort task

The decision-making process in TACSI can be seen as two finite state machines working in parallel. One state machine consists only of the Beta block state. The second state machine contains the other states of the task and has the Delta rules as transition rules. The system is therefore at all times simultaneously in two states: the Beta state and one of the other states present in the task.

3.3 The rule definition

The user defines rules through a rule editor, figure 3.2, marking the preconditions and the conclusions for each rule. For example, rule number 3 in figure 3.2 has as precondition that a target is selected, *Tgt1 selected*, and as conclusion *Go to state attack*. The user can also mark that a precondition must be false in order for the conclusion to execute. The order in which the rules appear in the editor decides the priority order among the rules. For example in figure 3.2 if the preconditions of the first rule is satisfied the rule is fired without testing whether the preconditions of the second rule are also true.

3.4 The decision-making process

At each time point the system tests first the rules in the Beta block state in order to assign a primary target to each aircraft. Then it tests the rules in the Delta rule block in the current state to decide whether to continue to stay in the same state or change state. Finally the system tests the rules in each block and when it has found a rule to fire or has examined all the rules in one block, it considers the next block. If no rules apply, the aircraft continues flying according to the guidance principle decided at a previous time point. For example if the aircraft was evading a missile and no rule applies, it will continue flying at the altitude, speed and direction most appropriate to evading the missile.

3.5 Future development

Saab is currently working on introducing hierarchical states in TACSI. A state hierarchy is assigned for each of the agents before the starting of the mission [2]. The state hierarchy depends on the nature of the mission and on the role that the agent has in the mission. It has the form of a tree

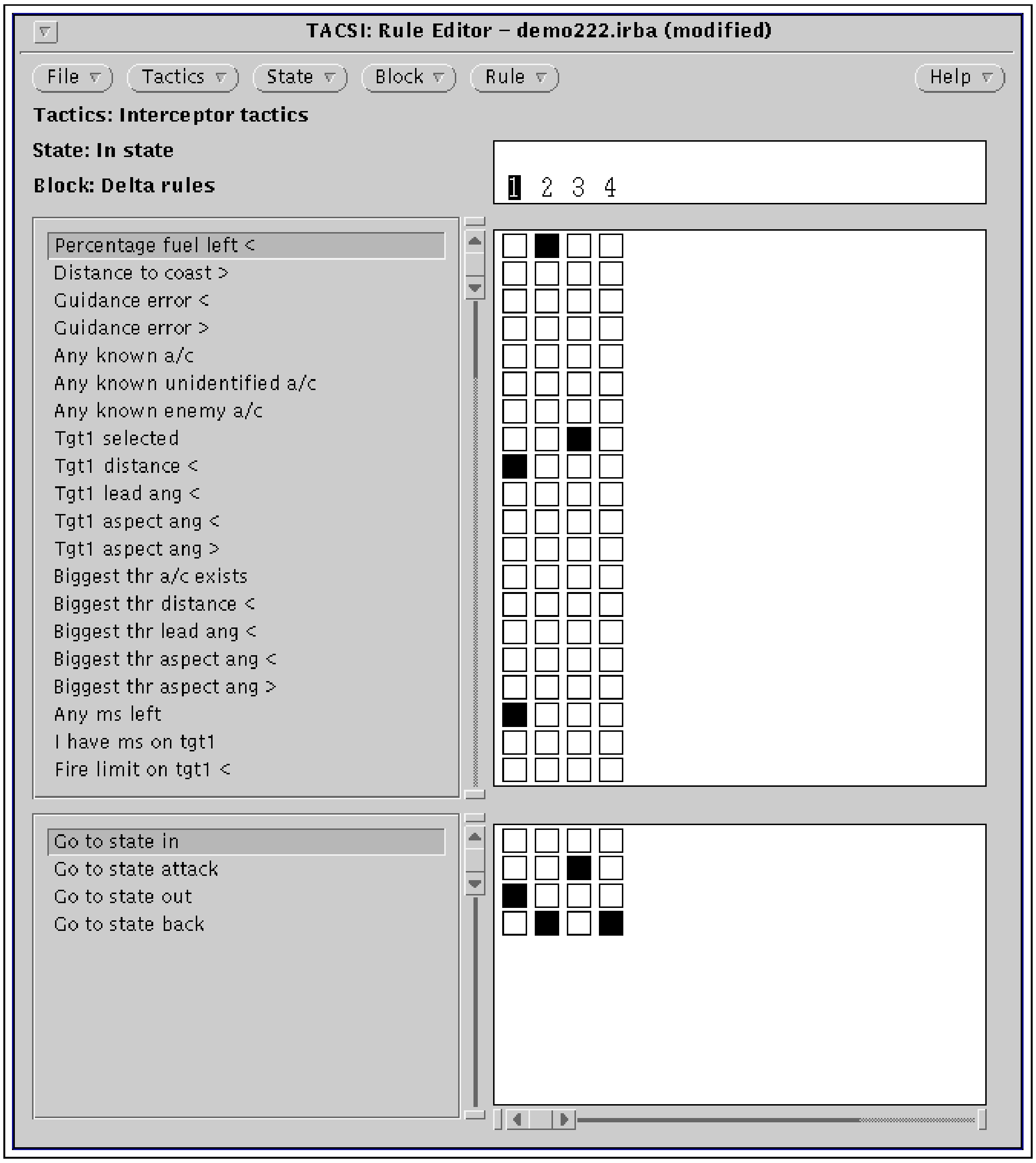

Figure 3.2: An example of the rule editor in TACSI and four rules

with states as nodes. The states on the leaves can contain several blocks, for example guidance, firing, radar, while the other states contain just the Delta rules block. An advantage of the state hierarchy is that the number of rules in each state is smaller. This facilitates the creation of robust tactics as there can be a branch for each specific situation.

A master thesis written in cooperation between Saab and Linköping University has presented the problem of how to introduce a military hierarchy between agents, how to send orders and what happens if one of the agents changes position in the hierarchy during the mission [17].

3.6 Comments

TACSI has a simple decision making mechanism: it just checks the rules in order of priority. This allows the system to make decisions fairly quickly, but on the other hand the tactics become less robust and experience shows that a slight change in the simulated scenario can make the agent react in a way that a human observer would consider inappropriate. One way to improve robustness is to create rules that apply not just to a specific scenario, but that codify principles applicable to a variety of scenarios. This can in general lead to a larger number of rules and more complex conditions in the rules as more aspects have to be taken in consideration. The increase in the complexity and number of the rules can make it difficult for the user of the TACSI system to put the rules in the right order of priority and formulate the conditions. This is especially the case as the rules are checked in a fixed order and as soon as the conditions of one rule are satisfied that rule fires and no other rules are considered. Therefore, if the user wants an action A to be preferred to an action B given certain conditions, but B is preferred to A if these conditions are not satisfied, then he should write instead of just two rules several rules that consider the combination of the cases.

Several aspects of LIBRA address these problems. One is a dynamic prioritization of the rules. If a rule fires, the action in the rule is not executed directly, but it is just considered as a candidate to be performed. If more than one rule fires, the actions to be executed are chosen on the basis of numerical priority values and compatibility. These priority values can be changed dynamically given the satisfaction of conditions, and thus it is straightforward to prioritize rules one over the other depending on satisfaction of conditions.

A second feature is the hierarchical organization of the conditions in the rules. A rule is a branch of a tree and each of the nodes in the branch is entered if the conditions in the arc are true. In the leaves there are actions. If the leaf of the branch is reached, the action become a candidate to be performed. With this organization of the rules, rules that have partly the same conditions can share part of the branch. This both helps the user in organizing and writing the rules and increases efficiency as the conditions are tested just one time. Furthermore, it is not necessary to consider all the conditions of a rule to decide that the rule is not applicable: if the conditions for entering a node in the branch representing a rule are not satisfied, then the node is not entered and no other conditions of that rule are tested.

Another interesting feature of LIBRA is the possibility of defining internal actions, that is actions that are not sent to the simulator for execution, but that update the state of the agent. For example, given certain conditions an agent can decide to mark a particular opponent. This decision is recorded in the state through an internal action and it will influence the future actions of the agent. The term "state" is used differently in LIBRA and TACSI. In fact a state in TACSI is a state in a state machine. For example, an agent can be in the escort state or in the attack state and the Delta rules decide in which state the agent is in. In LIBRA each agent has a state which records the information that the agent uses for making decisions. The passing from one state to another in TACSI can be expressed in LIBRA by the updating, through internal actions, of a variable in the state of the agent.

Finally, we have defined a language to specify conditions that allow the user to write more complex conditions with operators such as *not, and, or, for-all*, and *there-is*. This increases the expressiveness of the conditions and contributes to reducing the number of rules and increasing the compactness of rules. For example, in TACSI there is no *or*; therefore a rule that in LIBRA would have an *or* in the conditions is expressed in TACSI by two rules, each with one of the disjuncts in the conditions. If the rule in LIBRA has in its conditions nested *and* and *or* clauses, the number of rules needed to express the same rule in TACSI increases even more.

Chapter 4

Description of LIBRA

4.1 Introduction

In this chapter we present the general architecture of our system. In the next two chapters we then consider how the system has been adapted to the two applications we have been dealing with.

Each agent in our system, shown in Fig. 4.1, has a state which records the information needed by the agent and a decision-tree where the behavior of the agent is specified. The agent interacts with a simulator by sending actions and receiving information. The actions are selected by visiting a decision-tree. A decision-tree is a hierarchical organization of rules, with conditions on the arcs and actions on the leaves. Each node of the tree is entered if the conditions on the arcs are satisfied. The satisfaction of conditions is checked by examining the state of the agent. The actions reached in the visit of the tree are actions that are candidates to be performed. The actions that will actually be executed are the ones among the candidate actions that have higher priority and are compatible. The actions selected can be external actions and in this case are sent to the simulator or internal actions and in this case update the state. The information received from the simulator is processed before storing it in the state. The processing-information module derives additional information to that received by the simulator and stores the information in a form that is meaningful for the checking of conditions.

