The Agent Architecture InteRRaP: Concept and Application

Jörg P. Müller[1], Markus Pischel[2]
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-6600 Saarbrücken 11

[1]e-mail:jpm@dfki.uni-sb.de
[2]e-mail:pischel@dfki.uni-sb.de

Abstract

One of the basic questions of research in Distributed Artificial Intelligence (DAI) is how agents have to be structured and organized, and what functionalities they need in order to be able to act and to interact in a dynamic environment. To cope with this question is the purpose of models and architectures for autonomous and intelligent agents.

In the first part of this report, INTERRAP, an agent architecture for multi-agent systems is presented. The basic idea is to combine the use of patterns of behaviour with planning facilities in order to be able to exploit the advantages both of the reactive, behaviour-based and of the deliberate, plan-based paradigm. Patterns of behaviour allow an agent to react flexibly to changes in its environment. What is considered necessary for the performance of more sophisticated tasks is the ability of devising plans deliberately. A further important feature of the model is that it explicitly represents knowledge and strategies for cooperation. This makes the model suitable for describing high-level interaction among autonomous agents.

In the second part, the loading-dock domain is presented, which has been the first application the INTERRAP agent model has been tested with. An automated loading-dock is described where the agent society consists of forklifts which have to load and unload trucks in a shared environment.

Contents

List of Figures

Preface

Distributed Artificial Intelligence (DAI) is the subfield of AI concerned with concurrency in AI computations. Bond and Gasser [BG88] divide the world of DAI in two primary arenas: Research in *Distributed Problem Solving* (DPS) investigates how the work of solving a particular problem can be divided among a number of "nodes" or modules that cooperate at the level of dividing and sharing knowledge about the problem and about its solution. The second arena, called *Multiagent systems* (MAS), deals with coordinating intelligent behaviour among a collection of (possibly pre-existing) autonomous intelligent agents. Emphasis is placed on how these agents coordinate their knowledge, goals, skills and plans jointly to take action or to solve problems. Like modules in a DPS system, agents in a multiagent system must share knowledge about problems and solutions. However, apart from these issues, they also have to reason about the *process of inter-agent coordination* itself.

For a long time, the problem of agent coordination was by the metaphor of the cooperating expert society, for which Hewitt, in his early ACTORS work, raised the raised the broad research question of " what should be communication mechanisms and conventions of civilized discourse for effective problem solving by a society of experts?" ([Hew77]). The cooperating expert paradigm dominated the research in DAI for more than a decade. In the field of agent architectures it provided the basis for developments like Lenat's "Beings" ([Len75]), Hewitt's ACTORS (cf. e.g. [Hew73]), and for blackboard systems such as HEARSAY ([EHRLD80]) or the DVMT testbed ([CL88]).

Since the late eighties, research in Multiagent Systems has paid more attention to particular concepts that are of relevance for the coordination in dynamic agent societies, such as *cooperative planning* ([LBS92, Jen92, KvM91]), *conflict resolution* [Kle90, Syc88], and *negotiation*, ([Syc89, DL89, ZR91]). The purpose of particular agent models was now to provide a framework for integrating instances of these concepts required to deal with a particular domain of application.

In addition to this, there are quite a few more good reasons to concentrate on the agent architecture in the first place, and to use one of its instantiations in order to describe the actual process of problem solving:

- The architecture provides a valuable general guideline for the methodology of the design and the implementation of an application.
- The modules of the agent model precisely structure the classes of operational knowledge.
- The execution model which is implicit to the architecture avoids programming from scratch.
- Application-independent, predefined mechanisms such as negotiation protocols (e.g. the Contract Net) are directly available.
- The emergent functionality of the society can be predicted up to a certain level by regarding the basic patterns of interaction of the instantiated agents.

- An agent architecture provides a basis for the investigation of special strategies and of extensions of the modules.

The INTERRAP Agent Model

The agent model INTERRAP, which is presented in the following, is an extension of the RATMAN model [BM91]. INTERRAP was developed in order to meet the requirements of modeling dynamic agent societies such as interacting robots. Its main feature is that it combines patterns of behaviour with explicit planning facilities. Patterns of behaviour on the one hand allow an agent to react quickly and flexibly to changes in its environment. On the other hand, the ability to devise plans is generally regarded necessary to solve more sophisticated tasks. INTERRAP has been evaluated using three applications: (1) the implementation of a society of cooperating vehicles in a loading-dock [MP93], (2) the MARS system, a simulation of cooperating transportation companies [KMM93a], and (3) COSMA, a distributed appointment scheduling manager [Sch92].

The INTERRAP Architecture

While the novel feature of RATMAN - the idea of structuring a knowledge base according to the complexity of the knowledge contained - was commonly accepted, there was one main point of criticism of the system, namely the lacking separation between aspects of the *knowledge* used in the and the *functionality* shown by the model: the hierarchically structured levels of knowledge were not only constructed using the concepts of the lower levels, but they were also used to trigger activities at these lower levels.

INTERRAP clearly draws the separation between the pure knowledge base and the functional part, while preserving the hierarchical structure of the model. Thus, the two parts of the INTERRAP model are

- the hierarchical agent knowledge base, and
- the multi-stage control unit.

Figure 0.1 shows the INTERRAP model in more detail.

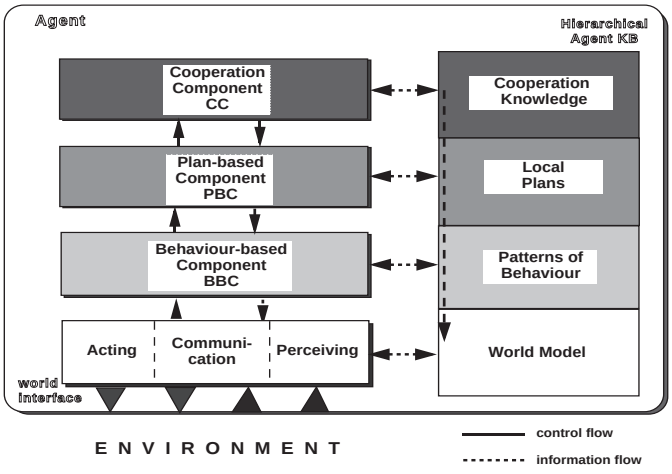

Figure 0.1: The INTERRAP Agent Model

The Agent Knowledge Base

The lowest level of the Agent KB contains the world model of the agent. It is organized as a taxonomical knowledge base. This kind of knowledge represents the objects in the world and the relationships which hold among these objects (which corresponds to the standard T-Box/A-Box structures). The second level describes the patterns of behaviour and the basic actions an agent can perform. A plan library, given as a set of skeletal plans, is modeled at the third level. The plans are defined recursively starting from basic actions, patterns of behaviour, or uninstantiated subplans. Finally, knowledge about cooperation and coordination, such as communication and negotiation protocols, and *joint plans* (which are basically multi-agent plans) is represented at the highest layer of the hierarchy.

The Control Unit

The agent control reflects the hierarchical structure of the knowledge base. It shows the operational flow as discussed in the RATMAN model [BM91], from the world interface level, where sensoric data is perceived, up to the behaviour-based level, to the plan-based and cooperation levels, and back again to the interface level, where finally actions in the world are performed. On the other hand, it was built according to the idea of combining the rational, plan-based paradigm with the concept of behaviour-based, reactive systems and situated actions [Bro86, Suc87, Ste90]. The four components of the INTERRAP agent control as shown in figure 4 are the world interface, the behaviour-based component (BBC), the plan-based component (PBC), and the cooperation component (CC).

Instead of discussing the single levels one by one (see [MP93] for a detailed description), at this stage, the flow of control and information through the different stages will be outlined. The lowest level reflects the input/output interface of the agent, the perception of changes in the world, and the receiving of messages. This information passes a first filter and flows into the world model of the agent. It is the basic information used by the BBC. There, it may either directly trigger a certain pattern of behaviour (e.g. the pattern "avoid collision" which has the agent moving aside)[1].

If there is no need for such a fast response, or if the situation is too complex to be coped with by the BBC, control is shifted up to the plan-based component. This component contains the agent's facilities for planning and local decision-making. If the actual situation requires cooperation and coordination with other agents (such as resolving a blocking conflict between two forklift agents in a narrow shelf), the PBC passes control to the CC, where a cooperative solution of the problem is worked out (for example a joint plan for resolving the conflict). In any case, the order for the next working step is passed to the next lower level:

- a joint plan is transformed into a set of single-agent plans together with a set of synchronization commands (representing the constraints among the plans) and passed to the PBC.
- a pattern of behaviour is activated by the PBC.
- the performance of basic actions or the sending of messages via the world interface is activated by the BBC.

Finally, note that each component of the agent control has access to its corresponding layer in the agent knowledge base *and* to all lower layers, but not to higher layers. For example, a pattern of behaviour has never access to the representation of plans.

[1] Note that, since patterns of behaviour may be concurrently active, the BBC needs a hard-wired control mechanism for coordinating these patterns. This must not be confounded with the deliberate mechanisms for decision-making located at the plan-based and the cooperation layer.

So far, we have given a brief overview of the InteRRaP agent model which underlies our applications. The report at hand is divided in two parts. The first part provides a thorough introduction to the InteRRaP agent model. The single modules of an InteRRaP agent and their interplay are discussed. Part two presents the loading-dock domain, which has been the first application our agent model has been tested with.

Chapter 1

Introduction

Over the past few years, Distributed Artificial Intelligence (DAI) has been recognized as a suitable approach for describing complex and dynamic distributed systems. Research in DAI explores [BG88] either how a group of intelligent and autonomous computational systems (agents) coordinate their knowledge, goals, plans, and skills (multi-agent systems, agent-centred approach) in order to achieve certain goals, or how the solution of a specific problem can be distributed among a set of nodes (distributed problem-solving, problem-centred approach).

One of the basic questions of research in Distributed Artificial Intelligence (DAI) is how agents have to be structured and organized, and what functionalities they need in order to act and interact coherently in a dynamic environment, and thus, in order to bring about an emerging functionality of the system as a whole. To cope with this question is the purpose of models and architectures for autonomous and intelligent agents.

1.1 Agent Models

When it comes to develop models of agents, there are two basic paradigms. One is the paradigm of behaviour-based, reactive agents. The other is the paradigm of plan-based, deliberate agents (see e.g. [WD92] for examples of both directions). Advantages and shortcomings of both approaches have been thoroughly discussed, and it has been widely accepted inside the DAI community, that an agent should have both reactive and deliberate abilities.

In the first part of this report, we introduce the INTERRAP[1] agent architecture. INTERRAP is a further development of the agent architecture RATMAN [BM91]. It extends RATMAN by the following aspects:

- Whereas RATMAN described a purely rational[2] agent, INTERRAP is a *hybrid* agent architecture[3] The basic data structures are patterns of behaviour and plans. The former allow an agent to react flexibly to changes in its environment, but, moreover, are a suitable means of expressing all activities which do not require the agent to have an explicit representation in the form of a plan, such as activities containing procedural knowledge, or the performance of routine tasks. The latter are often necessary if more sophisticated tasks have to be performed.

- INTERRAP provides a clear separation of control and the agent knowledge.

[1] **Inte**gration of **R**eactive Behaviour and **Ra**tional **P**lanning

[2] Here, the word *rational* has the meaning of *logic-based*. Compare the discussion in chapter 2 for a more detailed discussion of the different meanings of rationality.

[3] See chapter 7 as well as [HR89, SH90, CGHH89, GI89, Oga91, Fer92, Had93] for recent examples of hybrid agent architectures.

- INTERRAP preserves the hierarchical knowledge base which was commonly regarded as the novel feature of the RATMAN model. Furthermore, the agent control unit is organized hierarchically, too. It consists of the world interface, the behaviour-based component, the plan-based component, and the cooperation component.

- Cooperation knowledge and control are represented as an explicit part of the model.

In the first part of this report, the components of the model and the interplay of the control modules are described.

1.2 A Motivating Example

To give the reader a first idea of the general principle of how the concepts of patterns of behaviour and plans are to be understood, and of how their use can be combined to yield flexible and intelligent behaviour, we will provide an example from "everyday" life:

> *Imagine you are sitting in your office and reading your mail which - for you - is rather a routine pattern of behaviour than a real intellectual effort. Suddenly, you start feeling tired and you feel that having a cup of coffee might be a good idea. Of course, you do not run around randomly until you encounter a cup of coffee waiting for you. Instead, your planning component takes control and decides to go to the coffee bar at the other end of the corridor. Your planner devises an abstract plan: leave your room, walk down the corridor, enter the coffee bar, get coffee, go back to your room. Now you start the execution of the plan: walking straight down the hallway is a routine task you learned as a little child. Thus, you have an unconscious mechanism of execution for doing this kind of action at your disposal, a* pattern of behaviour*.*
>
> *Suddenly, while you are walking down the corridor, a door opens, and an eager colleague of yours rushes out of his office and gets in your way. Instinctively, you step aside and avoid a collision without leaving your current behaviour - this is a case of an exceptional situation which can be handled from within the active pattern of behaviour. However, when you reach the door to the coffee bar and become aware of the fact that it is locked, you find yourself in a situation your active pattern of behaviour cannot cope with. Thus, from inside it, again the planner has to be activated to find a solution to this situation, which might be to go and look for the secretary in order to get the key to the coffee bar.*

1.3 The Application

In the second part of the report, we show how a multi-agent system for the simulation of a robotics application can be designed using INTERRAP: a group of automated forklifts is modeled which have to load and unload trucks in a loading dock. The concepts presented in the first part of the report are made clear by means of a detailed example. Although we regard the scenario primarily from a multi-agent perspective, aspects of traditional robotics such as perception and path-planning [Bro86, FD90, ST92, Lat92] have to be considered.

For several reasons, the field of interacting robots has turned out to be a rewarding test-bed for evaluating architectures and mechanisms of DAI: Firstly, in these environments, control and information are inherently distributed. Secondly, the domain is highly dynamic and complex, and therefore requires both flexible and intelligent behaviour of agents. Thus, it is a touchstone for architectures and methods from both behaviour-based and plan-based approaches. Thirdly, there is a wide range of possible interactions among the members

of robot societies, ranging from mere collision avoidance to actually cooperating robots (cf. [MP93]).

1.4 Overview

The report is structured as follows:

- Part one describes the INTERRAP agent architecture. After briefly outlining our terminology in chapter 2, we present a general conceptual agent model in chapter 3. In chapter 5, we describe the process of decision-making, which closely corresponds to prioritizing goals and patterns of behaviour. The contribution of the INTERRAP architecture to the field of dynamic planning is discussed in chapter 6. Chapter 7 provides an overview of related work.

- Part two presents the loading dock application. The domain, the problems involved, and the aspects which make the domain suitable for a DAI approach are explained in chapter 8. Chapter 9 provides a detailed description of how the loading dock was modeled using the INTERRAP architecture. Chapter 11 explains the flow of control among the modules of the model by means of an example. In chapter 12, we give details of the current state of the implementation and reprot preliminary results. Chapter 13 includes the conclusion and an outline of important future work.

Part I

The InteRRaP Agent Architecture

Chapter 2

Terminological Remarks

Terms such as "deliberate", "plan-based", "rational", "behaviour-based", or "reactive", which we use in our work in order to describe properties of agents and systems of agents have many connotations in DAI research. Therefore, in the following, we should like to explain the meanings we attribute to these notions.

2.1 Deliberate, Plan-based, and Rational Agents

The term *rationality* has been used in the literature in order to address very different phenomena. In [BIP88], Bratman defines rational behaviour as "the production of actions that further the goals of an agent, based upon [its] conception of the world." Here, rationality is used in the sense of *goal-directed behaviour*, fairly independent from the agent having a specific type of representation of the world. In game theory [LR57, Ros85], rationality is used in a narrower sense. An agent which is *individual rational* will always do the thing which produces the best result with respect to a local utility function. Agents acting according to this principle are called utility maximizers. Finally, the standard AI meaning of rationality differs from the one defined by Bratman in that it places emphasis on how the conception that an agent has of its world is represented, and on according to what principles an agent makes its decisions. Thus, a rational agent is defined as an agent who has a logic-based representation of the world and who uses logical methods of inference as a means of making decisions.

An agent is called *deliberate* if it possesses an explicit representation of its mental state, of its beliefs, goals, and plans [Fer92], and if it has abilities to reason about its mental state in order to determine how to behave at a given point in time. Thus, the term *deliberative* subsumes the qualities of being directed towards a goal and of representation.

Plan-based agents also have explicit representations of their mental state. Although - unlike *deliberate* - the term *plan-based* does not suggest that and how agents reason about the representation of their knowledge, we use "deliberate" and "plan-based" as synonyms, because plan-based agents virtually always have the ability of reasoning about the structure of their plans [FHN71].