The characteristics of the agents, as for example the role that the agent has and the kind of aircraft, can be specified before starting the simulation. The user also defines the decision-trees for each of the agents.

In sections 4.2-4.4 we examine specification of agents, state, decision-tree and actions.

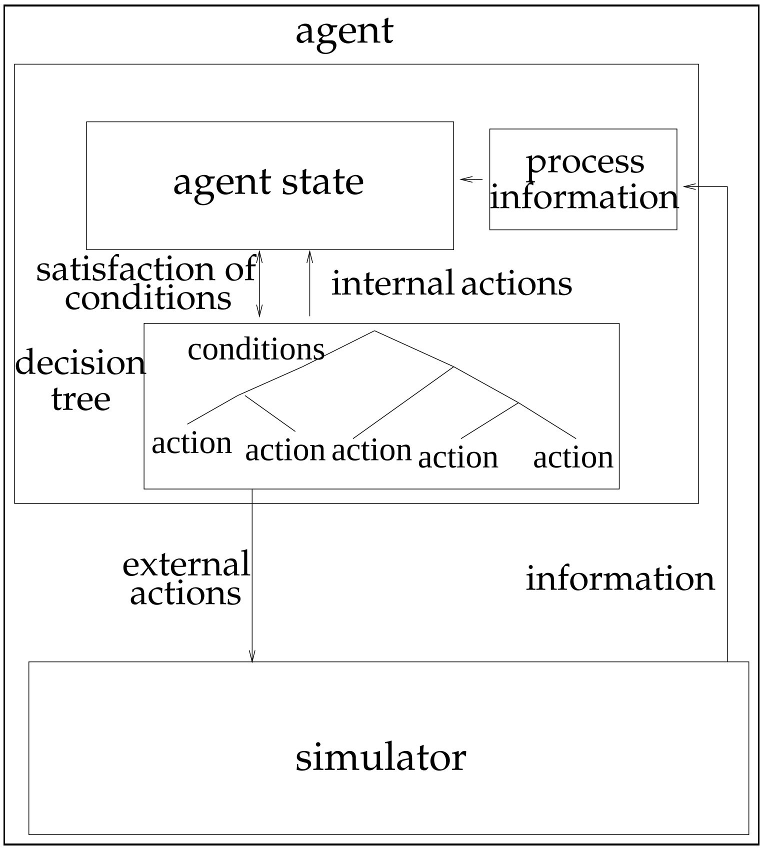

Figure 4.1: Structure of an agent in LIBRA

In section 4.5 we examine how the requirement that the user defines the decision-tree has influenced our choices in building the system. Finally in section 4.6 we explain what we mean by reactivity in LIBRA and in section 4.7 we consider coordination between agents.

4.2 Specification of the agent

The specification of the agent contains the characteristics proper to the agent. These characteristics are specified by the user before starting the simulation and are maintained in the state of the agent. Examples of characteristics in the case of an automated pilot are:

- Physical characteristics, such as type of aircraft, number and type of missiles and amount of fuel.
- The role of the agent in the simulation, for example leader or wingman.
- A description of the mission that the agent should perform.
- General criteria the agent should follow when performing actions, for example commit criteria, i.e., criteria for deciding whether to intercept an enemy aircraft, and rules of engagement, i.e., directives about general conduct during the mission.

4.3 State

Each of the agents has a state where the information that is necessary for deciding what actions to perform is maintained. The state is structured in the present implementation as pairs of attribute-values. The information is stored with a time stamp to make it possible to recognize whether the information is up-to-date.

The information present in the state is of four types. First, the state contains the characteristics of the agent specified before starting the simulation, for example team and position (midfielder, left-attacker, etc.) of the soccer player or side and role (leader, wingman, etc.) of the pilot.

Secondly there is the information the agent receives from the simulator about objects and other agents present in the simulation. This information is partly information directly received from the simulator (for example distance and direction of other players, flags, goals and lines) and partly information that is derived during the processing of the messages received. Examples of this last kind of information include which opponents or team mates are near the ball. In case of the pilot the information is what a pilot would receive from on-board sensors.

Thirdly, there is the information about the present status of the agent. This is partly information that the agent receives from the simulator about itself and partly information that the agent records and updates through internal actions. In the case of the player the information can be the angle of view and the quality chosen for the visual information, position and the actions currently being performed. In the case of the pilot the direction, velocity and position of the aircraft, the number of missiles left, friendly

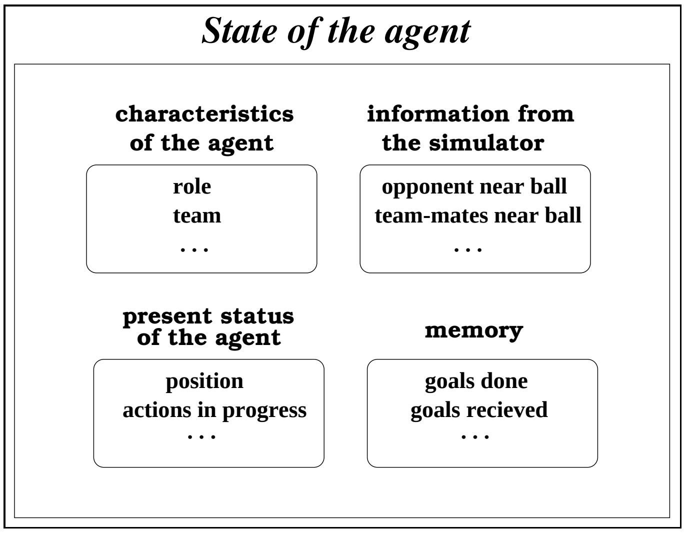

Figure 4.2: Example of information present in the state of a soccer playing agent

aircraft still present and the actions presently performed, for example the agent is following the leader or moving toward a point.

Finally, one part of the state represents the "memory" of the agent. What is recorded in the memory, among other things, are important past events, for example the number of goals scored for and against, decisions previously taken, for example the decision to select an enemy as main target or the decision to intercept an enemy, and messages received via communication channels.

4.4 The decision-mechanism

In our approach, each agent is equipped with a state representing the information that it uses for making decisions and a decision-tree. The decision-tree specifies the behavior of the agent and consists of a hierarchy of decisions with one decision for each node. At every step of the simulation the state is updated with information that is received from the simulator and interpreted. Then the decision-tree is visited which results in a list of actions being created (Fig. 4.3). The actions in the list are the actions that the

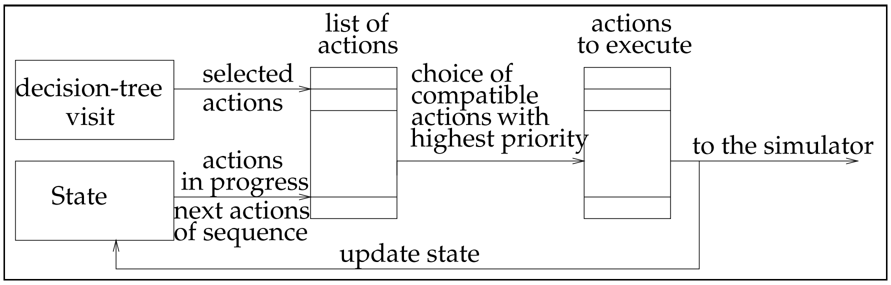

Figure 4.3: Phases of the decision mechanism

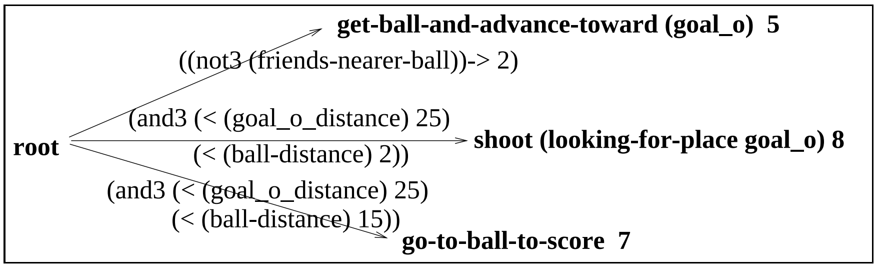

Figure 4.4: Simple example of a decision-tree

agent will consider performing. Each action is associated with a priority value that changes dynamically. To this list the actions and the activities still in progress are added. The agent then selects the actions to actually execute and then send them to the simulator or to update the state. Some of the actions in the list are mutually exclusive as they use the same resource and in this case the agent selects those he will actually perform depending on their priority values. Some actions can be performed concurrently and in this case the agent performs them in parallel. The agent can also decide to perform sequences of actions. However the agent is not committed to complete the sequence of actions as the sequence can be interrupted if an action with higher priority is selected.

Let us consider a small example of a decision tree in the soccer domain (Fig. 4.4). The agent, if there are no team mates nearer to the ball *((not3 (friends-nearer-ball))*, tries to get the ball and advances toward the opponent's goal, **get-ball-and-advance-toward(goal_o)**. When it is near the opponent's goal and it can kick the ball, *(and3 (< (goal_o_distance) 25) (< (ball-distance) 2))*, it tries to score, **shoot(goal_o)**. If it is near the goal but not near enough to the ball to kick, it goes toward the ball **go-to-ball-to-score**. In this example the conditions are not mutually exclusive and the correct action is chosen depending on priority. For example, the **shoot(goal_o)** action has priority 8, while the **go-to-ball-to-score** action has priority 7 so if the player is near the ball it will perform the shooting action.

The conditions on the branches can refer both to information obtained from perception and information stored in the agent's state. An example of the latter is when a defender follows an opponent due to a commitment to mark this opponent. This commitment is registered in the defender's state through an internal action. Making the commitment to mark a particular opponent constitutes an internal action that updates the state of the agent.

The external actions presented in the decision tree will be interpreted in terms of the low-level actions available in the soccer simulator. For instance, **go-to-ball-to-score** will make the agent turn and move towards the ball. Also the information received from the simulator is interpreted. For instance, the value of *friends-nearer-ball* is derived from the incoming message. This permits the user to work on a level of abstraction that is more meaningful and convenient than working with the low-level data and actions of the simulator. He/she can focus more on what to do than the details of how to do it. In the air-combat application, a certain level of abstraction is built into the simulator. In the soccer application, we had to provide this abstraction ourselves.