In this report, we will prefer the terms "deliberate" and "plan-based" to "rational", since they express better the intended meaning.

2.2 Behaviour-based versus Plan-based Agents

We draw a distinction between plan-based and behaviour-based agents by applying the criterion of representation: plan-based agents maintain explicit symbolic representations of their plans and are able to reason about the structure of these plans [FHN71]. Behaviour-

based agents are described by patterns of behaviour which, unlike plans, do not have an internal structure but are considered as black boxes [Suc87, Ste90]. Patterns of behaviour are closely linked to their execution. In this respect, the relationship between plan-based and behaviour-based agents corresponds to that between the declarative and the procedural paradigm [Win75]: the activities of behaviour-based agents are represented procedurally whereas plan-based agents are built according to the declarative paradigm[1].

2.3 Behaviour-based and Reactive Agents

In the literature, "behaviour-based" is often used as a synonym for "reactive". Behaviour-based agents are considered to be stimulus-response systems which are triggered by external trigger conditions [Fer89, Ste90]. Our definition of behaviour includes this relationship: since a pattern of behaviour is always linked to performing actions in the world, behaviour-based agents must be reactive in a sense that they can recognize changes in the environment and adapt to them.

However, we feel that the behaviour-based paradigm has more to offer than just describing reactive agents: to us, patterns of behaviour are a viable way to express any activity an agent can do without much reflection - and without providing an explicit plan structure for it. For example, walking down a hallway or driving a car are examples of such "hard-wired" *routine tasks*. Patterns of behaviour offer a possibility to perform these routine tasks more or less subconsciously. Minor events may be tackled from inside the behaviour, whereas in the case of unforeseen situations either different patterns of behaviour or explicit deliberation (i.e. planning) have to be activated. Thus, patterns of behaviour can be regarded as compiled multiplans, i.e. sets of plans submitted to an intelligent execution mechanism.

[1] In [GL86], the use of procedural knowledge is proposed by the authors. This closely corresponds to the procedural aspect we attribute to patterns of behaviour.

Chapter 3

A Conceptual Agent Model

It is the aim of a conceptual model of an agent to define an abstract interpreter for the agent, and thus, to describe what determines the behaviour of an agent in a dynamic environment. We draw a distinction between two basic kinds of decisions an agent has to make. Firstly, the agent has to decide what goals to pursue. This process is modeled by the *descriptive layer* of the model (which we also call the macro model). Secondly, the agent has to decide what mechanisms and strategies to use in order to achieve its goals. This is called the *execution layer* (*micro model*). To draw the distinction between these two layers has turned out to be very useful, since it allows a conceptual separation between problems and the methods used to solve them. It allows us to model and to compare different (plan-based and behaviour-based) strategies for one and the same situation. In figure 3.1, the conceptual

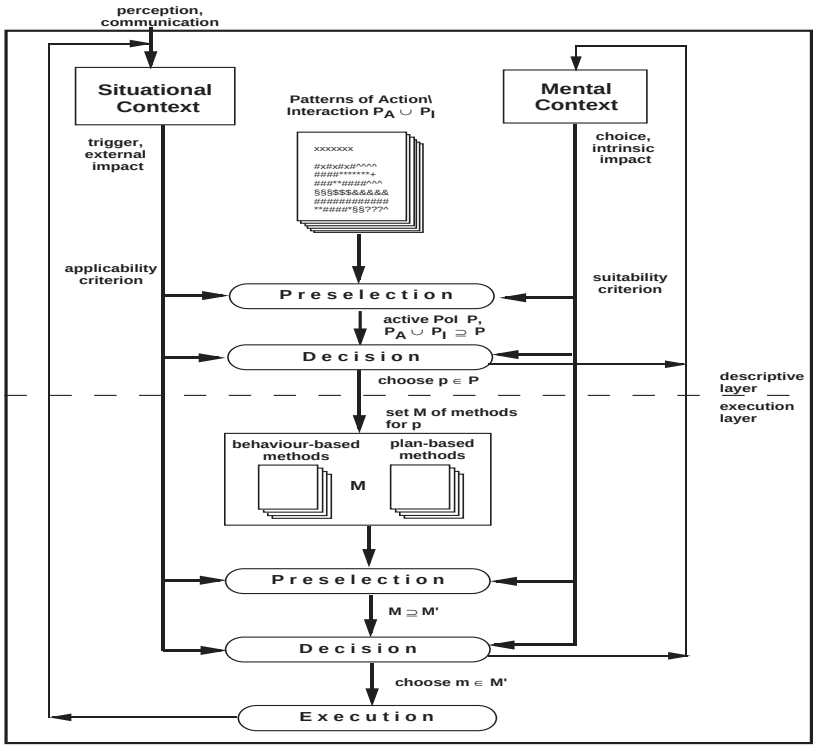

Figure 3.1: Structure of the Conceptual Model

INTERRAP model of action and interaction is shown. The single components are described in the following.

3.1 The Descriptive Layer

3.1.1 Patterns of Activity

An agent has a set of script-like patterns of action P_A and patterns of interaction P_I at its disposal which describe classes of situations the agent is faced with. In the following, we will refer to elements of the set $P_A \cup P_I$ as *patterns of activity*. When defining these patterns, we make a clear conceptual distinction between the declaration part (describing the representation, recognition, evaluation of situations and the expected outcome of executing the patterns) and the execution part itself, which is represented by a pointer to a set of methods. The basic idea of these patterns is that their declaration contains information which defines when they are applicable, suitable, and which helps an agent to evaluate their expected utility against alternative patterns. Therefore, the definition of a pattern of activity contains information about

- What are the external preconditions for the pattern to become active? We call this extrinsic impact the *situational context* of a pattern. For example, the situational context for a blocking conflict between two agents is that one agent stands in front of the other one[1].

- What is the mental state the agent must be in in order to make the pattern suitable, i.e. what goals does the agent currently pursue which affect the patterns, what beliefs must it have about itself, and other agents? For example, for a blocking conflict to exist, it is not sufficient to say that another agent stands in front of that agent. Additionally, the agent itself must have the goal to move to the field which is currently occupied by the other one. The achievement of this goal is blocked by the conflict.

- What are the postconditions of the execution of the pattern, i.e. what holds true after it has been executed?

- What are the termination conditions, i.e. what are criteria for determining a successful termination of pattern execution?

- What are the failure conditions, i.e. conditions that enforce to stop the execution before it has been completed?

- Information about the participants (only in case of a pattern of interaction): are there special requirement to be placed on a partner to be involved in the execution of the pattern. This information can be used for instance in order to choose appropriate cooperation partners.

- Criteria for an *a priori* evaluation of a pattern of activity. Since committing to a certain pattern may start a lengthy and costly process of (inter)action, it is very important for the agent to be able to evaluate the pattern against other, possibly local alternatives. Evaluation critria are e.g. the expected utility of executing the pattern, the probability of success, or the urgency of the performing the pattern (e.g. how urgent it is to resolve a currently existing conflict).

- The description of the execution. In the simplest case, in the description of a pattern, execution is represented only by a pointer to an executable piece of program code, together with a specification of the arguments of the program call. However, for a couple of reasons, a more structured description seems useful. This subject will be

[1] Strictly speaking, we define patterns of interaction from a local, agent-based point of view. Therefore, the situational context for a blocking would be described rather as "another agent stands in front of me".

treated in more detail in section 4.2.2, where we discuss our preliminary considerations towards an abstract description language for patterns of behaviour, which are the functional counterparts to the concept of patterns discussed in this chapter.

- Information about the capability context. Since agents are often defined as generic agents, which may be instantiated with different capabilities by the designer of a concrete system, for different configurations of agents, different patterns and different execution mechanisms are available. The capability context defines for which agent configurations the pattern can be used. For example, patterns may be used only by agents who have facilities to communicate, to reason about other agents' goals, to plan aso.

We do not claim that the above listing contains all the criteria which might be desirable in order to give a complete description of a pattern. On the other hand, not all the slots discussed above may be required in order to define a pattern in the specific case.

Our characterization of patterns treats them as *abstract actions*. Therefore, our effort to describe them bears similarity to much of the work done in AI planning (see chapter 6 and chapter 7), where plan steps are described by using preconditions, during conditions, post conditions, and so forth. In fact, what we aim at by providing this information is enabling the appropriate control module in the agent model (see next chapter), which can be the behaviour-based, plan-based, or cooperation module, to make decisions which patterns to pursue and which patterns to drop. A central issue of our future work is to define an abstract specification language for the patterns, which can be used by the system designer to define these patterns on a higher level, and independent from using a rule-based programming language such as OPS-5, which currently serves us as the implementation language for the patterns of behaviour. We expect that this abstract language will contain basically the features discussed in this section.

3.1.2 Situational Context and Mental Context

Two contexts are especially important to deterrmine how an agent is to behave. Therefore, they will be discussed in more detail in this section: the *situational context* (extrinsic impact) defines the patterns which are applicable in a certain situation. The *mental context* (intrinsic impact) describes which patterns are suitable for the agent with respect to the achievement of its current goals, and against the background of the agent's current beliefs about other agents and about its own capabilities. This distinction reflects a main feature of Lewin's theory of personality [Lew35]: it says that the behaviour of an individual depends on properties of its personality (which we represent in a simplified form by the agents' goals and by the decisions derived from these goals) on the one hand, and on situational conditions, on the other. Together, the two contexts describe a pre-selection and reduce the set of patterns $P_A \cup P_I$ to a subset of active patterns $P \subseteq P_A \cup P_I$. In a decision step, one pattern $p \in P$ is chosen for execution[2]. Evaluation information contained in the pattern declaration is used to make this choice.

3.2 The Execution Layer

If a certain pattern p has been selected, the agent has the choice among a set M of different possible mechanisms for the execution of p. These mechanisms may be either behaviour-based or plan-based. Again, depending on the situational and mental context, only a subset $M' \subseteq M$ is suitable in a certain situation. One of these mechanisms, let us say $m \in M'$ is

[2]If we assume concurrency, a set $P' \subseteq P$ of patterns can be executed in parallel.

chosen and executed. Since new subgoals are derived from this decision, the mental context may be changed. By executing the pattern p using mechanism m, the external situation (and thus, the situational context) is changed, and by this feedback, the loop of the agent cycle is closed.

Mechanisms of Execution: an Example

Let us look at an example taken from the loading-dock domain which is presented in more detail in the second part of this report, to see *behaviour-based* and *plan-based* mechanisms of action and interaction. A basic behaviour-based mechanism of activity is randomness. In a finite search space, making random moves is a (mostly inefficient, but) safe way to achieve a goal [Ste90]. Randomness is also very useful in resolving conflicts, especially in symmetric situations, for example, a situation where two agents are standing in front of each other. In these situations, following other, deterministic, strategies often cannot resolve conflicts, since both agents are likely to perform symmetrical actions at the same time, and since, in case a conflict solution is negotiated, there are no arguments one agent can put forward which could not be used by the other agent with the same right. Game theory [ZR89] proposes to toss a coin in these cases - which exactly corresponds to applying the principle of randomness.

Forklift agents can use potential field methods or weighted randomness in order to reach their destination and in order to avoid collisions. We consider these mechanisms to be basically behaviour-based, but it is hard to find a clear separation line between these and plan-based mechanisms. Definitely plan-based methods are representation-based path planning algorithms (e.g. the Voronoi algorithm ([OY82]), as well as the generation of joint plans in order to resolve conflict situations or to achieve transportation goals cooperatively.

3.3 An Abstract Agent Interpreter

Figure 3 can be regarded as the graphic description of an agent interpreter. An agent interpreter explains the behaviour of an agent as a function the agent's capabilities, of the actual external situation, and of the mental state of the agent. We describe the agent's capabilities by

- a set of patterns of action / interaction the agent can recognize,
- a set of methods the agent has at its disposal in order to execute the patterns,
- preselection functions π_P and π_M for patterns and methods, respectively, and
- decision functions δ_P and δ_M for patterns and methods, respectively.

In the following, we give a more formal definition of these notions. Let S_a^t be the situational context for agent a at time t, and let G_a^t be a's mental context at time t. We define an agent a as a tuple

$$a = (P, M, (\pi_P, \pi_M), (\delta_P, \delta_M)),$$

where

- $P = P_A \cup P_I$ is a set of patterns of action / interaction as presented in chapter 3,
- $M = \bigcup M(p)$ is a set of methods available for the patterns $p \in P$,
- $\pi_P : 2^P \times S \times G \mapsto 2^P$, $\pi_M : 2^M \times S \times G \mapsto 2^M$, are preselection functions, which map sets of patterns and methods into one of their subsets.

- $\delta_P : 2^P \times S \times G \mapsto P \times G$, $\delta_M : 2^M \times P \times G \mapsto M \times G$ are decision functions which select one element from the sets od patterns an methods, respectively, and which modify the mental context of the agent by generating the goal (commitment) of pursuing / executing the selected pattern.

What we still need is a function which explains the effect the execution of an action has on the situational context. For this purpose, we define a function $Exec : M \times S \mapsto S$ which describes how the execution of a method changes the actual state of the world.

Now, we can define the behaviour of agent a at time t as follows.

Definition 1 (Agent Interpreter) *Given an agent $a = (P, M, (\pi_P, \pi_M), (\delta_P, \delta_M))$, its situational context S_a^t and its mental context G_a^t at time t, the (external) behaviour of agent a at time t can be determined as*

$$Beh_a^t ::= Exec(\delta_M(M', S^t, G'^t)|_M, S^t)$$

with

- $M' ::= \pi_M(M(p), S^t, G')$,
- $G' ::= \delta_P(P', S^t, G^t)|G$,
- $P' ::= \pi_P(P, S^t, G^t)$, *and*
- $p \in P$ *is computed as* $p ::= \delta_P(P', S^t, G^t)|_P$.

$\delta|G$ means the state of G resulting from applying $\delta_P(P, S, G)$ and $\delta_M(M, S, G)$, respectively.

The effect of Agent a's actions on the situational context at time $t+1$ is determined by the function Beh_a^t defined above. Similarly, the new mental context of agent a at time $t+1$ is computed as

$$G_a^{t+1} ::= \delta_M(M', S^t, G')|_G,$$

where M', G' are as defined above.

It is important to note that the above agent interpreter does not define the state of the world resulting by the actions made by all agents. It merely defines how the world changes by the behaviour of a single agent derived in one agent cycle.

If we have multiple agents, the world changes by actions performed by each of these agents. In order to describe such a system, our model is to be extended similar to [HM90, HM92]. There, a distributed system is described as a finite set $\{p_1, \ldots, p_n\}$ of processors (agents) that are linked by a communication network. The system as a whole is described by the set of possible runs R. A run $r \in R$ of the system describes the execution of the system as a whole, from time 0 until the execution has finished. The knowledge ascribed to the single processors is described as a Kripke structure $M = (S, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, where S is a set of states (possible worlds), π is a truth assignment for the primitive propositions of the logical language underlying, and $\mathcal{K}_i$ is a binary relation on elements of S, for $i = \{1, \ldots, n\}$. The intended meaning of the K relation acording to agent i is the following: $(s,t) \in \mathcal{K}_i$ if in world s in structure M, agent i considers t a possible world.

Given a state s, the knowledge of an agent can be defined by an operator K:

$$(M, s) \models K_i\phi \text{ iff } (M, t) \models \phi \text{ for all } t \text{ satisfying } (s, t) \in \mathcal{K}_i.$$

Since we are not interested in a description of the system as a whole at this stage, but rather concentrate at describing the single agent and its interaction with the environment, our approach seems sufficient for us. The agent recognizes the behaviour of other agents and interactions with other agents by means of perception and receiving messages. Perception directly models the situational context.

The agent interpreter developed in this section describes an abstract, conceptual model of an agent. Therefore it serves primarily to understand how the agent is to work, and, at the current stage of our work, not so much as a direct basis for the implementation of the agent.

In the following, we will show how the model is implemented by defining a functional agent model. The functional model describes the information processing and the control units of the agent; in order to provide efficient processing, it is designed using a layered architecture.

3.4 Designing Agents

The conceptual model described above helps the designer of a multi-agent system modeling agents which behave adequate with respect to the domain to be modeled: if the application requires rather reactive, behaviour-based agents, the designer can meet this requirement by defining patterns which mostly depend on the situational context, and which are, if at all, influenced only to a small extent by the current goals of an agent. Furthermore, behaviour-based methods of execution are likely to be used by these agents. On the other hand, if the complexity of the application makes deliberate, plan-based agents desirable, the designer should put more emphasis on the goal context. Execution then should focus on mechanisms such as planning and, in the case of agent interaction, negotiation.

Chapter 4

The InteRRaP Agent Model

In this section, we explain the key ideas of the InteRRaP agent model and its basic functional structure. InteRRaP is a further development of the RATMAN agent model developed by Müller et al. [BM91]. The main idea of RATMAN was to describe the knowledge *rational agent* by a hierarchical knowledge base, and to use a general-purpose reasoning mechanism to define the knowledge processing in the agent model. InteRRaP extends this idea in two ways: on the one hand, it provides a clear separation among the control and the knowledge parts of the agent. On the other hand, the control mechanism provided by the InteRRaP model is much more general. Its main feature is that it allows to combine the use of patterns of behaviour with deliberate planning facilities. Therefore, InteRRaP does not force us to model *rational* agents *à la* RATMAN. One the one hand, patterns of behaviour allow an agent to react flexibly to its environment, and to perform routine tasks without having an explicit symbolic representation of how the task is to be performed. Due to the latter feature, we can characterize patterns of behaviour as abstract actions having procedural character, which makes them much more than what is generally understood by reactive, behaviour-based systems (see e.g. [Ste90]).

Planning, on the other hand, allows an agent to act in a goal-directed manner. Moreover, in a multi-agent context, planning is necessary to coordinate actions of agents. For instance, agents should be able to devise joint plans ([ZR91, Mül93]) to cope with special situations, or they should at least be able to exchange partial global plans (see [DL89]). In addition, since we consider communication in the light of speech act theory [Sea69] as planned communicative action, protocols for communication and negotiation should be represented in terms of plans.

4.1 The Overall Structure

Figure 4.1 shows the components of the InteRRaP agent model and their interplay. It consists of two basic parts: the hierarchical agent knowledge base, and the multi-stage control unit. The control unit is structured hierarchically in four sub-modules, the world interface, the behaviour-based component (BBC), the plan-based component (PBC), and the cooperation component (CC). The four control components exchange goals, plans, and information via communication. Section 4.2 contains a thorough description of the individual modules. Their interplay is discussed in section 4.4.

The agent knowledge base is also designed in a hierarchical manner. It consists of four components, namely the agent's *world model*, the *behavioural knowledge*, the *local planning knowledge*, and the *cooperation knowledge*. For a detailed discussion of the individual modules of the knowledge base we refer to section 4.3.

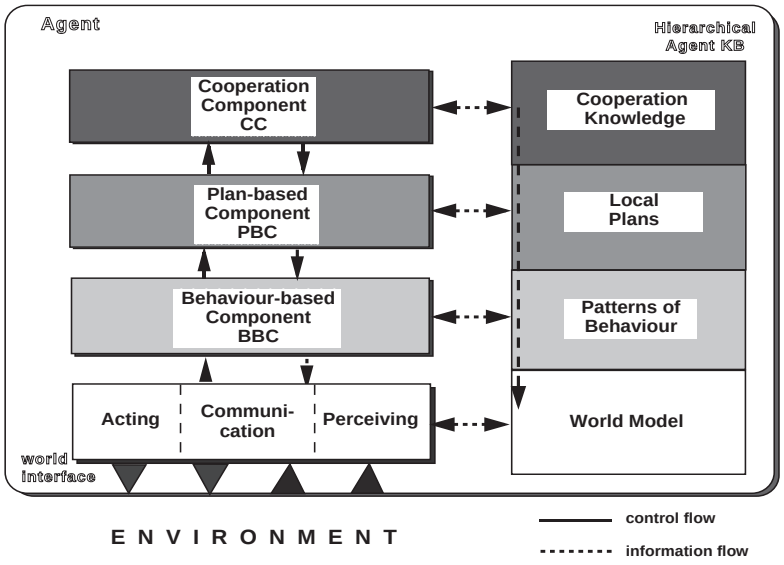

Figure 4.1: The InteRRaP Agent Model

4.2 The Agent Control Unit

In the following, the individual components of the agent control unit are described in more detail. The flow of control among these modules is defined in section 4.4.

4.2.1 The World Interface

The *world interface* bears the agent's facilities for perception, action, and communication. Its basic structure is displayed in figure 4.2. In the following, we will explain the single

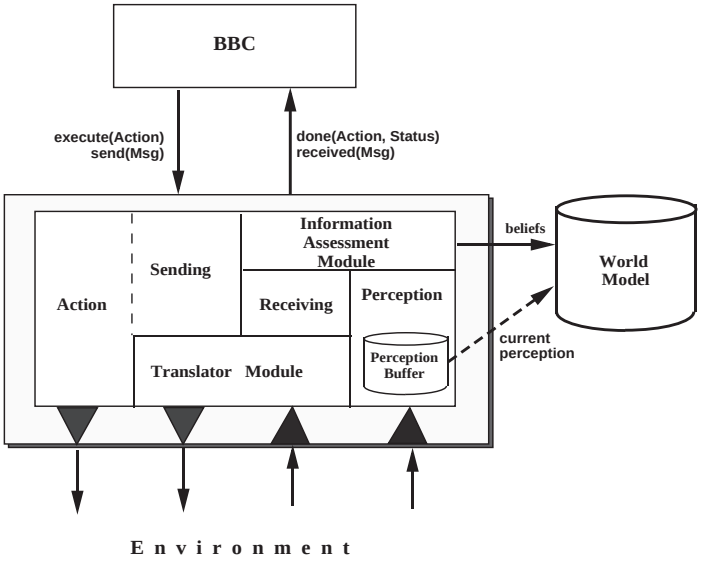

Figure 4.2: The InteRRaP World Interface

components of the world interface in more detail.

Performing Actions

The *actions* component controls the effectoric capabilities of the agent. Obviously, these capabilities are very domain-dependent. In the case of a robot, they are programs to control the arm movements, to control the speed and the direction of forward motion, and so on. In domains such as the transportation domain [KMM93b], where the agents do not manipulate the world in a physical sense, the *actions* component may be superfluous, at all.

Communication

The *communication* unit bears the agent's communicative facilities. It controls the physical realization of sending and receiving messages[1]. Since our communication concept is language-independent, outgoing messages have to be transformed from the agent's internal language into a knowledge interchange format (often called *interlingua*) which is understood by all agents. Corresponding to this, incoming messages must be transformed into the local agent language. This transformation is done by the *translator module*. In our current implementation, agents written in PROLOG, LISP, and MAGSY [FW92] are able to communicate with each other. A translator module for OZ agents [WHS93] is under development.

Perception and Sensing

The *perception* part of the world interface controls the vision and sensing facilities of the agent. Again the concrete implementation of this module heavily depends on what kinds of agents we want to model. In the case of a real robot environment, the perception part may include the transformation and processing of the data obtained by a video camera. Sensoric information can be received for example by laser or ultrasonic sensors. If we deal with a simulated scenarion, perception may be implemented by simulation, too; this may be done, for example, by the reception of messages from the simulated world model.

Information Assessment

In order to be useful in an agent's database, the information the agent receives or perceives must be transformed into an explicit representation, which is stored into the world model. Obtaining a high-level knowledge-based world model from low-level information has turned out to be a very difficult problem in AI since its early beginnings (see [Win84, RK91] for an overview). It implies all the problems of object and scene recognition computer vision is faced with, and for which no generally satisfying solutions have been achieved up to now. Information assessment also touches the issues of the attribution of credibility and of *belief revision*: an agent who maintains a world model that consists of what the agent believes of the world, and who receives a new piece of information from the world, has to whether, and in how much, it is willing to believe in the information. On the other hand, the incoming information may modify the agent's mental attitude towards its previous beliefs, and may force these to be revised.

In conclusion, the subject of information assessment is important and very difficult. Since we have not focussed on the issue of knowledge representation, up to now, we deal with this module in a quite straightforward way. Firstly, we assume the presence of a module which transforms low-level data in a symbolic representation of the world ("symbolic sensors"). Secondly, we assume that an agent believes all the information it receives - i.e. we model an uncriticial agent. The first assumption is reasonable since our applications - the loading

[1] In our implementation, it is realized via UNIX port socket communication according to the TCP/IP protocol.

dock, the transportation domain, and the COSMA schedule manager - only simulate real-world execution. Therefore, the messages received and the information perceived are already transmitted in a quite high-level format. The second assumption makes sense in our context because we are currently less interested in examining lying agents (as done for example by Zlotkin and Rosenschein [ZR91]) than in examining self-interested, but in principle honest agents[2], which can be classified as *non-antagonistic* agents.

4.2.2 The Behaviour-based Component

In the following, the behaviour-based component of INTERRAP is explained in more detail. The *BBC* implements the basic behaviour of the agent and the execution and decision component. It has access to a set of executable patterns of behaviour which are structured according to a two-dimensional hierarchy (stored in the hierarchical knowledge base). The position of a pattern in the hierarchy is an important factor to determine its priority (See chapter 5 for a more detailed discussion of priorities of patterns of behaviour and goals). The BBC is closely linked to the world interface, and thus, to the actions and changes in the world. Patterns of behaviour can be activated both by external trigger conditions and by the plan-based component.

The structure of the behaviour-based component is shown at figure 4.3. It consists of two

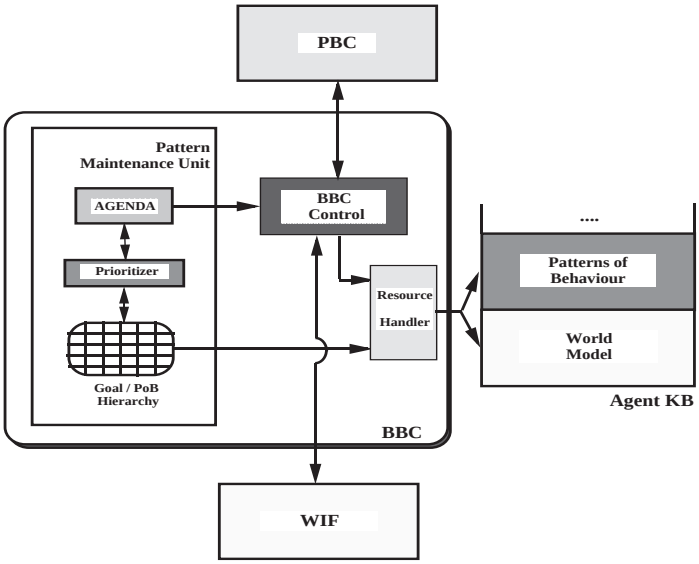

Figure 4.3: The Behaviour-Based Component

parts: a control component *bbc-control* and a set of patterns of behaviour. Both modules are described in the following.

Patterns of Behaviour (PoB)

In chapter 2, we defined the characteristic properties of patterns of behaviour. On the one hand, patterns of behaviour represent the basic reactive problem-solving facilities of the agent. Thus, there is a class of patterns of behaviour which are basically reactive. On the other hand, patterns of behaviour describe pieces of procedural knowledge [GL86], i.e. mechanisms which are not represented in a declarative manner, but which are basically

[2]Of course, the argument that designers might design their agents to lie, is right and of great importance for the design of negotiation protocols.

procedures. This kind of patterns of behaviour are abstract (pre-compiled) actions which can be activated by the planner. They are appropriate in order to activate *routine behaviours*, which do not require deep reflection or planning. Examples for such routine tasks are starting a car, or walking along a hallway, or moving straight ahead from one landmark to another.

Representation Whereas there has been done much work in the representation of plans, the question how to represent patterns of behaviour has been explored much less systematically for a couple of reasons: on the one hand, since it is one basic property of PoB that they have no explicitly represented internal structure,there seems to be no need for *the agent itself* to know about their representation. On the other hand, for designing PoB, and for allowing the planner to reason about PoBs as abstract actions, it seems very reasonable to think of representing them. Designing a pattern of behaviour requires a clear understanding of the activation conditions, the outcome of the execution, of termination and failure conditions of the pattern. Apart from this information, the planner needs to decide whether a certain PoB should be activated in order to reach a certain goal, or whether it should be preferred to generate a plan for the goal. In order to do this, heuristic information such as

> *If there is a pattern of behaviour available, always use it, because it tends to be more efficient than planning the goal in more detail!*

may be used. However, patterns of behaviour may be very complex so that it is often convenient to be able to evaluate the utility of their execution against alternatives *a priori*. Thus, the description of a pattern of behaviour should contain some information which allows such *a priori* evaluation of the consequences of executing the pattern. For example, using a weighted randomness behaviour in order to get from one landmark to another is certainly more flexible than planning the way down to a deeper level. However, it may also take the agent a longer time to reach its destination. In criticial applications, where hard time constraints have to be taken into consideration, it may be a hard requirement to plan more. Thus, an estimation of the time or the cost needed for executing a PoB, as well as an estimation of the probability of success (which can of course be improved while the agent is processing) may be valuable information for the plan-based component in order to assess how promising the use of a pattern of behaviour really is.

The BBC Control

The control component of the BBC has the task of coordinating the patterns of behaviour. Since the organization of the patterns provides the general possibility of concurrency, i.e. several patterns may be active in parallel, but only one pattern may be executed at a time, a coordinating instance is required. This corresponds to the organization of the conceptual agent model (see chapter 3) where the situational context and the mental context may activate a set of patterns of activity, one of which has to be chosen for execution by a decision instance.

What is actually required for this purpose is a priority mechanism for patterns of behaviour. Since patterns of behaviour correspond to goals of an agent, this affects the area of goal prioritization. Different strategies for attributing strategies to goals are described in chapter 5. The basic idea is to structure the patterns of behaviour into a two-dimensional hierarchy (see figure 5.2). One dimension of the hierarchy defines a *goal prioritization*. Lower level goals (i.e. patterns of behaviour for handling situation which might endanger the physical well-being of the agent) have a higher static priority than higher-level goals (for example cooperation with other agents The second dimension describes *goal expansion*. It reflects the actual state of processing of a certain goal, and is described in more detail in

chapter 5. Now, the priority of a goal (and thus, of the pattern of behaviour that is associated with the goal) is determined by combining a static priority (which is extracted form the hierarchy structure) with a dynamic part. The dynamic aspect corresponds to the fact that the relative importance of a goal may change over time. It is discussed in section 5.1.2.

In each cycle of the agent interpreter, the control mechanism selects one pattern (the one with the highest priority) and executes it. In the next subsection, we will look in more detail at the process of executing patterns of behaviour.

Execution and Control

According to the decision of the BBC control, a certain PoB is executed. The body of a pattern of behaviour is an executable (compiled) procedure which is called by the BBC control. In the body of the PoB there are activation calls to the *acting* and *perceiving* modules of the world interface. These are sent to the interface and result in the performance of actions in the physical environment or in the sending of messages to other agents. On the other, in the execution of a PoB, there can be a call to the PBC. In section 4.5, the interface between BBC and PBC is described in more detail. Chapter 11 shows the interplay among the modules by means of a concrete example.

An Example: The OPS-5 Way

In this section we will briefly outline and discuss a straightforward way to define patterns of behaviour and a control mechanism for these patterns by using the features of the rule-based programming language OPS-5 [For82]. OPS-5 is based on a forward-chaining rule interpreter using the RETE match algorithm which works on a working memory containing facts of the form

(name ^ $\texttt{attr}_1$ $\texttt{val}_1$ ... ^ $\texttt{attr}_n$ $\texttt{val}_n$).

Rules of the form

```
( p
    cond_1
      ...
    cond_k
-->
    action_1
      ...
    action_j )
```

can be defined by the programmer.

OPS-5, and its multi-agent extension MAGSY [FW92] which is used as the basis system in our group, has a couple of features which make it a suitable tool for designing patterns of behaviour:

- It supports concurrent activation of different rules. Several rules can be active at a time; in fact, each rule whose conditions are matched by the data in the working memory is included in the conflict set. A control mechanism chooses the rules that actually "fire".

- The algorithm used as the basic control structure of the OPS-5 rule interpreter is a straightforward and efficient implementation of a mechanism for goal prioritization. It was designed in order to select from the conflict set the rule with the highest priority. The prioritization mechanism is a dynamical one. It uses the criteria of

refraction, recency, and specificity in order to determine the rule with the currently highest priority.

- Using the MEA strategy [For82] which emphasizes the first condition of a rule, rules can be grouped to rule sets. Thus, a pattern of behaviour does not correspond to a single rule. Rathermore, it makes sense to implement it as a whole set of rules. The grouping is obtained by defining a working memory element *context* using the command

```
( literalize context
             name :  symbol[3]
             priority :  integer
             status :  symbol
                 ... ).
```

  The context element can be used in order to group rules in a very simple way. Using the MEA strategy, each rule of the rule set must have as the first condition

```
( context ^ name pob_xyz )
```

  This has the effect that this rule can only fire if the context with name `pob_xyz` is active. A context is activated by a

```
( make context ^ name pob_xyz )
```

  command in the action part of another rule.