4.4.1 Decision-tree structure

In a decision-tree a node is entered if the entry condition associated with it is satisfied. In the leaves there are actions with a basic priority value. In addition modifier conditions can be associated with a node. Modifier conditions have the form *(condition* $\rightarrow$ *number)*. If the condition is true, the number is added to the priority of the actions associated with the branches. If the priority of the action falls to 0 or below, the action is not performed.

4.4.2 Conditions

The conditions are defined as follows:

- **Primitive Conditions**

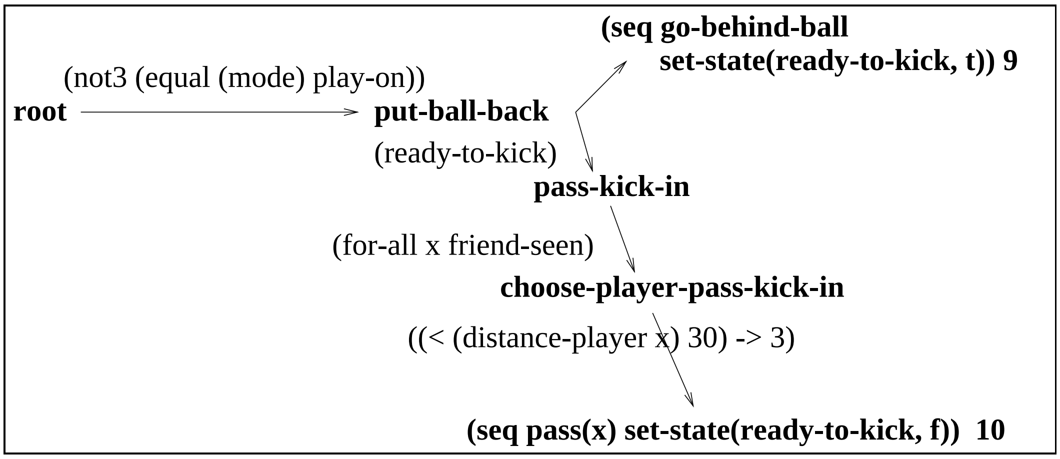

Figure 4.5: part of the decision-tree used to put the ball back in play

- Atomic Conditions

  Atomic conditions are true if the information is present in the state. For example, the atomic condition *friends-nearer-ball* is true if there is information that there are team mates nearer to the ball.

- Relational Conditions

  Relational conditions consist of relational operators such as $<, >, \leq$, $\geq, =, \neq$, applied to numbers and numerical functions. An example of a relational condition is *(< (distance-player x) 30)* where *(distance-player x)* is a function that returns the distance of the agent to another player x.

- **Composite Conditions**

  Composite conditions are formed by applying the operators *and3, or3, not3, for-all* and *there-is* to other composite or primitive conditions. *And3, or3, not3* are operators in a three-valued logic: true, false and unknown. In fact the agent can be in the situation in which he does not know the value of a part of a condition, for example he does not know his relative position with respect to the ball, but he can still decide the true value of the condition, for example using the rule that *and3* of false and unknown is false.

If the condition associated with an arc has the form "*(for-all x set condition)*", the condition is tested for each agent x in the set and x can also be used in the rest of the branch. The rest of the branch is visited once for each of the agents for which the condition is satisfied. All the actions selected are then taken into account as candidate actions to perform. "*(there-is x set condition)*" finds the first agent in the set that satisfies the condition and continues to evaluate the tree with x as this agent.

4.4.3 Actions

There are three kinds of actions: primitive actions, concurrent actions and sequential actions. Primitive actions are, for example, **pass(x)**, i.e., pass the ball to the agent x. They can also be internal actions, i.e., actions that update the state of the agent. For example **set-state(mark, x)** records in the state the fact that the agent is committed to mark the player x. The actions sent to the simulator can be instantaneous, for example **pass(x)**, or can have a duration. An example of an action with duration is the action **go-behind-ball** shown in Fig. 4.5. The part of the tree in this example is accessed when the game is suspended and the player should put the ball back in game. The player first goes behind the ball and then passes the ball to a fellow player. The action of going behind the ball requires a certain amount of time and it is performed in different phases. The agent keeps track of the actions that he is currently performing in the state and in which phase the action is. At each step of the simulation the agent decides whether to continue performing the action or interrupt it in order to perform a more important action (that is one with a higher priority value).

Sequential actions are composed of actions that should be performed one after another. The **seq** operator is used to define sequential actions. When a sequential action is started, the first action of the sequence is started and it is automatically recorded in the state the following actions in the sequence that should be performed. When the first action of the sequence is completed, the second action of the sequence becomes a candidate action to be started and it competes with the actions currently being performed and the other actions being selected at the present cycle of the simulation. If the second action is started, the same is then done with the third action and so on until the sequence is completed. If an action that is in its turn is not started, the sequence is aborted. An example of sequential action is **(seq go-behind-ball set-state(ready-to-kick, t))** shown in Fig. 4.5. When the action of going behind the ball is completed, the agent records in the state that he is ready to kick the ball.

When a sequential action is started, the priority of the following actions in the sequence is increased by a value decided by the user and depending on how important it is in his opinion to complete the sequence.

Concurrent actions consist of a collection of actions that should be started at the same time. The **con** operator is used to define concurrent actions. There is a concurrency of actions inherent in the system as actions in different branches can be selected and, if they are compatible, started at the same time. With explicit concurrent actions, however, we are sure that either all the actions are performed or none of them is. The actions can be a composition of both concurrent and sequential actions.

4.4.4 Compatibility of actions

The test to check whether two actions are compatible is based on the physical resources they use. Two primitive actions are compatible if they do not use the same resources. For example the action of firing is compatible with the action of turning left as they use different resources.

A concurrent action is compatible with another action if all the actions that form the concurrent action are compatible with the action. A sequential action is compatible with another action if all the actions that are present in the sequence are compatible with the action.

Compatibility of actions is more significant in the air combat domain than in the soccer domain, as the pilots can in general perform more actions in parallel than the soccer players. In the soccer domain the actions that are compatible with each other are the actions of sending a message and the actions that involve movement. The internal actions are in general compatible with all the other actions and among themselves.

4.5 Requirements for user-defined decision-tree

The knowledge required by an agent to perform realistically in complex and uncertain situations is in general difficult to acquire and to code precisely by the persons who implement the agents. Furthermore, the user can require different behaviors of the agents for different purposes, for example a pilot's trainer could require a very simple behavior of the agent in order to train an inexperienced pilot. It seems, then, reasonable to give the user the possibility to decide the behavior of the agents and to adapt it depending on the particular use of the system. The challenge is to design a mechanism for specifying the behavior of the agent that is powerful enough for the user to specify the behavior, but at the same time is easy to learn and to use.

4.5.1 Hierarchical structure and language of conditions

Specifying the decision mechanism in the form of rules is a natural way to represent behavior. However, the number of rules and conditions necessary for specifying complex behaviors can be very large and their structure complex. TacAirSoar, for example, contains about 1700 rules. Also several conditions can be common to more than one rule. Therefore organizing the rules in a tree can contribute to a more structured and compact representation and this can facilitate the writing of the rules and the understanding of the resulting behavior. In our solution each node in the tree is associated with a condition that has to be satisfied in order to enter the node, and each "end node", or leaf, is in addition associated with an action. A branch of the tree represents a rule that has as conditions the conjunction of all the conditions in the arcs of the branch and as action the action in the leaf. In this way the conditions are structured from more general to more specific and the conditions can be common to several branches yielding a more compact representation and more efficiency in testing the rules as the conditions are tested just one time. The efficiency is also enhanced as it is not necessary to consider all the conditions of a rule to decide that the rule is not applicable: if the conditions for entering a node in the branch representing a rule are not satisfied then the node is not entered and no other conditions of that rule are tested.

It is also important to give the user the possibility to write complex conditions. For example a condition for performing the action of returning to base can require that the agent has completed the mission or it has too little fuel to continue with the mission or it has no more missiles. A language has therefore been defined for writing conditions that allow the use of operators and functions (section 4.4.2).

4.5.2 Priorities and changes of priorities for actions

Several branches of the tree can be visited in parallel. If a leaf is reached, the corresponding action becomes a candidate to be performed. All the candidate actions to be performed form the candidate action set. From this set, the actions that are actually performed are selected on the basis

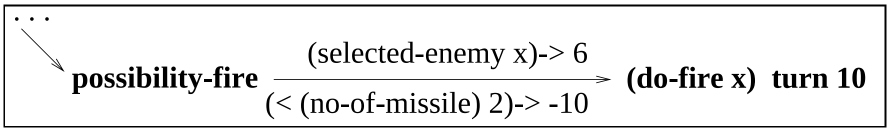

Figure 4.6: Example of dynamic change of priorities

of dynamically varying priorities. The user specifies a priority value for the action in each leaf and also specifies how this value can be changed during the evaluation of the branch conditions in which the action is a leaf. We have introduced the possibility of dynamic change of priorities in order to simplify the tree and to simulate the change in priority, in the pilot's mind, of the actions that he should perform depending on the situation.

The following example illustrates how the dynamic change of priorities is used. In figure 4.6 a part of a decision-tree is presented. The agent has the possibility to fire at an opposing agent **x** and should decide whether to fire or not. If **x** is the selected enemy, the agent should fire. If **x** is not the selected enemy, the agent should fire only if 2 or more missiles are left. In the example, if **x** is the selected enemy, the priority of the sequential action **(do-fire x) turn** is increased by 6, and if the number of missiles is less than 2, the priority is decreased by 10. Given that the priority of the action is 10 and that the action is not performed if its priority is equal to or less than 0, the behavior is the desired one. In order to implement the same behavior without dynamic change of priorities four action nodes would be necessary (figure 4.7).