- OPS-5 allows one to describe reactive behaviour in an intuitive and elegant manner by using the forward-chaining rule interpreting mechanism.

However, by our experience, there are also some problems with OPS-5.

- OPS-5 (with the exception of vector attributes) only allows the definition of flat data structures. There is no convenient way to represent objects whose description could best be covered by using a complex record structure. The description of a shipping company as an OPS-5 agent (see [KMM93a]) is an example for such an object.

- Whereas OPS-5 is very well suited for describing rule-based behaviour, it is totally inconvenient when it comes to model procedural knowledge, algorithms with a fixed sequential flow of control. We have solved this problem by integrating $C++$ functions into our OPS-5 code. However, a language which offers this kind of programming facilities from within it would be desirable.

- The data-driven structure of OPS-5 does not support the modeling of goal-directed behaviour.

- Except the hand-knitted context management which we mentioned above, OPS-5 offers no built-in way of structuring rules into rule sets, which forces the programmer to spend much programming effort and to be very careful when designing rule sets.

- In chapter 9, we will propose a hierarchical organization of patterns of behaviour which makes use of attribute inheritance. Of course, OPS-5 does not support modeling this kind of inheritance. Thus, the language is not well-suited as an abstract definition language for patterns of behaviour (see section 4.2.3). A language whose concepts

[3] The possibility of using type declarations for literals is a feature of MAGSY and is not part of standard OPS-5.

are based on the object-oriented principles, such as $C++$, $SMALLTALK$, or OZ [WHS93] should be preferred for this purpose.

The structural weakness of the language has been the reason why we started to reflect upon a more abstract language for the specification of patterns of behaviour. This issue will be discussed in the following subsection.

4.2.3 Towards a General Specification Language for Patterns of Behaviour

Above, we have motivated the necessity of developing a general language that allows one to describe patterns of behaviour on a high level, and, as a longer-term vision, to compile this abstract description down to LISP or OPS-5 code that can be executed on a physical machine.

In this section, we present a first approach towards such a more abstract description language. We start from a more informal structuring of patterns of behaviour according to some of the criteria already mentioned for general patterns of activity in chapter 3. We then outline how this description can be extended to a high-level specification language.

We define a pattern of behaviour by the following attributes:

- The *name* of the pattern, which serves as a reference for calling it.
- The static priority. This is one of $\{physical, task\text{-}oriented, social, optimization\}$. We refer to chapter chapter 5 for a more detailed discussion of this issue.
- A description of the participants. If the PoB is merely local, the only participant is `self` which stands for the agent itself. Note that in general, patterns of behaviour can also concern other agents.
- The *situational context* of the pattern. It describes the conditions which have to hold true in the external world (the environment of the agent) in order to be able to execute the pattern. Only patterns whose situational context matches the current external situation can become active.
- The *mental context* of the pattern. It defines the goals of the agents that are affected by the pattern, for example, what goals are supported by executing the pattern? Only patterns whose mental context matches the current goals of the agents can become active.
- A set of *postconditions*. These are conditions which hold after the PoB has been successfully executed. Postconditions can be used by a classical planner to reason about the suitability of a certain pattern of interaction.Note that - in contrast to the termination conditions defined below, postconditions may also be conditions which are not intended by the agent when executing the PoB.
- A set of *termination conditions*. These are conditions which enforce to stop the execution of the pattern of behaviour, because its goal is reached. For example, searching an object can be terminated when the agent has found the object. The pattern of behaviour `goto_landmark` can be terminated directly if the current position of the agent is equal to the goal landmark. Termination conditions are very useful since it is not trivial for a classical planner to realize that one of its current goals has been achieved by coincidence (for example by an action that has been performed by another agent). This problem contains the *frame problem* [MH90] - which is the problem of representing and computing which actions do not change when performing an action - and the *ramification problem* [Fin87] - which is the problem of representing and computing all

effects an action has. In practice, using termination conditions is a solution to these problems.

- A set of *failure conditions*. Basically for the same reasons as the ones mentioned above, it is useful to recognize when the execution of a PoB has failed, either because the goal corresponding to the pattern is regarded no longer achievable, because the agent does not pursue that goal any longer.

- Last, but not least, the actual execution part of the pattern has to be represented. It is a pointer to an executable piece of code (for example, a set of OPS-5 rules or a C++ procedure) together with the parameters necessary to instantiate the procedure call.

During the execution of the patterns, from time to time, the bbc-control checks whether the failure or termination conditions hold. If so, the execution is finished.

This informal description implies to define a pattern of behaviour as a frame-like structure as shown in figure 4.4. As a language for the description of the context and conditions, formula in First Order Predicate Logics, Horn Logics, or a terminological language can be used . At a later stage, a precompiler could be built which translates descriptions of patterns of behaviour generated in a frame language into lower-level code which can be executed.

```
( PoB
      :name <PoBName>
      :super <PoBName> ;; name of generalization
      :static <physical, task-oriented, social, optimization>
      :participants [:add] <names of participants>
      :sit_context [:add] <set of (first-order) formula>
      :mental_context [:add] <set of goal formula>
      :postcond [:add] <set of (first-order) formula>
      :termcond [:add] <set of (first-order) formula>
      :failcond [:add] <set of (first-order) formula>
      :exec_descr <proc_name> <param_1> ... <param_k> )
```

Figure 4.4: Abstract Description of a Pattern of Behaviour

However, we need to gain more practical experience before we can claim to have found a representation of the patterns of behaviour which covers all the important aspects of their description and execution. Experience in planning shows that there seems to be no universally accepted mechanism for representing abstract actions. New slots, such as evaluation information, interdependencies with other patterns of behaviour, incompatibilities with current commitments aso., could be integrated, leading to the definition of a very complex and expressive, but also possibly confusing language. In the examples at hand, we will use the formalism introduced above for modeling the patterns of behaviour.

4.2.4 The Plan-based Component

The *PBC* contains a planning mechanism which is able to devise local single-agent plans. The plans are hierarchical skeletal plans whose nodes may be either new subplans, or executable patterns of behaviour, or primitive actions. Thus, the plan-based component may activate patterns of behaviour in order to achieve certain goals.

Functionality of the PBC

Let us start with defining the external functionality of the component. The PBC shall be able to

- Devise a plan for a goal and control the correct execution of the plan. This is the "standard mode" of the planner when it is requested by a pattern of behaviour.
- Only devise, but not execute a plan for a goal. The plan structure is returned to the BBC. This functionality can be helpful when the agent is asked by another agent to tell it a plan for a certain goal.
- Evaluate a plan. This is required for two reasons. Firstly, as we will see, the *plan generator* component of the PBC may return a set of alternative plans. Thus, it is an internal functionality of the PBC to decide which plan to choose. For this purpose, however, expected utilities must be assigned to plans. Secondly, the PBC will produce joint plans. Since joint plans are negotiated between the partners, plans proposed by other agents have to be evaluated by an agent in order to decide whether to accept such a plan or whether to reject it. This is the external need for plan evaluation.
- Interpret a plan. If an agent receives a plan by another agent, it has to execute (interpret) this plan - without generating a plan itself. Therefore, the PBC must be able to interpret incoming plans and to control their execution.

This functionality leads to four different contents of messages which may be passed between the BBC and the PBC. They are shown in figure 4.5 and displayed in the overview table 4.10.

$BBC \rightarrow PBC$:

```
do(Goal)                  ;; make plan for Goal and control its execution.
plan(Goal)                ;; make plan for Plan.
eval(PlanList)            ;; compute expected local utility for Plans in Planlist.
interpret(Plan)           ;; control execution of Plan
retract(Goal)             ;; Goal is no longer current goal, plan obsolete.
done(PoB, Status)         ;; pattern of behaviour finished (Status ∈ {success, fail}).
```

$PBC \rightarrow BBC$:

```
done({Goal, Plan}, Status) ;; goal done/ plan interpreted (Status ∈ {success, fail}).
planned(Goal, Plan)        ;; returns plan for goal
evaled(Planlist, Eval)     ;; returns evaluation for plan.
activate(PoB)              ;; activation of a pattern of behaviour.
```

Figure 4.5: The Interface between BBC and PBC

Internal Structure of the PBC

In the following, the different subcomponents of the PBC and their interplay are explained. Figure 4.6 shows the internal structure of the plan-based component. It consists of a planning control module *PBC control*, of a *plan generator* module, a *plan evaluator* module, and a *resource handler* module. The individual components are explained in the following.

The PBC Control The *PBC control* is the "head" of the PBC. It consists of the PBC interface, the plan interpreter, and a set of goal stacks. The interface receives messages from and sends messages to the BBC. The plan interpreter controls the processing and the decomposition of a plan. For example, it transforms specific control structs of the

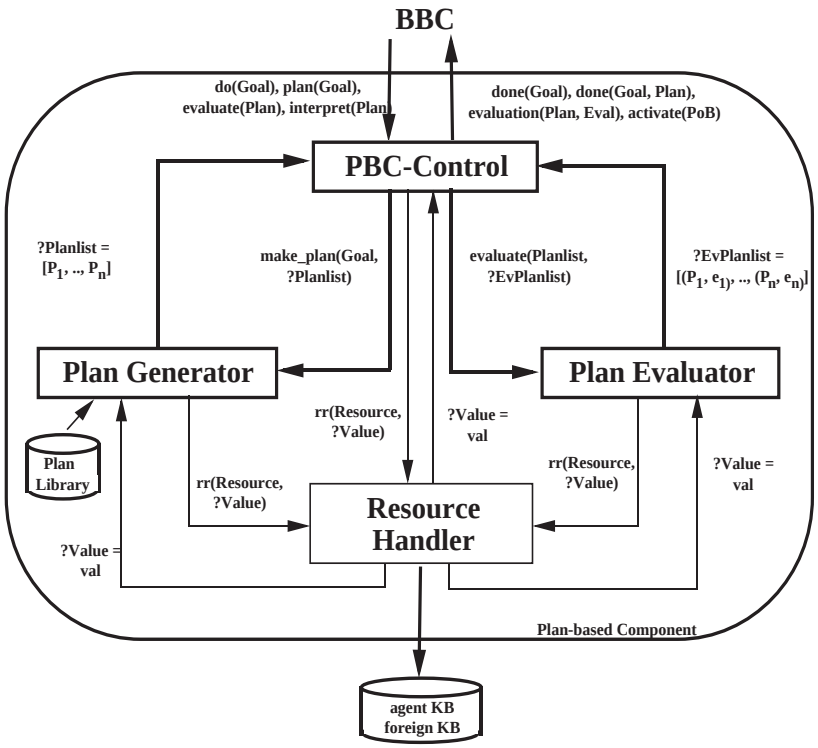

Figure 4.6: The Plan-Based Component

planning language (such as **while** e **do** a) in a format which can be understand by the BBC. Furthermore, based on the information brought about by the plan evaluator, it decides which goal to plan next. For this purpose, it maintains a set of goal stacks. This is necessary, because the planner may be called by several concurrent patterns of behaviour. Thus, for each goal, one goal stack is maintained for each goal. In each cycle, the interpreter chooses one of the goal stacks and processes the top goal of this stack. Processing a goal means either:

- to pass the goal to the plan generator, or
- to activate a pattern of behaviour.

The Plan Generator The *plan generator* has access both to a plan library and to a standard from-scratch planning algorithm (see e.g. [AKPT91]). The plan generator is called by the *pbc-control* by sending a command

```
make_plan(Goal, ?Planlist).
```

Generation of a plan means either choosing a suitable plan from the plan library, or, if such a plan is not available, devising a plan from scratch. The plan library consists of a set of entries

```
plan-lib ::=
    ( lpb-entry1,
  ...
      lpb-entryk ).
```

Each entry of the plan library is a tuple

```
lpb-entry(Goal, Type, Body) .
```

`Goal` is the reference name of the entry and specifies which goal (or rather: which plan step corresponding to a certain goal) is expanded by the specific entry. `Type` can be either `s` for *skeletal plan* or *b* for *executable pattern of behaviour*. For `Type = s`, the *Body* of the entry consists of a list of plan steps, which specify the expansion of the entry plan step. If `Type = b`, the Body denotes the name of an executable pattern of behaviour.

The Plan Evaluator The *plan evaluator* is able to associate utilities with plans. If it receives a list of alternative plans, it returns a list of evaluated plans. The evaluation is used by the plan interpreter in order to decide which of the alternative plans to pursue. Obviously, these utilities are estimations rather than exactly predictable values, because the real utility of a plan can often only be determined after it has been executed. Plan evaluation is a complex matter for itself, and quite a few syntactic (for example, the number of actions an agent has to execute) and semantic criteria (for example, estimating the cost implied by certain actions) can be defined and applied. In this paper, we will not deal with this matter in more detail.

The Resource Handler Finally, the *resource handler* module defines the interface to the resources an agent may need in order to devise its plans. Similar to [BS92], we feel that many actions of an agent can be expressed in terms of obtaining resources and obtaining resources from other agents / modules. An important resource needed by the planner is *knowledge*. For example, in order to determine the next plan step, an agent may have to know its current position - which is contained in the world model layer of the agent knowledge base. In general, answering the question where to find a certain resource is not a trivial problem. In the case of knowledge, which may be distributed within a multi-agent system, the problem can be considered as a problem of search in distributed data-bases.

Viewed under this perspective, the resource handler module serves as a monitor for database retrieval. It contains information which is necessary to decide, whether the information can be found in the agent knowledge base, or whether it has to be retrieved in an external knowledge base. This concept allows us to hide the information how to get from a certain resource from the *pbc-control* or the *plan generator*. Using the *resource request* call

```
rr(Resource, Value),
```

the resource handler can be commissioned to provide a certain resource, whose value is returned to the caller.

4.2.5 The Cooperation Component

The *CC* contains a mechanism for devising joint plans. It has access to protocols, a joint plan library, and knowledge about communication strategies stored in the cooperation knowledge layer of the KB. Figure 4.7 shows the structure of the cooperation component.

The basic parts the CC consists of are the *CC control*, the *joint plan generator*, the *joint plan evaluator*, the *joint plan translator*, and the *resource handler*. They are explained in more detail in the following.

The CC Control

The role of the control component of the cooperation layer can be compared with the role played by the PBC control in the plan-based layer. One the one hand, the CC control serves

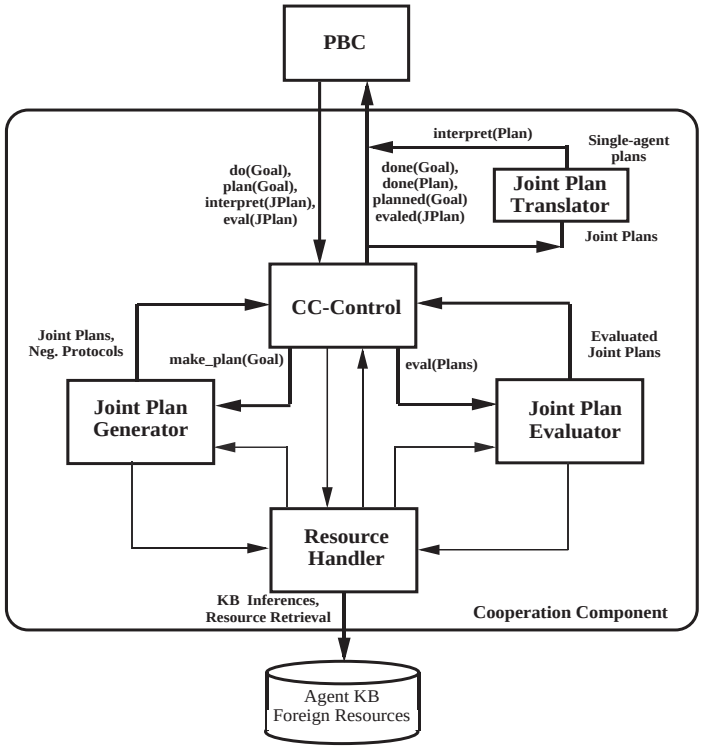

Figure 4.7: The Cooperation Component

as an interface between the CC and the PBC. On the other hand, it coordinates the work of the other submodules of the component. The two functionalities are described in the following.

Interface Functions From the perspective of the plan-based component, the cooperation component offers four basic functionalities.

- It can devise a plan for a goal which is passed to it by the PBC. This functionality can be accessed by a message `plan(Goal)`. This facility corresponds to the one also contained in the PBC. However, at the CC level, plans can be devised which describe activities of more than one agent. In the case of a `plan(Goal)` call, the cooperative plan is passed directly to the PBC, without any further translation or interpretation. This functionality is needed if the agent is asked by another agent for a plan whose execution will achieve a certain goal.