The actions at the leaves of the decision-tree can also be concurrent and sequential actions. Sequences of actions are useful if the user wants an action to be followed by other actions, for example firing and turning in the previous example. Concurrent actions are used when a number of actions should be started at the same time, for example turning and releasing a decoy in case a missile warning is received.

4.6 Reactivity

When we say that a property of LIBRA is reactivity, we do not mean that an agent in LIBRA purely reacts to the current state of the world. What we mean is that an agent constantly checks the state of the world and can

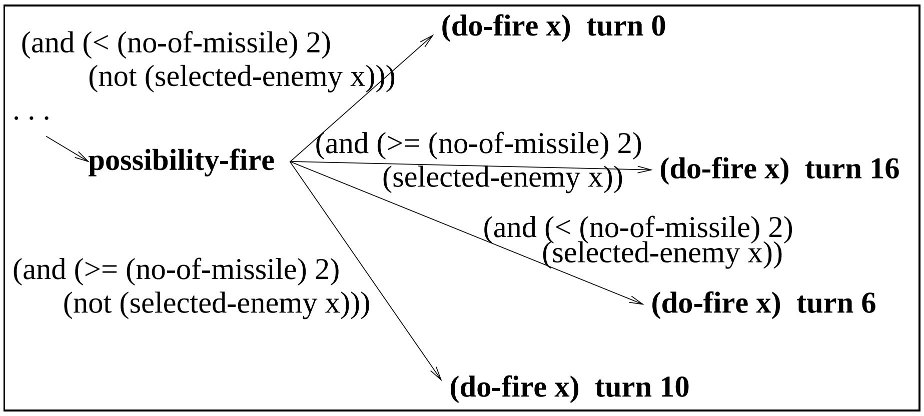

Figure 4.7: The same example as in figure 4.6 without dynamic change of priorities

quickly react and adapt to changes in the world. In fact the decision-tree of each of the agents is visited at each step of the simulation and the decisions previously taken are reevaluated given the current state of the world. The agents in LIBRA are not purely reactive as their decisions depend not only on the information received by the simulator, but also on the information present in the state, for example past events, decision previously taken and intentions.

4.7 Coordination among agents

In both the soccer and the air combat domain teamwork is essential, but communication between team mates due to real time constrains and the possibility of interception by the opponents should be reduced to short and simple messages. Coordination is therefore obtained by common tactics and strategies learned and practised during training. Each agent knows the tactics of the team and its role in the tactics and tries to act according to this knowledge. The agent should initiate tactics depending on the situation and recognize the tactics that the other team members have initiated in order to play its part in it. The agent has three ways of recognizing the tactics that the team is following: observation of the general situation and knowledge of the tactics appropriate for the situation, observation of indicative actions of the team mates, and explicit communication. In general there can be a combination of these three capabilities.

In our approach the knowledge required for applying the tactics is codified in the decision-tree and the observations of the situation and of the actions of other agents influence the decisions about which tactics to adopt. Explicit communication is performed through communication actions present in the decision tree. The message is stored in the state of the agent and will influence future decisions. Tactical decisions made by the agent are also registered in the state. The agent can change tactics as the situation changes. Further, the use of priorities helps in taking advantage of opportunities or reacting to threats that are not considered in the tactics selected.

In section 6.4 an example of coordinated behavior between a left attacker and a midfielder and how we codify it in the decision-tree is presented.

4.8 Summary

We have described a decision mechanism for autonomous agents which attempts to provide a relatively simple and easy-to-learn way to specify the behavior of autonomous agents. It is based on a decision-tree where actions are assigned dynamically changing priorities and decision are based both on sensory input and the internal state of the agent.

Coordinated behaviors are encoded in the decision trees of the individual agents. What links these individual behavior descriptions together are explicit communication and common means to describe the situation the agents are in, both in terms of their observations and their internal states.

Chapter 5

The application of LIBRA to the air-combat domain

5.1 Introduction

As the air-combat domain is the original application of LIBRA, the system has been tested since its early stages of development by interfacing it with an air-combat simulator. Initially the simulator that we have been using is a simulator, WSIM++, developed at Linköping University by M. Morin and L. Karlsson [25] and updated by P. Nordqvist and L. Törner.

The results of this interfacing were positive and have been summarized in [26]. However, it was important for the testing of tactics and strategies in the air-combat domain that aircraft, weapon and radar systems were simulated realistically and this was not possible in the WSIM++ simulator. To test the system more accurately, we have interfaced it with the simulator used in TACSI where all the physical aspects of the air-combat mission are accurately modeled. The system resulting from the interface between TACSI and LIBRA is TACSI-LIBRA.

In the next section we provide details of the interface between TACSI and LIBRA and then we present a scenario that we have tested in the system obtained after the interfacing.

5.2 The interface between TACSI and LIBRA

Most of the TACSI system has been maintained in TACSI-LIBRA. The conditions and actions are common in the two systems; what has been changed is the decision-mechanism used in selecting the actions to perform at each step of the simulation. The actions performed are selected for each agent by visiting its decision-tree instead of the finite state machine in TACSI.

The conditions and the actions in TACSI are on a high level of abstraction such as "escape from missile". Using these conditions and actions has been an advantage as it has allowed us to easily specify complex and realistic tactics. On the other hand conditions and actions have been developed in relation with the use of TACSI, so they offered little support for specified tactics not considered in the original TACSI, for example complex coordinated tactics that involved observing the manoeuvers of team mates. We have, therefore, tested in TACSI-LIBRA tactics that did not involve complex coordination and we have considered coordination mostly in the soccer domain. We expect eventually to be able to add new conditions and actions in TACSI and apply the coordination techniques used for the soccer domain in the air-combat domain too.

5.3 Example

In this section we consider an example of a scenario that has been tested in the system. In the scenario six agents are present, two on the blue side (friends) and four on the red side (foes). In figure 5.1 the aircraft on the blue side are the ones with numbers 1 and 2.

The red force consists of two attack aircraft and two escort aircraft. The purpose of the escort is to protect the attack aircraft which are carrying bombs. The escort aircraft are flying at a high altitude while the attack aircraft are flying at a low altitude. The attack aircraft do not use radar since radar can easily be detected. The escort aircraft provide information about the current situation to the attackers through link information.

On the blue side, the interceptor aircraft are supposed to protect the area from the enemy. They will try to manoeuvre around the escort aircraft, to reach the attack aircraft and attack them as they are the most interesting target. The intercept aircraft can either try to shoot down the escort and then get to the attackers, or try to get around the escorts by different manoeuvres. In figure 5.2 is shown the scenario after a number of iterations.

There are three decision trees, one for the red attackers, figure 5.3, one for the red escorters, figure 5.5, and one for the blue intercepters, figure 5.4. The aircraft in the section behave differently even if they have the same decision-tree as they are in different positions and have different roles, leader

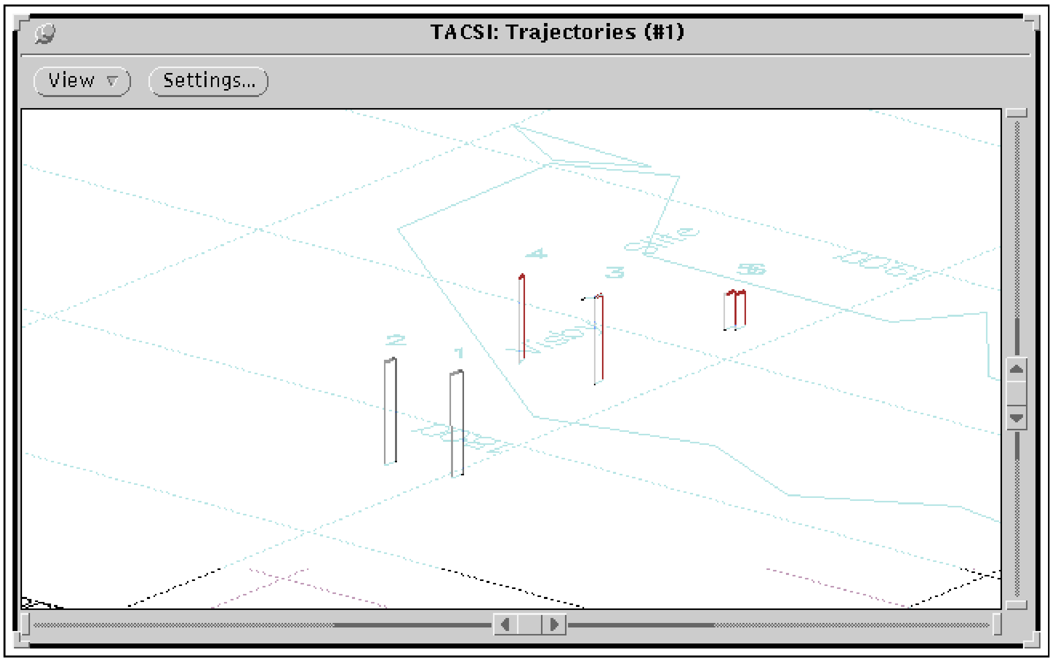

Figure 5.1: The scenario at the beginning of the execution. The vertical lines indicate the altitude of the aircraft and the horizontal line the trajectories