- Using the the `do(Goal)` call, the CC may plan a goal and control its execution. This also corresponds to the process activated in the PBC by the `do(Goal)` call.

- The cooperation component shall be able to interpret a given cooperative plan. This is necessary in the case that the agent has received a joint plan by another agent. The interpretation functionality can be activated by the PBC sending a `interpret(JPlan)` message to the CC. `JPlan` stands here for a cooperative or joint plan.

- The cooperation component must be able to evaluate a cooperative plan in the same way as the plan-based component evaluates a single-agent plan. This is important since joint plans can be subject to negotiation, and since an agent has to decide whether it shall accept or reject a plan which has been proposed by another agent (see below).

The evaluation functionality is activated by the plan-based component by sending a message `eval(JPlan)` to the CC. It is implemented in the *joint plan evaluator* module.

The functionalies offered by the PBC and the CC are displayed in the overview table 4.5.

Coordination Function The CC control obtains the above mentioned requests from the PBC and monitors their execution. For this purpose, it coordinates the activities of the other submodules of the CC, namely the *joint plan generator*, the *joint plan evaluator*, the *resource handler*, and the *joint plan translator*. When the CC control receives a message `do(Goal)`, it orders the plan generator to make a plan for this goal. The plan generator returns (in the case of success) a list of plans for the respective goal. The CC control passes this list to the plan evaluator, where the utilities of the single plans are computed (see the discussion below). If information or additional resources are required during the computation process, these are provided by the resource handling module. The plan evaluator returns an evaluated list of joint plans for the goal. The CC control selects the best plan and interprets it. Interpretation here means the monitoring of the execution of the plan. Since the plans generated by the CC are plans for multiple agents, they cannot be directly interpreted by the plan-based component. Rather, the joint plan is transformed into a single-agent plan by the joint plan translator. The output of the plan translator is a sequence of single-agent plan steps (which can be again plans or patterns of behaviour) which can be interpreted by the plan interpreter of the PBC as described in section 4.2.4.

In the following, the individual modules which are coordinated by the CC control are described.

The Joint Plan Generator

Similar to the generator of the PBC, the joint plan generator is responsible for devising a joint plan for an interactive situation. It does this based on the goals of the agent itself, and, if these are available, based on the goals of the other agent(s) participating in the interaction. It has two alternatives to devise a plan. For a couple of standard situations, ther are predefined joint plans in the joint plan library. A subset of the joint plans in the library are negotiation protocols. For other cases, a plan has to be devised from scratch, or at least has to be specified so far that predefined plans or protocols can be used as subplans.

The Joint Plan Evaluator

Since joint plans are subject to negotiation, the agent must be able to evaluate a joint plan which has been proposed to it by another agent. On the other hand, in order to generate "reasonable" joint plans itself, the agent must have a measure for what is a reasonable plan. It is the task of the joint plan evaluator to determine whether a plan is reasonable by attributing a utility for the plan. The evaluator accepts as input a list

$$[P_1, \ldots, P_k]$$

of joint plans proposed for achieving a goal, and outputs a list of evaluated plans

$$[(P_1, e_1), \ldots, (P_k, e_k)]$$

where e_i is the utility ascribed to P_i. The implementation of the evaluation function depends on the representation of the plans. As we already mentioned for the case of single agent plans, the (a priori) evaluation of plans is very difficult to do, especially in the case of hierarchical, non-linear plans, where it is not known at planning time what effects the plan will have, and how expensive its execution will be. In the case of multi-agent plans, there is an additional dimension: What should the utility be an agent ascribes to a joint plan? Should it be the

estimated local utility for the agent itself? Should it be the global utility? Or should even the utility of the other agents be taken into consideration? The effects different evaluation strategies may have on the negotiation behaviour of the agents is discussed in the following.

Evaluation Strategies In the simplest case, agents only consider the utility a joint plan has for themselves. In game theory, this behaviour is called "individual rational" (see e.g. [LR57, Ros85]. A characteristics of negotiations among individual rational agents is that the solutions found are - in the best case - pareto-optimal solutions[3].

The second alternative, using the global utility of a plan in order to evaluate it is a fine idea, since it obviously leads to good global solutions. Unfortunately, in most interesting DAI domains, either the agents are not able to compute the global utility due to their incomplet knowledge, or the agent are not really interested in finding a globally optimal solution, but in finding a solution which maximizes their local utility.

The third alternative, trying to maximize one's own utility but taking into consideration the utility a joint plan has for the other agents, is a very interesting one for a couple of reasons. Firstly, an agent who generates proposals for joint plans can considerably shorten the time needed for negotiation if it tries to propose plans which are acceptable for both agents. Therefore, it seems worth at least trying to estimate whether a plan could be acceptable for the partners. Secondly, if we introduce long-term criteria for decision-making in negotiation, an agent will probably take it badly if it is continuously faced with inacceptable proposals devised by its negotiation partner. Depending on the negotiation strategy it uses, this agent might also start to make proposals which are inacceptable for the other agent, as well[4].

In conclusion, taking into consideration the utility a a plan has for another agent seems to be stringent if reasonable negotiation behaviour shall be modelled.

The Joint Plan Translator

Depending on the representation of a plan, it is often not possible to use the same language for describing (multi-agent) plans and protocols at the cooperation layer, and for describing their local counterparts, the subplans which are executed by the plan-based component. For example, if two agents i and j who are facing each other directly devise a joint plan fro changing positions, a joint plan may be that i should move one field to the right, then one field ahead, and then one field to the left, whereas agent j moves one field ahead (see chapter 11 for a more detailed treatment of this example, as well as figure 10.1 for a graphical display of the scene). However, it is not sufficient to give the plan-based component of agent i only the i-projection of the joint plan, namely "move to the right, move ahead, move to the left". The joint plan contains also the plan constraints, i.e. the interdependence of plan steps performed by different agents. In our example, agent j may only start performing its plan after agent i has moved to the right. On the other hand, i must wait before it can move again to the left until j has performed its action *walk_ahead*.

It is the task of the joint plan translator to transform a joint plan into a single-agent plan by projecting the agent's part of the joint plan and by adding plan steps which guarantee that the constraints contained in the original joint plan are satisfied during plan execution. The organization of the translator depends on the way joint plans and single agent plans are represented. In chapter 11, we will show the functionality of the tranlator module by means of the loading dock example.

[3] A solution S for a problem P among n agents is pareto-optimal if there is no solution S' for P so that not at least one agent has a worse local utility for S' than it has for S.

[4] This would be the case if the agent used the winning strategy in the prisoners' dilemma [Axe84] tournament, namely to cooperate as long as the partner cooperates and to defect as soon as the partner does.

The Resource Handler

The role of the resource handler is the same as already discussed in the case of the plan-based component in section 4.2.4. According to the principles of modularity and *information hiding*, the modules of the cooperation component should not have to care where to get a certain resource or a certain piece of information from: it is the task of the resource handler to administer the local resources of an agent, and to look up for a certain piece of knowledge in the appropriate part of the agents knowledge base. If a certain resource or information cannot be obtained locally, but the resource handler has information about how and from whom the resource information may be gained, the CC control can use this information in order to obtain the resource via an appropriate protocol.

4.3 The Hierarchical Knowledge Base

The *knowledge base* is structured in a hierarchical manner as it has been proposed in the RATMAN agent model [BM91]. It consists of four layers which basically correspond to the structure of the agent control, namely the world model layer, the behavioural knowledge layer, the planning knowledge layer, and the cooperation knowledge layer. The individual layers are discussed in the following subsections.

4.3.1 The World Model

The lowest layer contains the world model of the agent. It contains what the agent believes to be the current state of the static and dynamic world. The knowledge[5] in the world model is organized according to a taxonomy of classes. Since our classification is rather functional, and therefore does not draw a distinction between object-level and meta-level knowledge, the world model layer contains both object-level knowledge and meta-level knowledge, i.e. both factual knowledge about the state of the world and epistemic and auto-epistemic knowledge. For example, the knowledge a transportation agency has about its own capacities and capabilities is represented using the taxonomical knowledge representation system KRIS [BH90].

4.3.2 Behavioural Knowledge

Layer two defines the primitive actions and the patterns of behaviour. Moreover, it contains control knowledge which is used by the BBC-Control in order to maintain the patterns of behaviour. According to the RATMAN agent model [BM91], primitive actions are represented as precondition-action-postcondition triples. The representation of patterns of behaviour has been discussed in detail in section 4.2.2. It ranges from simple condition-action pairs (for simple reactive patterns of behaviour) to a description in an abstract specification language.

4.3.3 Local Planning Knowledge

Layer three contains local plans and knowledge which is specific for planning. The representation of plans has been discussed in section 4.2.4 and will be shown in more detail by means of an example in chapter 11. In our applications, plans are stored as hierarchical skeletal plans [BKL92] in a plan library. Thus, the planning component can find our whether

[5] In the following, we will use both the terms *knowledge* and *belief* in order to refer to the information kept in the world model. Note, however, that in general all the information an agent has is unsecure. Therefore, it is more adequate to speak of the beliefs of an agent instead of its knowledge. However, for certain pieces of information, for example for "facts" the agent actually perceives, or for attributes of an agent such as the current position of a forklift, it is desirable and also acceptable to say that the agent *knows* them.

there is a skeletal plan for the current goal in the plan library. Plans are represented as tree structures whose leaf nodes contain only patterns of behaviour (abstract actions) and primitive actions which may be directly executed using the primitives offered by the world interface.

4.3.4 Cooperation Knowledge

Finally, layer four contains knowledge of and strategies for cooperation. This knowledge comprises *joint plans* for coordinating the actions of multiple agents. The plans are stored in a joint plan library and accessed similar to the way single-agent skeletal plans are accessed in the plan library. In addition, negotiation protocols are stored in the cooperation knowledge module. These protocols are represented as tree structures whose leafs are plans which can be executed by the plan-based component. Moreover, the decision layer of the negotiation model is described here by means of different negotiation strategies among which the designer of the system - or, if possible, the agent itself - can choose (compare [SS93] for a description of possible strategies for the case of appointment scheduling negotiation).

Finally, for different patterns of cooperation, such as blocking conflict resolution (in the loading dock), exchanging orders, offering empty rides (transportation domains), cooperation-specific partner information is stored. Thus, it is described whether another agent is considered a good partner for a specific kind of interaction. This is done based on the cooperation history, i.e. based on the experiences the agent has gained on the occasion of earlier encounters with other agents.

4.3.5 Inter-layer Relationships

There are two kinds of relationships between the individual layers of the hierarchical agent knowledge base. Firstly, information may be passed from layer to layer. The basic idea is that information contained in lower layers is "visible" for the higher layers, but not vice versa. For example, the plan-based component can access information about the world model, whereas the behaviour-based component does not have access to planning or cooperation information. Note that this enables us to describe negotiation protocols explicitly (by a plan) and implicitly, from a local point of view, by a pattern of behaviour. For example, the contractor or manager roles in a contract net protocol can be defined in a straightforward manner by patterns of behaviour.

Secondly, higher-level data structures are constructed in terms of lower level ones. Figure 4.8 shows how this works. On top, cooperation knowledge is defined by joint plans and negotiation protocols. These protocols are represented by graphs (in special cases also by trees), whose inner nodes are decision nodes, and whose leaf nodes describe parts of the protocol which are local plans. For example, leaf nodes of the contractor part of a *contract net* protocol [DS83] are subplans for computing and sending a bid for an offer, for executing the order if it has been granted to the agent, and for reporting the state of affairs to the manager after the execution has been finished. These subplans are defined one layer below, namely at the planning knowledge layer of the knowledge base. Analogously to joint plans and negotiation protocols, local plans are defined as tree structures whose inner nodes denote decision and expansion (for example labeled as AND and OR nodes), depending on the plan language), and whose leaf nodes are executable patterns of behaviour. The patterns of behaviour are again defined one layer below, at the behavioural layer of the agent knowledge base.

Note that, in figure 4.8, the patterns of behaviour are represented as structured boxes. This is to denote two essential characteristics of patterns of behaviour, which distinguish them from plans: one the one hand, patterns of behaviour are regarded as "black boxes", from the point of view of the planner. They represent procedures which are activated and

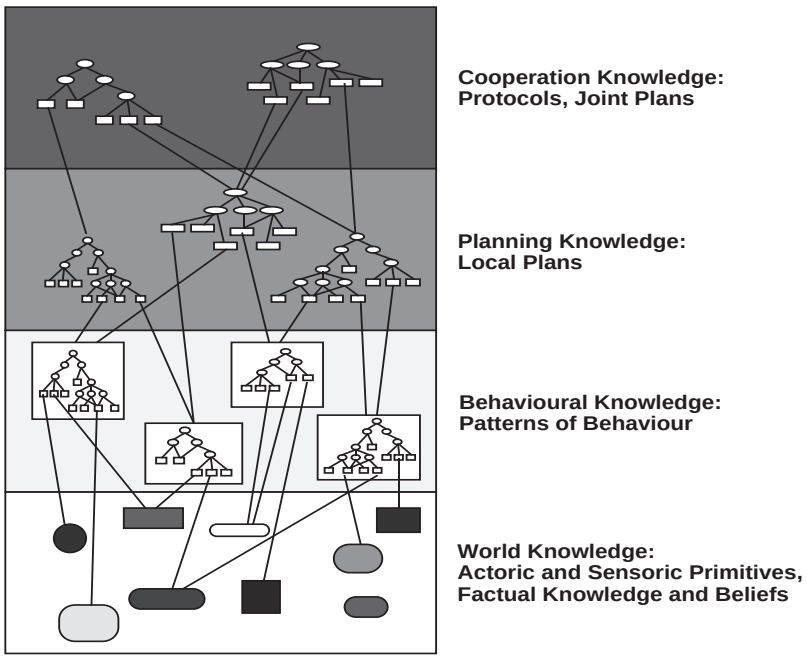

Figure 4.8: Structure of the Hierarchical Knowledge Base

executed without reasoning about their internal structure. On the other hand, patterns of behaviour are compiled multi-plans whose external effects are situated: in different situations, one and the same pattern may produce different behaviour - which can be explained as taking different paths through the compiled tree-structure, caused by situative decisions made at several internal nodes in the behaviour[6]. Moreover, if we view patterns of behaviour from the perspective of the layer below them, the world model, their output is the activation of primitive actoric and communicative acts - which are defined in the layer below. Therefore, the relationship between PoBs and plans is similar to the relationship between the data structures at the world model layer and the patterns of behaviour.

Finally, there is the question whether there is a flow of information amon the levels of the knowledge base in the same way as it happens in the control unit, and if so, how is this flow of information defined? Up to now, in our system, there is no information passed explicitly between different layers of the KB. If the KB is changed, then this is done by an appropriate layer on the side of the control component, for example by a pattern of behaviour which observes the world and changes a knowledge base entry at the behavioural knowledge level if a certain situation in the world occurs.

A slightly different proposal would be to organize the knowledge base as a hierarchical blackboard structure. For each layer of the structure, administration demons (similar to the knowledge sources in standard blackboard systems [Len75]) are defined. Each demon is able to transform the knowledge at its layer based upon information it receives from lower layers.

Thus, similar to the way this happens in the control unit, very specific information contained in the lower layers of the knowledge base could be transformed into more general and more abstract information at lower layers of the knowledge base. This way, the functional hierarchy provided by the INTERRAP agent KB could be enhanced by an additional di-

[6]From the point of view of the design of patterns of behaviour for a concrete domain, it is clear that the structure of the patterns must be designed - just as any (good) program should have a clear structure. This is a further motivation for developing an abstract specification language for PoB.

mension, the dimension of specificity and abstraction. This, however, remains to be future work.

4.3.6 Coupling Agent Control and Knowledge Base

Up to this stage, we have described the two basic parts of an INTERRAP agent, namely its control module and its knowledge base. Of course, the knowledge contained in the knowledge base is used by the modules of the control unit in order to make decisions and in order to determine how the agent should behave in a given situation. Since both the knowledge base and the contol unit are hierarchically structured, it is a nontrivial question how the interaction between the two parts are defined, i.e.

- Which control modules have access to which parts of the database?
- If a control module can use information from more than one part of the knowledge base, how does it know where is should search for certain pieces of information?

In the following, we provide brief answers to these questions.

Knowledge Base Access

The question which control modules may retrieve inormation from which layers of the knowledge base is answered by using a simple mechanism: a control layer has access to its corresponding knowledge layer, and to knowledge contained in layers *below* this layer. Thus, the world interface has access only to the world model, the BBC has access to the world model and to the behavioural knowledge, and so on.

Information Retrieval

The knowledge in the agent KB is divided in four parts. Therefore, if the agent has to look up for a certain piece of information in the database, the question arises of in which part of the knowledge base this information should be retrieved.

The way we are solving this problem is by providing a resource handler module for each of the control modules. The resource handlers have access to the knowledge base interface and put queries to the knowledge base. The structured design of the knowledge base allows one to use indexing techniques which greatly enhance the access to certain information kept at some specific layer of the knowledge base. Knowledge is indexed acording to its content. The beliefs of an agent about the world are kept in the world model part, local plans are kept in the planning knowledge part, joint plans and protocols are kept in the cooperation knowledge part of the agent knowledge base. Thus, in many cases the appropriate layer can be chosen according to the characteristics of the information to be retrieved.

In order to yield intelligent and flexible behaviour, the different modules of the INTERRAP agent have to be coupled by defining an appropriate control mechanism. In the following section, we provide such a mechanism.

4.4 The Flow of Control

The flow of control in INTERRAP is determined by two directions, which are described in the following.

4.4.1 Bottom-up Control

Firstly, there is a bottom-up direction which describes how the agent processes information which is coming in through the world interface, i.e. changes in the world which the agent perceives, or messages sent by another agent. Per default, this information is handled by the behaviour-based component. There, immediate reactions on the new situation may be triggered. However, if the situation is too complex, because the task to be solved requires more sophisticated reasoning, or because a highly constrained conflict is to be resolved, the control is passed to the plan-based component. We call this kind of shift of control *competence-driven*, since module n is able to recognize its own limitations, and passes on the actual task to module $n+1$[7]. If the agent can tackle the situation by a local plan, the bottom up flow of control stops here. Otherwise,if the task requires explicit coordination with other agents, control is passed on to the CC. There, the actual problem is solved using a joint planner, and several negotiation protocols which may be used in order to retrieve information from other agents, or in order to coordinate common activities.

4.4.2 Top-Down Control

After the incoming information about a situation has diffused from the lower levels of control to the higher ones, and after a way of tackling the situation has been worked out by the appropriate control layer, the solution diffuses back top-down through the control hierarchy. Joint plans devised by the CC are translated into a single-agent plan enhanced by synchronization commands, negotiation protocols are decomposed into local partial protocols which are basically single-agent plans. These plans are passed to the PBC where they are interpreted. As we have seen in section 4.2.4, single-agent plans are structured as trees whose leaf nodes correspond to executable patterns of behaviour. Once the plan interpreter has reached such a leaf node, it activates the corresponding pattern of behaviour. Thus, control is shifted to the behaviour-based component. Finally, the pattern of behaviour directly triggers the primitive effectoric and communcative facilities which are provided by the world interface, and which are directly transformed into the performance of actions and the sending of messages in the physical world.

4.4.3 Integration

As we have seen, the flow of control can be described by two phases, a *bottom-up* phase, where information and control flow from lower layers up to higher layers in the hierarchy, and by a *top-down* phase, where they flow back down from the higher layers to the lower layers where they result in the execution of primitive actions or the sending of messages to other agents.

Note that this flow of control does not always have to occur throughout all the layers of the agent. Rather, caused by the competence-driven control mechanism, short-cuts are possible in many cases. Figure 4.9 shows the different possible flows of control in the INTERRAP model. The thickness of the arrows corresponds to the frequency control follows the respective flow. The behaviour-based level can cope with a considerable number of situation by activating appropriate patterns of behaviour. For a smaller number of situations, more sophisticated reasoning or more goal-directed activities are required, which are provided by the PBC. Finally, for a couple of highly constrained, interactive situations, the cooperation component has to be activated.

[7]This philosophy requires that the individual layers are able to estimate whether they are capable of performing a certain task. Whereas this is a severe problem in general, it seems to be a reasonable assumption when we consider a special application.

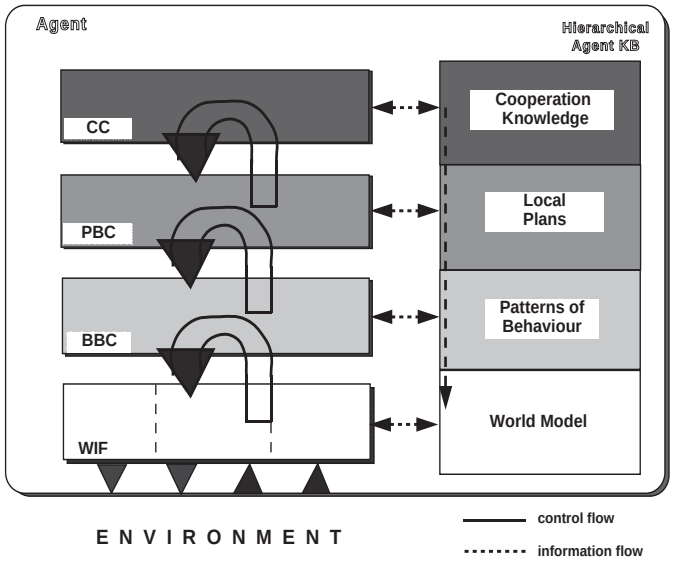

Figure 4.9: The Flow of Control

So far, we have described the general flow of control in the INTERRAP model. In the following, we will define this flow of control in a more precise manner. We define the interface among the moduls of the control unit as a communication structure, according to the functionalities of the different modules of the agent control defined in sections 4.2.1, 4.2.2, 4.2.4, and 4.2.5, and according to the general control model we have presented in this section.

4.5 The Communication Structure

As we have described above, the control unit of an INTERRAP agent consists of a set of modules among which control is shifted and which can access the hierarchical agent KB. In this section, the interplay among these modules is described by outlining the internal *communication structure* of the agent. The basic idea is to treat inter-module communication basically the same way as inter-agent communication, namely by defining a set of communication primitives denoting the intended semantics of messages [Sea69, BM92, LBS92].

4.5.1 Message Formats

The format of a message among two modules *Sdr* and *Rcp* is

```
msg = (ID, Ref, Sdr, Rcp, Type, Content).
```

The control modules communicate by sending and receiving messages. Examples for basic message types are *accept, command, demand, inform, modify, propose, refine, reject, request.* Complex protocols can be constructed starting from these communication primitives.

4.5.2 Contents of Messages

The content of messages sent among modules reflects the functionality of the modules. Table 4.10 specifies the an overview of the possible contents of messages which are exchanged among the modules of agent control. Most of the slots defining the interface among the modules have been discussed in the respective subsections in this chapter. Therefore, we

	World Interface	BBC	PBC	CC
World Interf.		done(Action, Status) done(MSG, Status) received(MSG)		
BBC	execute(Action) send(MSG)		do(Goal) plan(Goal) eval(PlanList) retract(Goal) interpret(Plan)) done(PoB, Status)	
PBC		activate(PoB) done({Goal, Plan}, Status) planned(Goal, Plan) evaled(PList, Eval)		do(Goal) plan(Goal) eval(JPList) interpret(JPlan) done(Plan, Status) retract(Goal)
CC			interpret(Plan) done({Goal, Plan}, Status) planned(Goal, JPlan) evaled(JPList, Eval)	

Figure 4.10: Communication Interface of the Control Modules

will not provide a more detailed explanation of the table in this place. Note, however, that we use basically the same speech-act oriented communication model for intra-agent communication as the one employed for communication between different agents.

Chapter 5

Decision-Making

In this section, we will go into more detail in how agents draw the decisions as to which goals to pursue in a certain situation. Thus, we will take a closer look at the "decision" box in figure 3.

An agent is characterized by the set of goals it can possibly have and by the mechanisms it possesses in oder to achieve these goals. The goals are structured in a two-dimensional goal hierarchy. This goal hierarchy is explicitly represented and can be accessed by the decision component within the decider. The first direction of the goal hierarchy is the dimension of *goal prioritization*. It incorporates a kind of subsumption architecture [Bro86]. The second dimension is the dimension of goal refinement or *goal expansion*. It describes how goals split up into subgoals and patterns of behaviour. This dimension is characterized by a plan language which allows to build a plan consisting of subplans, executable patterns of behaviour, and primitive actions. Such a plan language has been discussed in chapter 4.2.4 and ists use is shown exemplariliy in chapter 11.

5.1 Goal Prioritization

As we have seen in section 3, an agent may have a bulk of possible goals with respect to its current situational and mental context. These goals are called *active goals*[1]. However, since we assume that an agent is a sequential machine, it can only pursue a single goal at a time. Therefore, a crucial decision an agent has to make is which goal to pursue next.

This decision is obtained using a mechanism of goal prioritization. With each active goal, a priority is associated using a priority function $P : \mathcal{G} \rightarrow \mathcal{N}$, where $\mathcal{G}$ is the set of goals, $\mathcal{N}$ is the set of integers.. The goal with the highest priority is pursued in the next cycle. The priority of goals is computed by taking into consideration two components: a static component, and a dynamic component. The latter component respects the fact that the priority of a goal depends on the current situation (situational context) and on the set of currently active goals of an agent (mental context).

5.1.1 Static Priority

In order to determine the static prioritization of a goal, we use a model which has been proposed by the psychologist Maslow (in [Gor77], pg.33f): Maslow has classified the importance of human needs in a pyramid which has five levels. The Maslow pyramid is shown in figure 5.1. Maslow claims that the lower-level needs have higher priority than the higher-level needs, and that the former must be satisfied before a human starts satisfying the latter. For example, if you are very hungry, you will do anything (for example shoot a buffalo) in order

[1] Rather, we should speak about active patterns which create goals. However, in our model, we assume that there is a one-to-one mapping between patterns and (top-level) goals (see figure 5.2).

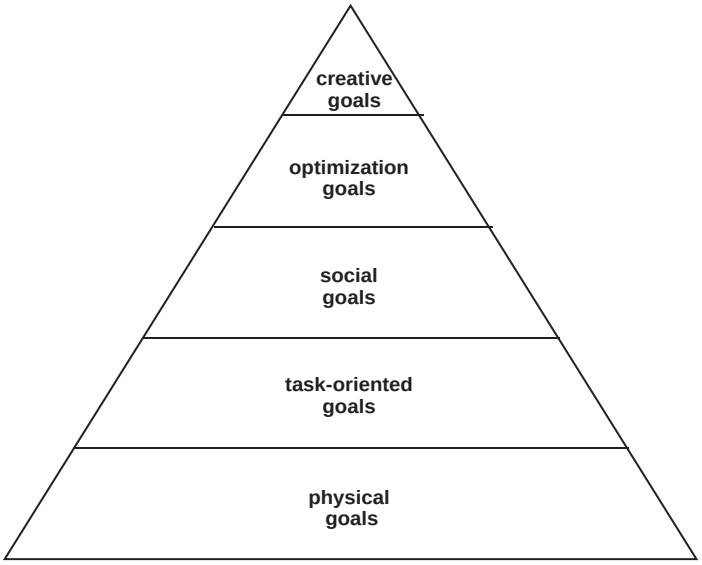

Figure 5.1: The Maslow Pyramid of Human Needs

to satisfy this physical need. Especially, you will "overwrite" your need for security and for performing your everyday tasks. On the other hand, you will only satisfy your social needs, for example help other people, after your physical and security-relevant needs are satisfied (for example after you have eaten parts of the buffalo and stored the rest of the meat for future meals).

We applied this model to the goals of an agent. The basic needs of agents can be regarded as the basic (top-level) goals (sometimes, the notion of intention is used for this, too, e.g. [BS92]). We call level four the level of *optimization*, since at this stage, an agent has to reflect on its own behaviour, and on how it can be optimized. We are very careful with level five, since we feel that self-realization and creativity are not even subjects for current research in AI. Therefore, we refrain from instantiating this level in our model.

For example, the mapping of the goals of an agent to the levels of the Maslow pyramid is provided by the vertical dimension of figure 5.2. Physical goals are to keep physically unhurt by avoiding collisions. Security-relevant goals are to perform the tasks given to the agent, and to resolve situations of conflict with other agents. Social goals are the goals of helping other agents and of passing information, and optimization goals are e.g. the refinement of existing plans, and the exploring of an unknown environment.

Lower-level goals are of higher priority than higher-level goals. Thus, an agent will explore interesting things - but it will do only if there is no current transportation task to be performed. This kind of static prioritization is often useful and wanted. However, there are two problems with this kind of goal selection:

- Often, a higher-level goal is confliging with a lower-level goal. In these cases, the higher-level goal should be able to *inhibit* the lower-level goal from being pursued.
- Consider figure 5.2: if we assume that a forklift is currently busy with performing a task "`load truck`", according to the static priorities of the goals, it will never engage in any cooperation. This, however, is certain not what we want. Here, a more flexible mechanism of prioritization is required.

We see that there are two kinds of relationships among different layers in a goal hierarchy: firstly, lower-level goals must be satisfied before higher-level goals are pursued. Secondly,

higher-level goals can have higher priorities than lower-level goals and can even suppress these. The former effect is covered by our Maslow pyramid. For the latter type of relationship, Brooks [Bro91] provides two mechanisms he calls *suppression* and *inhibition*. In our context, inhibition means that a higher-level goal can totally eliminate a lower-level goal. Suppression can be regarded a concept similar to what we mean by dynamic prioritization. We will discuss this in more detail in the following subsection.

5.1.2 Dynamic Priorities

At this stage, we enhance our static model for goal prioritization by a dynamic aspect. What we want to obtain is a model according to which an agent pursues goals in the order of their *relative importance*. The principle idea is that the relative importance of a goal is expressed in terms of a "degree of satisfaction": if an agent adopts a new goal, the degree of satisfaction of this goal is instantiated. Depending on time constraints, the degree of satisfaction decreases over time - the importance of the goal grows. If the agent is working on the goal, the degree of goal satisfaction increases - the relative importance decreases.

Reasonable decisions arise from combining the static priority described above with the dynamic mechanism:

Definition 2 (Priority Function) *Let $\mathcal{G}$ be the set of goals an agent may have, $G \in \mathcal{G}$. The priority function $f : \mathcal{G} \rightarrow \mathcal{N}$ is defined as $f(G) = f_{stat}(G) + f_{dyn}(G)$, where $f_{stat} : \mathcal{G} \rightarrow \mathcal{N}$ is the static part, $f_{dyn} : \mathcal{G} \rightarrow \mathcal{N}$ is the dynamic part of the priority function.*

Thus, in general, lower-level goals are more likely to be chosen than higher-level goals. However, if a higher-level goal has a high relative importance, it may be preferred to a lower-level goal.

Unlike for static prioritization, we do not provide a formal model for the dynamic prioritization, since we feel that the criteria are highly domain- dependent. Rather, we propose that the dynamic component is computed by applying heuristic criteria, such as time constraints on goals, availability and scarcity of resources, or the current stage of processing of a goal.

5.2 The INTERRAP Goal Hierarchy: an Example

Figure 5.2 shows in an exemplary manner the two-dimensional goal hierarchy for a concrete example, namely the description of a forklift in the loading dock. The vertical dimension is organized according to the different layers of the Maslow pyramid. The horizontal layer represents the goal expansion hierarchy. It incorporates a plan language by means of which the top-level goals are split in subgoals. The execution mechanism can use the goal expansion information in order to assign appropriate execution methods to a goal. In figure 5.2, a simple plan language is used which allows conjunction as the only way to build more complex plans from simple plans and patterns of behaviour. The goals at level 0 correspond to the basic patterns of action and interaction an agent may have. At levels i, $i \geq 1$, the basic goals are more and more expanded according to the plan language. Note that each node in the expansion tree corresponds to a certain stage of execution of the respective top-level goal. Therefore, as long as we have a plan language which does not allow disjunctive constructs, inside the expansion of a single top level goal, at most one goal pursued at a point in time. Thus, inside one layer of the Maslow pyramid, decisions have to be made only between alternatives at level 0.