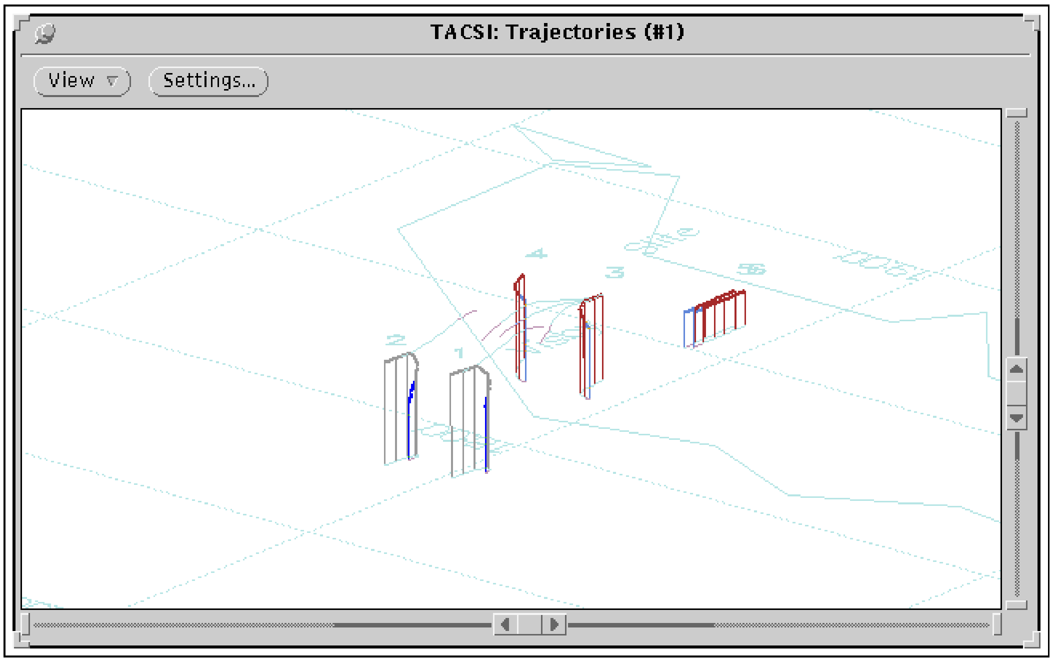

Figure 5.2: The scenario after some iterations. The vertical lines indicate the altitude of the aircraft and the horizontal line the trajectories

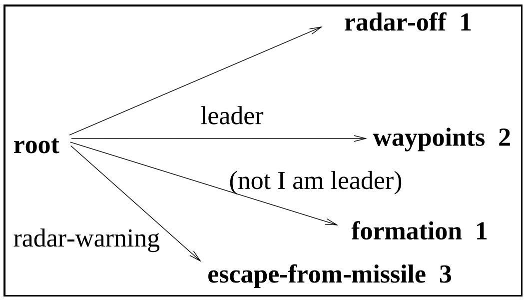

Figure 5.3: The red attacker decision-tree

and wingman.

Let us consider the three decision-trees. We do not go into the details of the explanation of the trees as the conditions and the actions employed are rather technical and domain-specific.

The red attacker decision-tree is the simplest one. The red attacker that is leader of the section goes to the attack point (**waypoints**) and the wingman follows the leader keeping in formation (**formation**). In the case that the radar warns of a missile heading toward them, they escape from the missile.

The blue interceptors attack if they have a target and enough fuel (*and tgt1 selected, percentage fuel left* $>$ *15.0*). In the case of an attack they move along a path that allows them to come near to the opponent, but without entering its firing area (**gimbal-limit**). When the conditions for firing are favorable they fire an active missile (**fire-active-ms**).

The tree of the red escort is the more complex. When the attack planes that they are protecting are still present (*own attack a/c exists*) they escort them. If they detect an enemy aircraft, they attack. The guidance principle used in the attack (**lead-pursuit** and **constant-lead** with different parameters) depends on technical conditions related to the target. When the conditions are favorable, they fire.

5.4 Communication between agents

In the TACSI system exchange of messages between agents through sending and receiving actions is not present. It has, therefore, been of interest for

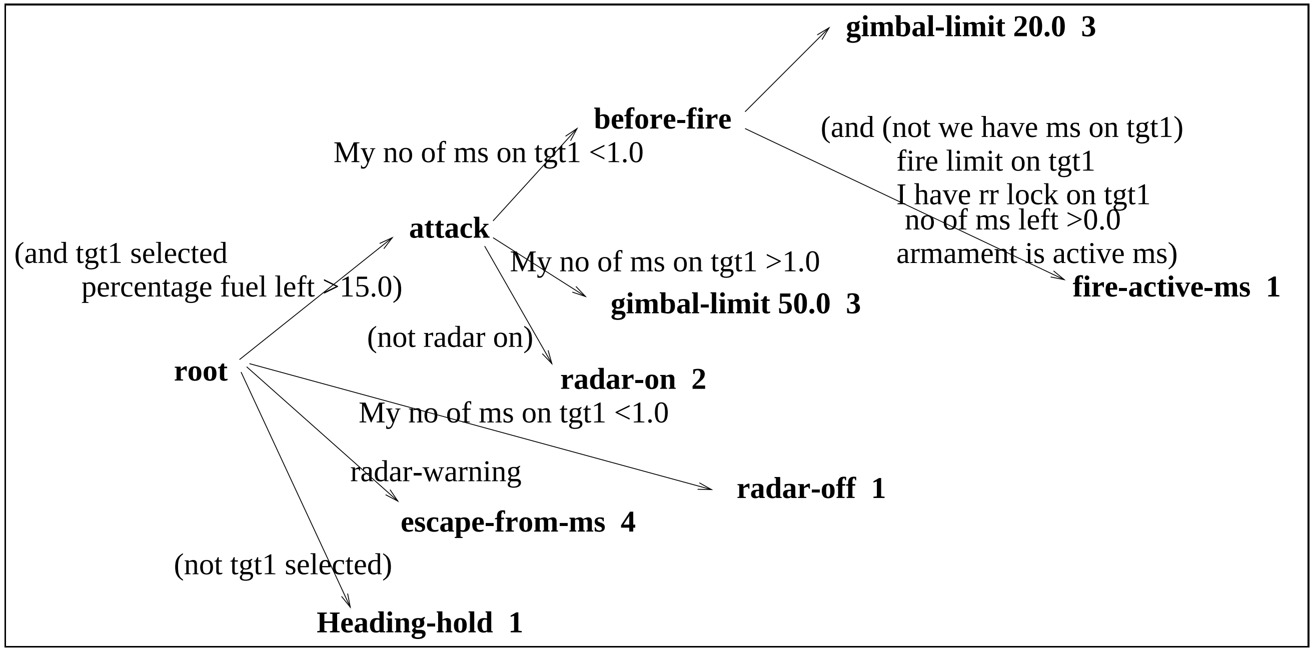

Figure 5.4: The blue interceptor decision-tree

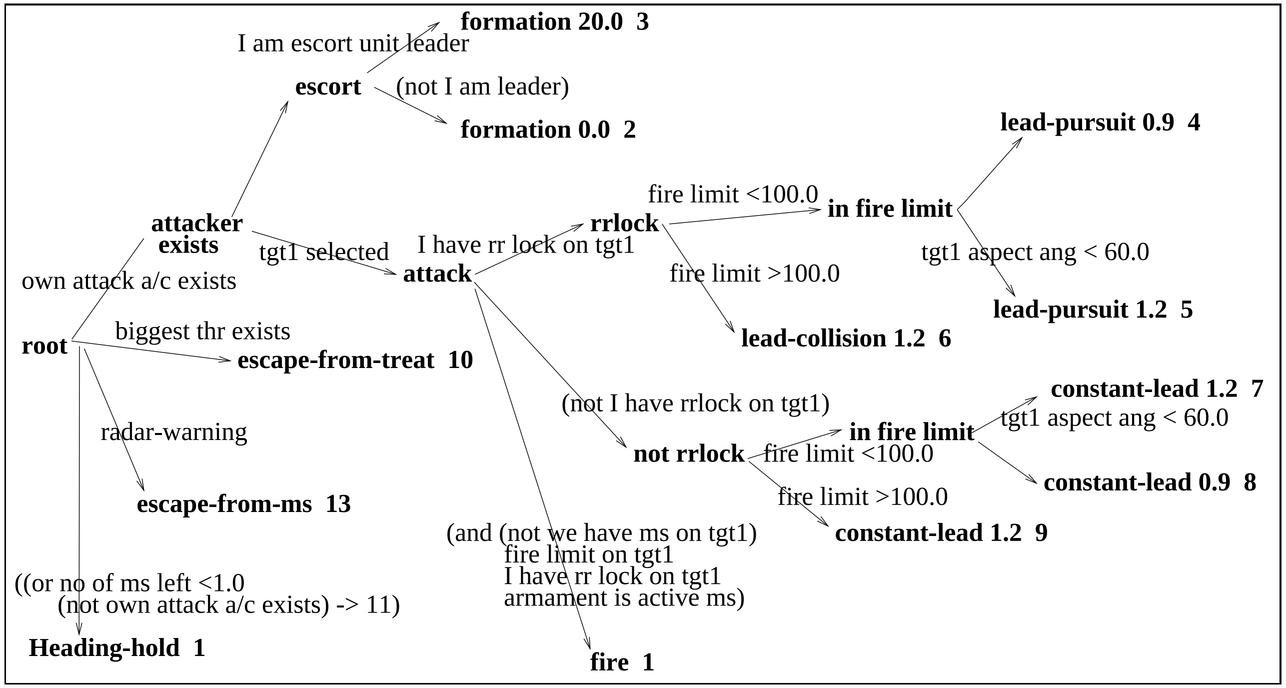

Figure 5.5: The red escort decision-tree

us to try some simple cases of communication in TACSI-LIBRA. In TACSI-LIBRA the agents can perform the actions of sending and receiving messages. After a message is received, it is processed and it is stored in the state of the receiver.

We have tested a simple case of communication in the scenario previously described. In the new scenario the leader of the blue intercept section sends a message to the wingman about when to start firing. The new decision-tree for the blue interceptors is shown in figure 5.6. The new parts with respect to the previous tree are in italics. If the agent is a leader, at the same time that it fires it also sends a "fire" message to the wingman **(con fire-active-ms send-message("fire" wingman))**.[1] The wingman, having an incoming message, performs the action **receive-message** that processes the message and stores it in the state of the agent. As the message has been sent by the leader, the wingman interprets it as an order to fire. The wingman will then fire as soon as the other conditions for firing are satisfied. The condition *(received-command fire)* among the conditions for the action **fire-active-ms** makes the wingman fire only after it has received the command of firing.

5.5 Testing the user-aspect of the system

One requirement of the LIBRA system was the ease in specifying agents' behaviors by people who are not computer science experts. The intention was to have the system tested with respect to this aspect by the actual users of the TACSI system. The TACSI-LIBRA system has been tested together with Saab personnel responsible for the TACSI-system. The specification of the tactics was found quite easy and straightforward. Modifying scenarios and adding new capabilities, as for example the adding of communication in the previous section, has also been quite straightforward.