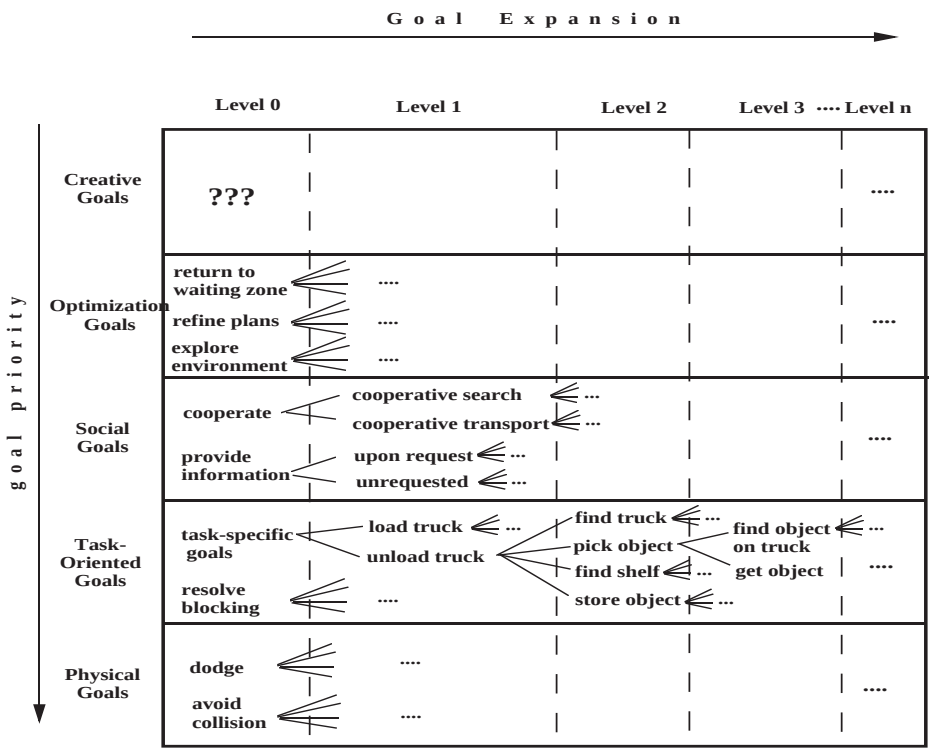

Figure 5.2: Goal Hierarchy of a Forklift Agent

5.3 Algorithms for Goal Selection

Depending on the priority type, the decider component of the BBC which has to select one goal from the set of active goals to be pursued next can have different forms. If we have only a static prioritization, the decision algorithm is very simple:

1. Choose the lowest possible layer with an active goal.
2. If there is only one active goal within this layer, choose that goal. Else, if there is more than one goal, choose a goal g randomly.
3. Expand the chosen goal g[2].
4. Continue with step 1.

If we use dynamic priorities, an agenda with active goals is maintained, and the algorithm is described as follows:

1. Compute the priorities of the active goals and rank the goals according to their priority in an agenda A.
2. Choose the first goal g in A.
3. Expand g.
4. Continue with step 1 as long as there are active goals.

[2] "Expansion" can have several meanings: if there exists an executable pattern of behaviour p_g for g, then it means to execute p_g. If there exists a skeletal (sub-)plan for g, then expansion is taken literally.

5.4 Discussion

In this section, we have developed a two-dimensional goal hierarchy as a data structure which allows an agent to decide what goals to pursue next. But there is more than that: the goal hierarchy can be regarded as a first step to an agent development tool for the designer of a multi-agent system. Our long-term vision is that the goal hierarchy represents a generic agent model. A designer can define possible goals, their priorities, and their implementation.

A major criticism of the model is that, in complex real-world domains, enumerating all the goals an agent may have, and thus also describing all the situations an agent may have to react to, may be too expensive, or even impossible. Although our current experience with three scenarios - which we regard as having "real world" size - does not confirm this assumption, we think that we have to take serious this objection in so far as it concerns the representation and recognition of possible situations. A good example is the representation of potential collisions between agents in the loading dock. Informally, the situational context for such a threatening collision of two agents can be described from the point of view of one of these agents as follows:

> A potential collision situation with an agent a_i exists if that agent is constantly approaching me, or if there is a field where the trajectories calculated in advance by linear interpolation of the current movement of myself and of agent a_i intersect.

Let us only consider the first part of the situational context: "an agent is constantly approaching me". Apart from the problems of defining the semantics of *constant movement* in terms of our simple model of perception, there are infinitely many ways how an agent may approach another one. It may approach from different angles, with different speeds, in different regions (for example in a wide hallway or in a large shelf). For different kinds of "approaching", different mechanisms may be required, such as making a random move if the threatening collision is constated in a wide hallway, or negotiating a joint plan if it happens in a narrow shelf region.

We regard the concept of *abstraction* to be the clue to solving these general problems. Therefore, the basic idea for coping with representing and recognizing all these different types of situations is to define a *third dimension* for our goal hierarchy: the dimension of *goal specialization*. It corresponds to a specialization / generalization hierarchy of situations which may possibly occur. Since situations are defined as specializations of other, more general situations (similar to the data model of an object-oriented class hierarchy), both the description and the recognition of situations can be performed in a clearer and more conomic way. Figure 5.3 shows an example for the specialization dimension of the conflict situation *blocking*, which describes the situation where one agent blocks another agent's way. The idea is to associate different - more or less efficient mechanisms of interaction to different situations. The more the agent knows about the current situation, the more goal-directed it can act, and the more efficient methods for avoiding the conflict can be employed. What is important is that there is a not necessarily efficient, but but robust and safe "emergency behaviour" which serves as a backtrack point for the agent either if the situation requires to react quickly, or if other, more sophisticated mechanisms of avoiding / resolving the conflict fail. For example, if all the agent can recognize is that there is an agent standing in front of it, the only mechanism it can use is a very simple one, namely making one or a couple of random moves (which can also consist in doing nothing at all). If it can recognize where the encounter is to happen, more specialized methods for blocking resolution can be employed. For example, if it occurs in the region around the truck, the other agent will probably wait for a possibility to load or unload a box from the truck. Thus, it does not make much sense to start an explicit method for conflict resolution, but it can be better just to wait until

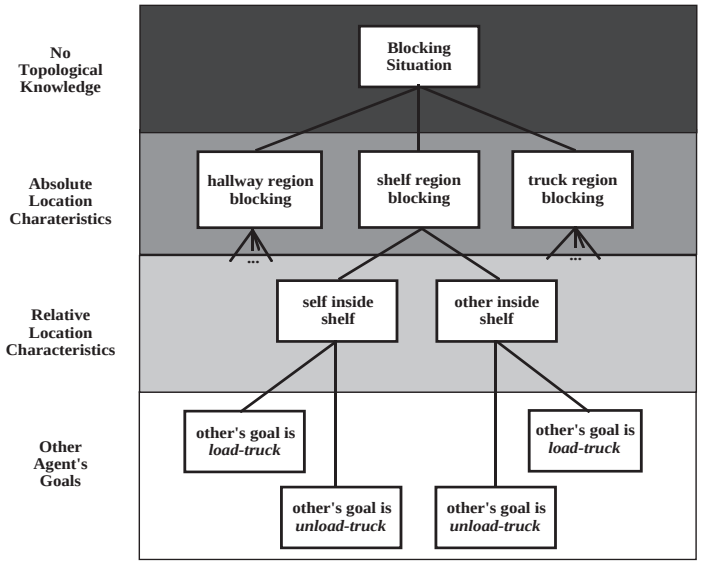

Figure 5.3: Example: Specialization Hierarchy of a Blocking Situation

the other agent is gone. If the blocking happens inside a shelf region, a cooperative method for resolving it can be used - a joint plan is proposed and negotiated. If, in addition, the agent knows which one of the two agents involved in the conflict stands inside the shelf, it can directly conclude whether or not it has to move away. If the conflict happens in a wide hallway, a local treatment via random moves is often the most appropriate one.

Note that the simple random move strategy, which is used as default, is sufficient for resolving the conflict in all relevant cases [Chu74] - although it may take a long time until it finishes successfully. In conclusion, using a hierarchical representation for the situations the agent may be faced with is a promising approach for coping with the complexity of real-world domains. The extension of the two-dimensional goal hierarchy (see figure 5.2) by the additional dimension of goal specialization is subject to our future work.

Chapter 6

Planning in Dynamic Environments

Planning is, without any doubt, one of the central issues in Artificial Intelligence. Moreover, most researchers believe that the facility to devise plans is the key property of intelligent systems. Therefore, a great deal of research has been settled in the area of AI planning (see [AHT90, AKPT91] for an overview). Most of these traditional approaches describe models of how a single agent can plan its actions in a static environment, i.e. in an environment which does not change except by actions performed by the agent itself.

However, the step from idealized, single-agent to realistic, multi-agent domains [BG88], and from closed systems to open systems [Hew85, HI91] has shown in a drastical manner the limitations of classical AI planning: what is required, are agents

- that can act in a goal-directed manner,
- that can react to unforeseen changes and events in their environment,
- that can maintain their bounded resources and use them in a reasonable manner,
- and that can flexibly handle requests for cooperation made by other agents.

These criteria are covered by the notion of agents who are able to *plan dynamically.*

So far, we have presented how InteRRaP agents can perform their tasks using plans and patterns of behaviour, and how the use of these concepts can be combined to yield an appropriate overall behaviour of the agent. In this section, we will go into more detail as to is how the InteRRaP model supports dynamic planning, i.e. what makes the model especially suited to tackle highly interactive and dynamic domains such as the robotics domains described in [MP93]. A comprehensive example for aspects of dynamic planning in InteRRaP is given in chapter 11 by means of the loading dock application.

There are two basic points which contribute to making the system flexible: firstly, the designer of the system has the possibility to define both behaviour-based and plan-based mechanisms for achieving certain goals. Secondly, due to the flexible flow of control, a new situation can be tackled at an appropriate layer. For example, a threatening collision can be avoided by a pattern of behaviour unnoticed by the planner that keeps on planning the path of the agent in the meantime.

6.1 Behaviour-based vs. Plan-based Mechanisms

What makes planning in dynamic environments such a difficult task is that the probability of getting a plan accomplished smoothly decreases if the number of agents rises. Let us make

this clearer informally by means of an example.Let us have a look at the *goto_landmark* plan step in the above example. As is shown in figure 11.1, agent i has two alternatives for reaching a given landmark. Firstly, it can use a pattern of behaviour, i.e. at a certain level of abstraction, this is a routine task for the agent which does not require deep reflection. Secondly, the agent can further decompose the plan step by first determining its actual location and then planning a list of moves the last of which leads to the goal landmark (this is accomplished by the predicate `gen_moves`).

The choice the agent makes between these two alternatives is crucial for how it can master the task. On the one hand, if it chooses the plan, after a brief planning time, it can walk straight ahead to the goal landmark. A pattern of behaviour for this task is likely to be less goal-directed; for example, it may use a weighted random function[1]. However, let us now assume that another agent, say j, blocks the way of agent i, and let us first consider the case that the agent has chosen the plan-based processing of the `goto_landmark` subgoal. In this case, i notices that a specific plan step `goto_field(From, To)` cannot be executed, since field `To` is not free. As a consequence of this, the PBC has to be activated again. The original plan must be modified, an alternative way has to be planned, or a negotiation with agent j is initiated where a joint plan to resolve the conflict is devised. This is, of course, a very costly and complex process considering the fact that also the new plan is likely to be interrupted.

What we would prefer is to shift some intelligence into the execution. This is done by defining a pattern of behaviour `goto_landmark_beh` which *somehow* finds a way to the goal landmark, and which is able to handle some exceptional situations from within it. For example, while walking straight ahead, a common exceptional situation is that another agent blocks the way. Thus, the behaviour can cope with this situation by trying to take a square aside randomly. In most cases, applying this strategy will be sufficient to resolve the conflict. What is important is that the pattern of behaviour is able to recognize situations it cannot cope with, i.e. it has *self-monitoring capabilities*. In these situations, the behaviour-based component calls teh plan-based component. This leads us to the second crucial point which explains the dynamci planning facilities in INTERRAP: the flexible interplay among the control modules. It is explained by means of an example in the following section.

6.2 Planning as an Interplay of Control Modules

6.2.1 The Bottom-up Control

The first phase of the planning process is defined by the bottom-up direction of control, which has already been discussed in section 4.4 describes the flow of control from the world interface up to the cooperation component

World Interface and BBC

As it has been described in section 4, incoming information is stored in the agent's perception buffer and in the world model part in the world model. Such information can be, for example, the existence of another agent inside an agent's range of perception. This information can be used by patterns of behaviour in the BBC in order to recognize certain conflict situations. Thus, the bottom-up direction of the interaction between the world interface and the behaviour-based component is implemented via shared memory - that means, the world model and the perception buffer.

[1] We use randomness as an important means to keep the system deadlock-free.

BBC and PBC

The blocking situation in a narrow shelf is recognized by a pattern of behaviour. It is a special case of the general *blocking situation*. The pattern of behaviour recognizes that it is not competent in this situation, but that it is better to pass on control to the PBC[2]. It does this by sending a request message

```
do(resolve_shelf_conflict((self, Agent)))
```

to the PBC

PBC and CC

The *pbc_control* recognizes that solving a blocking conflict requires a joint plan. Therefore, it calls the cooperation component. This one devises a joint plan, and initiates a negotiation with the other agent on the join plan. In the simplest case, negotiation may consist of directly accepting the first plan proposed. However, an iterated process of plan modification and plan refinement may be necessary in order to come to a mutually accepted plan.

6.2.2 The Top-Down Control

The second phase of the planning process is top-down. It starts from the plan-based or - in the case of an interactive plan - from the cooperation component and ends in the execution of actoric and communicative primitives by the world interface.

CC and PBC

The resulting joint plan is translated by the CC in a single-agent plan which is augmented by synchronization commands. For example, a very simple joint plan for two agents i and j changing places is

```
JP = [[[walk_aside(i, north)], []],
      [[walk_ahead(i)], [walk_ahead(j)]],
      [[walk_aside(i, south)], []]].
```

The plan `JP` has three phases. Phase describes that agent i must first walk aside. In phase two, i and j may concurrently walk ahead. Phase three, where agent i steps to the other side again, can only start after agent j has executed the `walk_ahead` action. This joint plan is translated into the following single agent plan P for agent j:

```
P = [walk_aside(north),
     send_synch(j, ready),     ;; send message to agent j
     walk_ahead,
     on receive(j, ready) do ;; wait until j has performed walk_ahead
        walk_aside(south),
        send_synch(j, ready)].
```

The synchronization commands ensure that the precedence constraints expressed by the joint plan are respected in the execution. This plan is passed to the PBC.

[2]This is what we mean by *competence driven* activation.

PBC and BBC

The PBC executes the plan by activating patterns of behaviour for the leaf steps of the plan, such as `send_synch`, `walk_ahead` etc. For example, in the case of the `walk_ahead` pattern, the activation is performed via a

```
request(bbc, activate(walk_ahead))
```

command.

BBC and World Interface

The pattern of behaviour which has been selected by the plan-based component now becomes active. When it is chosen by the *bbc-control* mechanism, its execution part is performed. In the case of a `walk_ahead` in a real robotics environment, execution means to launch a (possibly hard-coded) control program which has the robot take one step ahead. If execution is only simulated, it means sending a *walk_ahead* message to the world process. According to the definition of the communication interface between the BBC and the World Interface (see section 4.5),the BBC performs this activation by sending a message

```
request(wif, execute(walk_ahead))
```

to the world interface. After having executed the action, the world interface reports a

```
inform(bbc, done(execute(walk_ahead), Status))
```

to the BBC. The `Status` parameter describes the outcome of the action. In this case, `Status` $\in$ { `success, fail`}.

6.3 Conclusion

In this section, we described two features of the INTERRAP model which makes it possible to cope with the caveats of dynamic planning. Firstly, the use of patterns of behaviour allows a shift of intelligence from planning to execution. Thus, the difficult and time-consuming process of revising a current plan can be often omitted. Secondly, the modular structure of control and the flexible interplay among the individual modules of the control unit allows to cope with unforeseen events at the appropriate level.

In conclusion, patterns of behaviour are flexible, but maybe less efficient mechanisms, whereas plans lead to intelligent solutions, but are likely to fail in a dynamic environment. The crucial point is to find the appropriate granularity, i.e. to find criteria for deciding what should be done by defining a pattern of behaviour, and what should rather be performed by devising and executing a plan. Apart from intuitive criteria, which are useful but definitely insufficient, empirical results gained by testing different configurations and decision strategies in a concrete scenario will provide valuable information. We refer to section 12 for a discussion of that issue.

Chapter 7

Related Work

In this section, we will take look at important work done in the area of agent models, knowledge representation, dynamic planning, and agent interaction, which is closely related to what we do. However, due to the variety of related fields, it is impossible to provide a detailed overview of the fields in this reports. Thus, we will restrict ourselves to discussing the research which we feel is mostly related to ours.