An important feature to be developed in the future for a better testing of the system will be a user-interface allowing the decision-trees to be specified graphically.

5.6 Summary

In this section we have presented some issues that have arisen in the interfacing between LIBRA and TACSI and an example of a rather complex scenario

[1]Remember that the **con** operator is used to define concurrent actions.

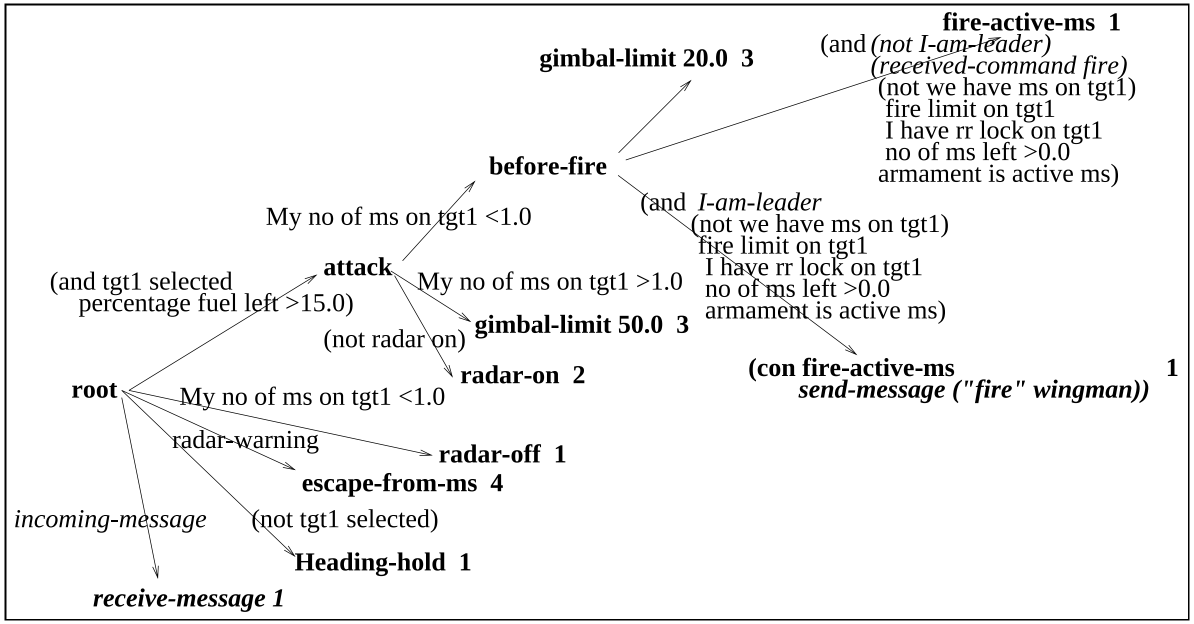

Figure 5.6: The blue interceptor decision-tree with communication added

that we have specified in TACSI-LIBRA. An example of communication between leader and wingman has also been presented. The specification of scenarios has not presented difficulties, but several features are still to be considered. First of all we would like to test the user aspects of the system with the actual users of TACSI. It would also be valuable to test the system on scenarios involving coordination and more complex cases of communication.

Chapter 6

Extension to another domain: the RoboCup competition

6.1 The RoboCup competition

The Robot World Cup Initiative (RoboCup) is an attempt to foster AI and intelligent robotics research by providing a standard problem, playing soccer, where a wide range of technologies can be integrated and examined. RoboCup offers two tasks: one is the creation of a team of multiple fast-moving robots and one is the creation of a team playing using software platform which allows researchers to concentrate on the software aspects of RoboCup.

Since August '96 we have been implementing a team using our multi-agent reasoning system and we have presented our initial results at the RoboCup Workshop held in Kyoto in December '96 [10].

Soccer is an area with some strong similarities to air combat: there are two opposing forces/teams, a high degree of coordination within the teams and real-time decision making in a rapidly changing situation.[1] Therefore it was a natural step to consider how our decision-mechanism could be applied to autonomous agents for a synthetic soccer robot competition. In particular, soccer is a very suitable domain for us to study how coordinated behavior can be specified using the type of rule-based system presented here.

[1]Of course, there are differences as well. Soccer has far more restrictive rules regarding how you can treat your opponents.

Figure 6.1: The RoboCup simulator

In the following section we introduce the RoboCup simulator and in section 6.3 we explain how we have interfaced our system with the simulator. Finally in section 6.4 we consider an example of coordination among players.

6.2 The RoboCup simulator

The RoboCup simulator (Soccer Server) consists of 2 programs, 'soccerserver' and 'soccermonitor'. 'Soccerserver' is a field simulator that calculates the movements of the players and the ball. 'Soccermonitor' is a graphical interface that displays a window representing the field on an X-window terminal. In figure 6.1 the field and ten players are shown. A client controls the actions of the player. Communication between the server and clients is performed using sockets. Each client can control only one player, so a user must invoke the same number of clients as players.

The main actions that can be performed in the Soccer Server are the following:

- *(turn Moment)* Change the direction of the player according to Moment. Actual change in direction is reduced when the player is moving quickly.
- *(dash Power)* Increase the velocity of the player in the direction it is facing with Power.
- *(kick Power Direction)* Kick the ball with Power to Direction if the ball is near enough.
- *(say Message)* Broadcast Message to all players that are near the player that relays the message.

The information received from the simulator is the following:

- *(see Time ObjInfo ObjInfo ...)* provides visual information. Time indicates the current time. ObjInfo is information about a visible object.
- *(hear Time Direction Message)* provides auditory information. Direction is the direction of the sender. Time indicates the current time. This message is sent immediately a client sends (say Message) command. Judgments of the referee are also broadcast using this form.

In Soccer Server the movements of the players and the ball are subject to noise and the information received by the players deteriorates with increasing distance.

6.3 The interface between our system and the RoboCup simulator

The main difference we have encountered in interfacing our system with the Soccer Server compared to the interface made for the TACSI simulator is the complexity of the actions performed by the simulator and of the information received from the simulator. In TACSI the actions to be sent to the simulator are high-level actions that do not describe how to perform an action, but more what the action should achieve. For example, the action **escape from missile** does not say how the aircraft should avoid the missile; the simulator takes care of this. The same can be said for the information received from the simulator.

In the soccer simulator, actions and information are rather primitive and, as mentioned in section 4.4, do not allow decision-trees to be written at a meaningful and convenient level of abstraction. Therefore we have created a layer between the rule layer and the simulator. In this layer procedures are defined that translate high level actions in the low level actions present in the simulator. Also the information obtained by the simulator is preprocessed in this layer and stored in the state in a way that is more convenient for use by the conditions in the decision-tree.

The fact of having a preprocessing layer between the decision-tree and the simulator gives us the possibility to define high level actions in terms of low level actions without the need to be aware of the technical details in the simulator. This allows more freedom in writing new rules. We have experienced some difficulties in writing the rules due to the absence of a preprocessing layer between the decision-tree and the TACSI simulator when we have tried to introduce high-level coordination among team mates in the TACSI system. As this kind of coordination was not considered in the original TACSI, there were not adequate conditions and actions to implement the coordinated behavior. On the other hand, the fact of having high-level actions incorporated in the simulator increases the efficiency of the system, especially in a domain such as air combat where the translation from high- to low-level actions can be quite complex and technically demanding.

6.4 Coordination among soccer players

Coordination in our system is achieved, as mentioned in section 4.7, by a combination of common tactics and strategies, observations of the situation and explicit communication.

The agents act according to their knowledge of the tactics and strategies of the team, initiate tactics depending on the situation and recognize the tactics that the other team members have initiated in order to play their part in it. The agents recognize the tactics that the team is following by observation of the general situation and knowledge of the tactics appropriate

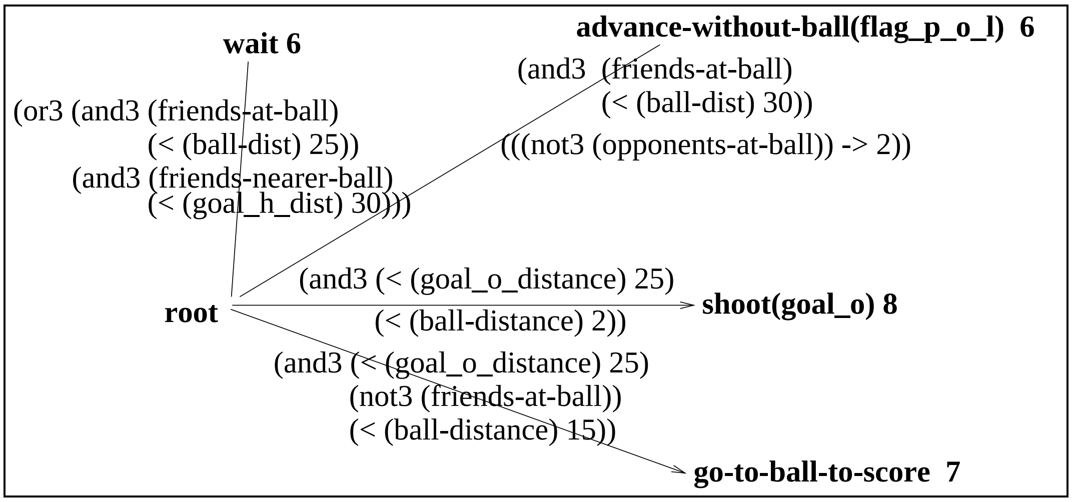

Figure 6.2: Part of the decision-tree of a left attacker

for the situation, observation of indicative actions of team mates, and explicit communication.

In the soccer application it is especially important to limit the use of communication between team mates for achieving coordination. This is due partly to the domain itself. In fact it is important, having a team of 11 players and strong real-time constraints, that tactics and strategies are specified before the match. Further, there are technical constraints in the Soccer Server. In the Soccer Server, the communication between team mates is performed through the action of sending a message, and only the players near the one sending the message can hear it. Therefore the communication is computationally expensive, as it involves an action of sending a message and the processing of the message by the players that hear it and it can be used only within a limited range.

The part of the coordinated behavior associated with each agent involved in the coordination is specified in the decision-tree of the agent. Observations of the situation and of the actions of team mates are considered while choosing the actions to perform.

Let us consider a simple example of coordinated behavior between a left attacker and a midfielder and how we codify it in the decision-tree. Part of the decision-tree of the left attacker is shown in Fig. 6.2 and part of the

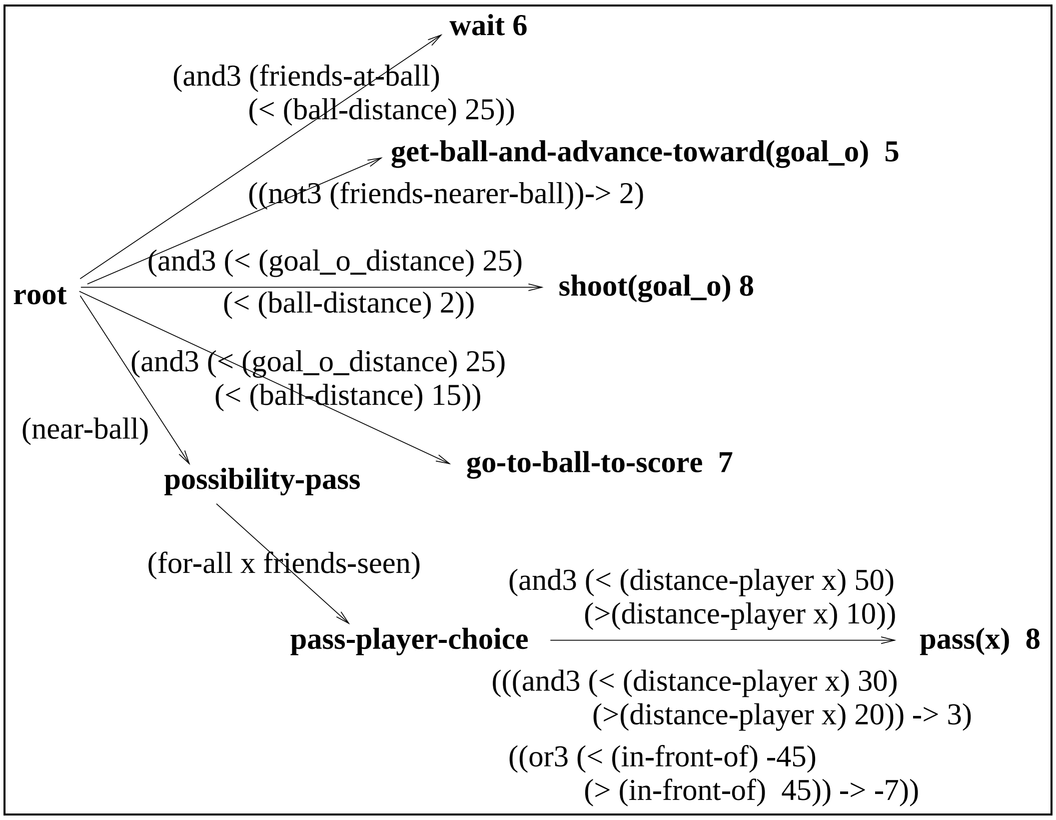

Figure 6.3: Part of the decision-tree of a midfielder

decision-tree of the midfielder is shown in Fig. 6.3.

If no team mate has the ball, the midfielder tries to get the ball and to advance towards the opponent goal: **go-to-ball-and-advance-toward(goal_o)**. If a team mate has the ball it tries to keep a distance from the ball that will allow to support the team mate action. If it has the ball, *(near-ball)*, then it decides to which fellow player to pass the ball among those that it can see (*for-all x friends-seen*) and that are at the right distance:

(and3 (< (distance-player x) 50)
(> (distance-player x) 10)).

The preference as to which player to pass to is based on the distance of the player (if the distance is between 20 and 30 the priority of the action to pass to the player is increased by 3), and also on the distance to the opponent goal (if a player is further than him from the goal, the priority of passing to the player is decreased by 7).

The left attacker, if a team mate has the ball, expects that the player who has the ball will advance and pass the ball. Therefore it will tend to take position at the left corner of the opponent's penalty area, **advance-without-ball(flag_p_o_l)**, but meanwhile keeping a position not further from the ball than 30. It is possible that the player who has the ball will do something else, but the attacker should position in the best place for continuing the action depending on the schema that the player who has the ball is most likely to follow. When it is near the opponent goal, it will try to get the ball and to score (**go-to-ball-to-score** and **shoot(goal_o)**).

Notice that with the approach presented here, the different elements of coordinated behavior of a group of agents have to be specified in terms of the behavior of each agent individually. However, it would be straightforward to provide a decision-tree editor that permits the user to visualize, edit and test the parts of the decision tree that constitute some specific coordinated manoeuvres for several agents at the same time. This could simplify the work on coordinated behaviors considerably, without requiring additional complexity in the representation.

Good guidelines for the user when he describes cooperative behavior are to structure the tree in similar ways for different agents and to use the same names for state parameters for all agents. In addition, good support from the processing of the information is necessary; the user should be able to work with meaningful conditions describing the situation on the field. For instance, a defender playing a zone defense should be able to obtain information on how many opponents there are in his fellow defender's zone in order to judge whether this other defender will need his support or not.

6.5 Related work in the soccer domain

Both in the dMARS system and in the TacAir-Soar system the development of intelligent agents for the air combat domain is now coupled with the development of intelligent agents for the soccer domain.

The problem of building teams of responsible agents using roles and team-plans in the dMARS system has been considered in [5]. A role is a *frame of mind* of the agent that provides information about the agent's expected behavior. In a team-plan the coordinated behavior of the agents is expressed in terms of roles. Roles are represented in a network and the network expresses the responsibilities toward other agents involved in teamwork. The responsibilities derive from having a role and from the relations among roles.

The work is concentrated mostly on team composition and change of roles during the match. The agents have an initial role and team strategies that can change the roles are adopted by the agents during execution. The team strategies are adopted by the agents who reach agreement by broadcasting their roles and team-plans. When the agreement is reached and each agent has adopted a suitable role, messages indicating the execution of each step coordinate the actions.

We consider this work to be of particular interest as we are also in the process of introducing suitable handling of roles in our soccer system. We agree about the importance of roles in coordination and in the need for responsibility related to roles, but the broad use of communication in coordination and role assignment can be questioned. Negotiation about roles and team-plans seems especially difficult to pursue in a domain that presents such hard time constraints.

In TacAir-Soar too, an issue considered in the context of the soccer domain is the responsibility and commitment inside a team especially in context where unanticipated events can disrupt preplanned coordination. The techniques used for commitment in teamwork presented in section 2.2.6 have been applied to the soccer domain in [36]. The central idea is that the reactive team-plans of the team members should explicitly express a team's joint activity. The explicit model of teamwork is based on the joint intentions framework of Cohen and Levesque [6]. The team tracking techniques that we have presented in section 2.2.5 have also been applied to the soccer domain [34].

6.6 Summary

The application of our system to the RoboCup domain has been very rewarding. First of all it has shown that the system is easy to apply in different domains. In fact we have been able to use the system developed for the air-combat application in the soccer domain without any important modification. Furthermore, it has given us the opportunity to experiment in creating a layer between the simulator and the decision-tree and to experiment to a larger extent with coordination and teamwork.

Chapter 7

Conclusions and Future Work

7.1 Summary and conclusions

In this thesis we have presented a system, the LInköping Behavior Representation for Agents (LIBRA), for the specification of agents inhabiting uncertain, real-time environments where reactivity to changes in the environment is combined with coordination with other agents.

The main concept in the system is the one of decision-tree. A decision-tree is a tree of hierarchically structured and prioritized decision rules. By visiting the decision-tree several actions can be selected as candidates to be performed. The selection of those that will actually execute is made on the basis of priority values and compatibility. Actions are compatible if they can be executed concurrently. The priority values can change dynamically as different circumstances can alter the importance of actions and behaviors during the simulation.

Coordination is obtained by common tactics and strategies of the agents. The knowledge required for applying the tactics is codified in the decision-tree and the observations of the situation and of the actions of other agents influence the decisions about which tactics to adopt.

7.2 Future work

The system presented in this thesis is just a preliminary step toward the specification of autonomous agents for complex and real-time environments and much work is still to be done. We present here some of the main topics on which we expect to concentrate our research. The topics in sections 7.2.1, 7.2.3, and 7.2.4 have been already partly developed in [9] and [11].

7.2.1 Commitment to long-term courses of actions (behaviors)

A feature that is missing in the LIBRA system is the commitment to long-term courses of actions and modularity. We intend to add this to the system by introducing behavior modules.

A behavior should specify what actions and sub-behaviors should be executed and under what conditions some specific aims or purposes should be fulfilled. It can either be constantly present or it can be started under certain circumstances, go on for a period of time and then terminate. Behaviors can have different complexities; they can be used for specifying coordination and the proper behavior of a role, and also for specifying simple routines as for example the positioning in the penalty area of a soccer player. Behaviors can recursively contain other behaviors. The same structure is therefore going to be repeated at several levels of the decision mechanism. A behavior is intended to be characterized by a priority, which indicates the importance of the behavior, and a decision-tree in which is coded what the behavior accomplishes.

Behaviors will also add modularity to the system. In fact the internal specification of the behavior should not be visible in the rest of the decision-tree of which it is a part. This will simplify the specification of decision-tree and will increase the reusability of part of the decision-tree in other parts of the tree or in different decision-trees.

7.2.2 Deliberative layer

As mentioned in the introduction, we cannot have truly autonomous agents without a certain degree of deliberation. However, the process of deliberating in domains with strong real-time constraints as the ones examined here can be very time-consuming. Our intention is to consider deliberation just at a high level of reasoning for selecting behaviors, checking their compatibility and monitoring their execution. An important aspect to be considered will be the handling of interruptions of behaviors and errors and exceptions that can occur in the system.

7.2.3 Roles and construction of decision-trees for roles

Roles provide an essential contribution to efficiency and effectiveness of teams. In the absence of roles in a soccer team all players would tend, for example, to get the ball, leaving their penalty area unguarded, or would try to defend the penalty area without leaving some players in the central part of the field ready for a counterattack.

With roles, the resources of the team are distributed depending on the need and can be utilized in an effective way. Furthermore, the coordination is facilitated by the presence of roles both with respect to specification of coordination and to recognition of coordinated tactics. In fact knowing the role of an agent gives an indication of the capabilities of the agent and of the kind of coordinated behavior he is expected to perform. For example an attacker already has a specification of tactics for scoring. Therefore, this part does not need to be considered in the specification of a coordinated attack. Also, knowing that the player who has the ball is a left-midfielder gives an indication to an attacker who wants to be coordinated with him on what direction the attack will take.

In general the roles of the agent are maintained during the entire execution, but in some special cases the roles of the agent can also change during execution. For example, if the team is losing, one midfielder can become an attacker.

An important aspect of roles is the inheritance relation between roles. For example, a left-midfielder is also a midfielder and a midfielder is also a player. This allows the possibility of easily specializing roles without having to redefine the parts common to several roles. Also an agent can have several roles in a team that can be active at the same time as captain and midfielder, or that can alternate depending on the circumstances as midfielder and corner-kick shooter. There are, therefore, two kinds of composition of roles, one for specializing preexistent roles and one for combination of roles in the same agent. The problem in both cases is how to compose roles that are in general independently specified, without having to modify manually one or both the roles in order to deal with possible conflicts of actions and behaviors between roles. For example, a midfielder could try to pursue the ball in whatever position in the field the ball is, but a left-midfielder would be more inclined to pursue the ball if it is in his zone.

7.2.4 Simultaneous encoding of coordinated behaviors for all participating agents

Specifying coordinated behavior requires taking into consideration all the roles involved in the coordination, but then for each agent the importance of the coordinated behaviors should be evaluated in the context of the roles of that agent. Therefore coordinated behavior is intended to be specified in decision-trees for multiple roles, and then the parts of the behavior related to a role are collected, added to the specification of the role and reconsidered in the context of the role. The coordination is then obtained with a totally distributed control. Decision-trees for multiple roles are intended to have the same structure as the decision-tree of an agent, but each action and condition is associated with a specific role.

7.2.5 Assessing the current state of the environment

The problem of assessing the current state of the environment is another challenge. Sensors have in general a certain degree of noise and can malfunction. An area of research of great importance is the sensor fusion area, where the problem of integrating the inputs of several sensors is considered. We have already been working under the supervision of Erik Sandewall on the problem of misperception of the environment by an agent, [7] and [8], in the Features and Fluents framework [31], and it will be valuable to reconsider this problem in the context of the LIBRA system.

References

[1] Saab Military Aircraft. The TACSI user's guide (in Swedish). Technical Report TUCU-MI-95:103, Saab Military Aircraft, 1995. edition 3.1.

[2] J. Andersson. Plan-oriented and rule-based system for generation of robust tactics in long-range air combat with multiple agents (in Swedish). Master's thesis, LiTH-IDA-Ex9575. Linköping University, IDA, 1995.

[3] E. Andrè, J. Müller, and T. Rist. The PPP persona: A multipurpose animated presentation agent. In *Advanced Visual Interfaces*. ACM Press.

[4] N. Bent. The helicopter air-to-air value-driven engagement model (HAVDEM). In *Proceedings of the 19th Congress of the International Council of the Aeronautical Sciences (ICAS)*, volume 3. September 1994.

[5] S. Ch'ng and L. Padgham. Role organisation and planning team strategies. In *Proceedings of the 20th Australian Computer Science Conference*. Sydney, Australia, 1997.

[6] P.R. Cohen and H.J. Levesque. Teamwork. In *Nous*, number 35. 1991.

[7] S. Coradeschi. Reasoning with unreliable observations in the features and fluents framework. Technical Report LiTH-IDA-R-95-44, Department of Computer and Information Science, Linköping University, 1995. Available on WWW: `http://www.ida.liu.se/~silco`.

[8] S. Coradeschi. Reasoning with misperception in the features and fluents framework. In *ECAI96*. John Wiley & Sons ltd, 1996. Available on WWW: `http://www.ida.liu.se/~silco`.

[9] S. Coradeschi and L. Karlsson. A behavior-based approach to reactivity and coordination. 1997. To be presented as a poster at IJCAI'97, Nagoya, Japan.

[10] S. Coradeschi and L. Karlsson. A decision-mechanism for reactive and cooperating soccer-playing agent. In *Linköping Electronic Articles in Computer and Information Science*, volume 2. 1997. http://www.ep.liu.se/ea/cis/1997/001/. Presented at "RoboCup Workshop: Soccer as a problem for Multi-Agent Systems" in the Second International Conference on Multi-Agent Systems (ICMAS-96), Kyoto, Japan.

[11] S. Coradeschi and L. Karlsson. A role-based decision-mechanism for teams of reactive and coordinating agents. 1997. To be presented at "The First International Workshop on RoboCup" at IJCAI'97, Nagoya, Japan.

[12] S. Coradeschi, L. Karlsson, and A. Törne. Intelligent agents for aircraft combat simulation. In *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL, 1996. Available on WWW: `http://www.ida.liu.se/~silco`.

[13] C. R. Eliot and B. P. Woolf. Video: A simulation-based tutor that reason about multiple agents. In *Proceedings of AAAI'96*. Portland, Oregon, 1996.

[14] O. Etzioni and D. Weld. A softbot-based interface to the internet. *CACM*, 37(7):72–76, 1994.

[15] M. Fenster, S. Kraus, and J. S. Rosenschein. Coordination without communication: Experimental validation of focal point and techniques. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*. San Francisco, CA, 1995.

[16] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of IJCAI'89*. Detroit, Michigan, 1989.

[17] J. Görsjö. Extending the TACSI air combat simulator with communication and control. Master's thesis, LiTH-IDA-Ex9629. Linköping University, 1996.

[18] M. J. Huber and E. H. Durfee. Deciding when to commit to action during observation-based coordination. In *Proceedings of the International Conference on Multi-Agent Systems (ICMAS-95)*. San Francisco, CA, 1995.

[19] Decision-Science Application Inc. The TAC BRAWLER air combat simulation management summary. Technical Report 907, DSA, Virginia, USA, 1991.

[20] R. M. Jones, J. E. Laird, M. Tambe, and P. S. Rosenbloom. Intelligent automated agents for flight training simulators. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL, 1993.

[21] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and Asada M. Robocup synthetic agent challenge 97. In *Proceedings of IJCAI'97*. Nagoya, Japan, 1997.

[22] J. E. Laird, R. M. Jones, and P. E. Nielsen. Coordinated behavior of computer generated forces in TacAir-Soar. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL, 1994.

[23] J.R. McNeece and al. X-brawler... applying the science that supports the art of decision making. Technical Report 84/1673, Decision-Science Application Inc., DSA, Virginia, USA, 1995.

[24] H. Moravec. *Mind Children*. Harvard University Press, Cambridge, 1990.

[25] M. Morin and L. Karlsson. Object oriented simulation with WSIM. Technical report, IDA-RKL-94-TR8. Linköping University, 1994. Presented at the 36th Simulation Conference, SIMS'94, Stockholm, Sweden.

[26] P. Nordqvist and L. Törner. WSIM++ a generic environment simulator for real-time experiments. Master's thesis, LiTH-IDA-Ex9642. Linköping University, 1996.

[27] V. Rajan and J. R. Slage. The use of artificially intelligent agents with bounded rationality in the study of economic markets. In *Proceedings of AAAI'96*. Portland, Oregon, 1996.

[28] A.S. Rao and G. Murray. Multi-agent mental-state recognition and its application to air-combat modeling. In *Proceedings of the Distributed Artificial Intelligence Workshop*. Washington, USA, 1994.

[29] A.S. Rao and M. Silvestrel. Agent-oriented architecture for air-combat simulation. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*. Melbourne, Australia, 1993.

[30] P. S. Rosenbloom, J. E. Laird, A. Newell, and R. McCarl. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47:289–325, 1991.

[31] E. Sandewall. *Features and Fluents. The representation of knowledge about dynamical systems. Volume 1.* Oxford University Press, 1994.

[32] O. Shehory and S. Kraus. Cooperative goal-satisfaction without communication in large-scale agent-systems. In *Proceeding of ECAI'96.* Budapest, Hungary, 1996.

[33] M. Tambe. Recursive agent and agent-group tracking in a real-time dynamic environment. In *Proceedings of the First International Conference on Multi-agent Systems (ICMAS95).* Kyoto, Japan, 1995.

[34] M. Tambe. Tracking dynamic team activity. In *Proceedings of AAAI'96.* Portland, Oregon, 1995.

[35] M. Tambe. Teamwork in real-world, dynamic environments. In *Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS-96).* Kyoto, Japan, 1996.

[36] M. Tambe. Towards flexible teamwork in robocup. Presented at "RoboCup Workshop: Soccer as a problem for Multi-Agent Systems" in the 2nd International Conference on Multi-Agent Systems (ICMAS-96), Kyoto, Japan, 1996.

[37] M. Tambe, W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), 1995.

[38] M. Tambe and P. S. Rosenbloom. Resc: An approach for dynamic, real-time agent tracking. In *Proceedings of IJCAI'95.* 1995.

[39] M. Tambe and P. S. Rosenbloom. Architectures for agents that track other agents in multi-agent worlds. In *Agents, Theories, Architectures, and Languages (ATAL-95).* Springer Verlag Lecture Notes in Artificial Intelligence (LNAI 1037), 1996.

[40] G. Tidhar, M. Selvestrel, and C. Heinze. Modeling teams and team tactics in whole air mission modelling. In *Proceedings of the 8th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*. Gordon and Breach Science Publishers, Melbourne, Australia, 1995.