7.1 Agent Architectures

Agent architectures for DAI have been basically influenced by three seminal research projects: the HEARSAY speech understanding system [FL77],which introduced the blackboard architecture, BEINGS' cooperating experts paradigm[Len75], and the work done by Carl Hewitt in the ACTORS system [Hew73, Hew77, AH85]. In contrast to the former projects, ACTORS is still the subject of ongoing research in DAI. ACTORS can be regarded as a precursor of concurrent object-oriented programming systems. ACTORS have a set of behaviours which are triggered by receiving a message. However, the ACTORS paradigm describes very fine-grained agents. For example, it does not support the representation of higher-level constructs for communication and cooperation. Moreover, planning facilities are not an integral part of the architecture. Thus, the model is rather an elegant model of concurrent computation [Woo92] than an agent model in our sense.

Since the late eighties, quite a few agent architectures have come up whose development was coupled with a *DAI testbed*, the first of which has been MACE [GBH87]. Other examples are CooperA [ALS89], the high-level operating system AGORA [BAF+87], MAGES [BFS91], MICE [DM90], MCS/IPEM [DCJL91], RATMAN [BM91],and DASEDIS [BS92].

[SMH90] propose to view an agent to consist of a *mouth* (the communicator), e *head*, and a *body*. The head reasons about the functions of the body and exerts the agent control. The body describes the application-oriented processing facilities and knowledge of the agent. Whereas this agent architecture is useful in its generality, it obviously requires concretization and refinement. We consider the INTERRAP as a refinement of this *head-mouth-body* architecture. The mouth corresponds to the world interface in INTERRAP. The agent control module and the application-independent parts of the hierarchical knowledge base can be regarded as a refinement of the head; the functionality of the body is represented by the application-specific part of the agent knowledge base.

ARCHON [Wit92] has been developed as an architecture for Multiagent Systems for industrial applications. Therefore, the model mainly concentrates on aspects of cooperation (which are covered in the so-called *ARCHON layer* of the model), and is not aimed at defining the functionality of the individual system..

In the area of micro models of deliberate agents, important work has been done by Shoham (see [Sho93] for an overview). Similar to Shoham, our model defining the behaviour

of an agent in terms of its mental state (beliefs, goals, commitments). However, our system architecture is more fine-grained and provides an explicit description of a behaviour-based part.

What is common to all the models mentioned above is that they embody architectures for rational agents with an explicit symbolic representation of the knowledge, skills, goals, and plans of the agents. At the other end of the scale, there are biological approaches (behaviour-based architectures). These (e.g. the work of Maturana as well as [Hub88, KHH89, Fer89, Ste90] describe the individual agent as very simple stimulus-response systems and focus on the behaviour of systems consisting of large numbers of these agents.

Finally, [Kae90, Fer92, Had93] are current approaches towards hybrid agent architectures, which come close to our basic ideas. [Kae90] proposes to regard an agent as organized vertically in two components, the action component and the perception component. The action and perception modules themselves are structured horizontally (cf. [Bro86]) in various submodules, which correspond to several partial functionality such as reacting, planning, learning, modelling etc. Each submodule of the action component has access to all submodules of the perception component. However, Kaelbling's model does not provide a structured knowledge base. Like the PRS system [GL87], Kaelbling's system is discussed in more detail below in the context of dynamic planning.

In his brilliantly written dissertation [Fer92], Ferguson presents an architecture for dynamic, rational, mobile agents, called *Touring Machines*. His model basically describes a dynamic layering of a number of deliberative and nondeliberative control functions called modelling, planning, and reacting. There are several similarities between Ferguson's ideas and ours. However, there are quite a few considerable differences:

- In contrast to INTERRAP and similar to [Bro86], he proposes a horizontal coupling of the control layers with the perception and action subsystems, i.e. each control module has direct access to perception and action. This leads to a control architecture which differs considerably from ours, using a set of global censor and suppressor rules to be defined for application at the input and ouptut of each layer. Censor rules serve as filters between the agent's sensors and its control, suppressor rules serve as filters between the output of the control layers and the actuators. Censors are used to prevent certain information from being transmitted to selected control layers. The suppressors can check whether actions proposed by several layers might conflict with each other. The bidirectional connections among control modules and their hierarchical organization in INTERRAP allows us to obtain one direction of control for free. This is covered by the goal priorization dimension of the INTERRAP goal hierarchy (see figure 5.2). The opposite direction (top-down control) is covered by the top-down activation mechanism (see section 4.4). Since INTERRAP has a vertical architecture for perception and action, suppression can be exerted by direct communication among the modules. This is not possible in Ferguson's horizontal architecture, because one control component (for example the reactive one) cannot see what actions another component (say the planner) proposes.

- The *Touring Machine* model does not account for cooperative agent interaction in the sense of collaboration towards the achievement of common goals. The modeling layer $\mathcal{M}$ is restricted to cope with unforeseen situations such as conflicts among agents. In INTERRAP, the functionality of layer $\mathcal{M}$ is covered by the plan generator components contained in the PBC and the CC. The cooperation component is an extension of Ferguson's model since it allows us to maintain knowledge and strategies for cooperation, and it could thus be designed as a fourth layer $\mathcal{C}$ of the Touring Machine model.

- We feel that our notion of patterns of behaviour is much stronger than the situation-action rule sets in Ferguson's reactive layer, since it includes procedural knowledge and

routine behaviour, which is not merely reactive, but which allows a shift of intelligence from planning into the execution.

- Finally, Ferguson's model does not provide an explicitly layered knowledge base.

Haddadi [Had93] proposes an RDR ("Reasoning, Deciding, Reacting") architecture. Its main components are an agenda, an intentional structure, and the procedures realizing the reasoning, deciding, and reacting facilities of the agent. However, in her model, both reactive, deliberative, and goal-directed features are based on the same symbolic, script-like representation. The focus in the model seems to be rather on the deliberate, rational part than on the reactive part.

7.2 Knowledge Representation

Approaches concentrating on describing decisions of an agent as a function of its mental state (beliefs, goals, intentions) (see [CL90, RG91]), provide a great deal of formal and semantical clearness. The focus of our work is defining architectures for building concrete multi-agent applications. From this pragmatic point of view, and for the time being, a more straightforward way of handling the mental attitudes attributed to our agents seems sufficient to us. At a later stage of development, it can be extended to a formal logical model similar to the ones listed above. The work of Halpern and Moses done in formally defining several types of knowledge in distributed systems [HM90, HM92] and its relation to our research were discussed in detail in section 3.3.

Another field of research which is of interest for knowledge representation in multi-agent systems are terminological logics, also known as concept languages or descriptive logics [BBH+90, WS92]. By the example of the transportation domain [FKM+93], we evaluated the applicability of the KRIS system for knowledge representation and inference [BH90]. It turned out that KRIS is well-suited for the representation of static knowledge such as object taxonomies[1]. However, for the time being, there are some problems:

- Most languages which were explored (e.g. $\mathcal{ALC}$) turned out to be not expressive enough to model the relevant features of the domain.

- KRIS does not provide the possibility of modeling knowledge and beliefs of agents as modalities. This, however, is crucial if we want to draw a distinction between what holds in the world and what an agent believes to hold. Currently, there are research efforts to integrate modal operators for knowledge and belief into the language $\mathcal{ALC}$ [Lau93].

- A formal treatment of dynamic scenarios such as the loading dock presented in this paper crucially requires coping with changing, possibly inconsistent knowledge, and thus requires some form of nonmonotonic reasoning. Up to now, this is beyond the scope of KRIS. However, the recent integration of *defaults* [BH92] is regarded as a first step into this direction. Unfortunately, first results with the default extension of KRIS reveal that the system efficiency seems not yet tractable for practical use.

These problems caused us not to use KRIS as a knowledge representation tool for our current implementation of the loading dock. However, there are continuing tight contacts between the two research groups, and we hope to be able to use the results achieved in the TACOS project, which will allow us to represent changing and inconsistent knowledge and beliefs of agents.

[1] In the concrete case, KRIS was used to model different kinds of goods, trucks, and the *can_transport* relation between trucks and goods.

7.3 Agent Interaction

The topic of agent interaction is one of the basic fields of research in DAI. Much attention has been paid to the study of multi-agent interaction in DAI, psychology, economics, cognitive science, and sociology (e.g. organizational theory). Different types of interaction, such as avoidance and resolution of conflicts, competition, and cooperation have been described. As regards the understanding of "interaction", there are different points of view: For instance, Sycara [Syc87] described interaction as positive or negative interference among the goals of different agents. She called this relationship *goal interaction*. In many interesting multi-agent domains, such as robotics, where plans are not only devised, but also physically executed, there is an additional dimension, which can be characterized by the term *physical interaction*. Physical interaction implies the necessity to react quickly to unforeseen events caused by other agents. It will be discussed in the area of dynamic planning.

Research in conflict resolution which affects our work has been done by [Syc89, Kle90]. Sycara proposes to resolve conflicts between agents by a mediator. In cases where such a mediator is available, this is of course useful. However, for the sake of generality, we do not assume the existence of such a mediating process *a priori*. We prefer a centralized conflict resolution strategy where one of the agents is elected to be the mediator, i.e. to generate proposals. This enhances the robustness of the system in the case of a break down of the mediator. Similar to social laws, we regard a central mediator as an additional feature for increasing the efficiency of the system in cases when it is available.

The SIPE system [Wil88] considers conflicts occuring in the processes of executing plans in parallel branches.

Seminal work in describing architectures and protocols for exchanging and processing distributed knowledge, goals, and plans has been done by [DS83, Ros85, CL88, CML88, DL89, DM91]. Purely cooperative agents have been modeled by [ZR89, Jen92]. Actually, all of the research mentioned before describes rational, plan-based interaction. So far, hardly any attention has been paid to the scope and the mechanisms which are available for *behaviour-based agent interaction*. Havingmade explicit both behaviour-based and plan-based concepts in INTERRAP allows us to develop and to compare both paradigms also for the topic of agent interaction.

7.4 Negotiation

One of the most difficult topics in the research on agent interaction inside DAI has been *negotiation*. Apart from voting mechanisms, negotiation is the central technique needed in order to allow autonomous agents to find mutually acceptable or beneficial agreements on some matter. The contract net model [DS83] introduced a simple negotiation protocol among autonomous agents. It has been enhanced by a sophisticated economical model by [Mal87]. Rosenschein et al. [RG85, ZR89, ZR92] consider negotiation from the corner of game theory. They distinguish between goal-oriented, task-oriented and worth-oriented domains. Most of their past research has dealt with how agents can agree on some form of coordinated action, and on what forms of protocols should be used for this purpose. Recently, there have been attempts to improve the shortcomings of local decision-making and to guarantee the honesty of agents in *voting protocols* by introducing a tax system which is proportional to the utility an agent receives for a deal [ER92].

Sycara has suggested to combine the use of Case-Based Reasoning [Syc89] and decision-theoretic such as preference analysis [KR76] or the use of heuristic methods like texture measures [SRSF90] to generate proposals and to reach decisions. Over the past few years, starting with [SF89], the idea of using of constraints to express interrelations among a group of agents has become quite popular (cf. [SRSF90]).

As regards possible purposes of negotiation, [Syc89, KT93] have described negotiation for conflict resolution. Negotiation aimed at implementing cooperative action has been described by [Les91] (FA/C paradigm), DurfeeLesser89 (exchange of partial global plans), [CML88] (multi-stage negotiation) and [vM90, KvM91] (plan coordination).

All the above approaches towards modelling negotiation have concentrated on describing the protocol and decision layers of negotiation. They do not deal with the question for what kind of interaction negotiation is required, and what kind of interaction local and behaviour-based mechanisms can be used.

7.5 Social Laws

An alternative approach to increase the robustness of a system is to decrease the number of possible conflict situations in it. For this purpose, Shoham and Tennenholtz [ST92] have proposed the use of social laws. We regard this as a valuable possibility to optimize the performance of an agent society and as a feature that can be used on top of a basic agent architecture such as INTERRAP rather than as an alternative to our concept.

7.6 Plan Recognition

Plan recognition [Kau91] is an important field for agents acting in a dynamic environment. An agent can only react in an intelligent way to a situation if it takes into consideration the current and future behaviour of other agents. Plan recognition is one possibility to infer the presumable future behaviour of anotehr agent (i.e. its current plans) abductively from its previous behaviour. However, plan recognition is a difficult field of research on its own. Since we assume that agents can communicate, and thay they - in our application - willing to reveal their goals to other agents, we use communication in order to exchange the goals of agents.

7.7 Dynamic Planning

A great deal of our work affects the area of dynamic planning. Classical AI planners [FHN71, Wil88] usually consist of a plan generation module and a plan interpretation module. Plans basically are sequences of primitive actions. Since in many real-world domains, information comes in incrementally, other approaches have tried to interleave plan construction and plan execution (e.g. [DL86]). Georgeff and Lansky [GL86] have proposed the use of *precompiled methods* in order to be able to cope with real-time constraints. A more drastical treatment of the reactivity requirement is postulated by [Bro86, Kae90, Suc87]. Recently, architectures for reactive planning have been proposed [GL87, McD90] which have shown new ways to integrate aspects of deliberate planning and reactive behaviour.

This development has lead to a general architecture for these kind of systems. A reactive planning system consists of a planner and a reactor module [Par93]. There exist different possibilities how to define the interplay among these modules: either there is a behaviour-based component which can call a planner (e.g. Newell's SOAR system[Wal91]), or there is a planner with an associated mechanism for interrupt handling and replanning [PR90]. Approaches described by [CGHH89] and [LH92] seem to be closer to our model, since the planner and the behaviour-based component (reactor) run in parallel: in [CGHH89] the planner can adjust (overwrite) decisions of the behaviour-based part; in [LH92], the reactor can even be rewired by the planner, i.e. new patterns of behaviour can be learned from plans. However, there is a static control hierarchy in these approaches, which, in addition, is restricted to a single level of depth.

The InteRRaP model presented in this report provides a modular, layered organization both of control and of the agent knowledge base. Bottom-up control is *competence-driven*, i.e. a module (for example the behaviour-based component) shifts control to the next higher component (the plan-based component) if the task to be performed exceeds its competence. Top-down control is activation-driven. For example, as it has been described in section 4, the plan-based component can activate a pattern of behaviour in the behaviour-based component.

Part II

The Loading-Dock Application

Chapter 8

The Domain

8.1 The Domain

The application described in this section deals with the simulation of an automated loading dock. In the loading dock, there are shelves with different types of goods. Automated forklifts have to load and unload incoming trucks. Figure 8.1 gives an idea of the structure of the scenario. The forklifts need to have capabilities for perceiving their environment.

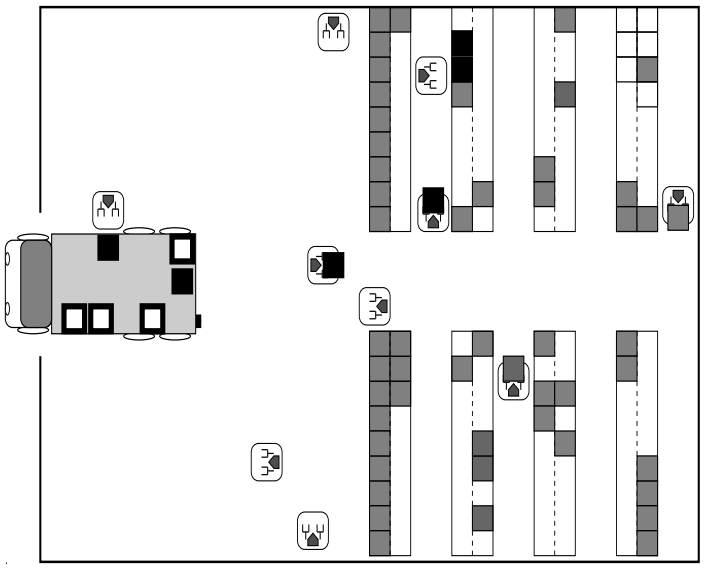

Figure 8.1: The Loading-Dock Multi-Agent Scenario

They must be able to recognize the relevant objects in the domain, such as the truck, the shelves, the boxes, and other forklifts. Moreover, they must be equipped with the ability to grasp a box and to load it on a truck or to store it in a shelf. Thus, the scenario is basically a robotics application. A typical real-world application for this kind of scenario are Flexible Transport Systems (FTS). These are characterized by the fact that the movements of the robots are not arbitrary, but that they are directed by guiding lines or by tracks. The simple grid representation of the world, which is discussed in more detail in section 9, turns out to be compatible to the idea of modeling groups of flexible transport vehicles.

The nature of the application involves several difficult and interesting problems, which we discuss in section 8.2. The relevance of concepts and methods from Distributed Artificial Intelligence for tackling these problems is outlined in section 8.3. Finally, motivated by

these considerations, we briefly describe the main ideas of our approach to modeling the loading dock application in section 8.4.

8.2 The Problems

The main feature of the scenario is its immense physical dynamics. The actions of the forklifts moving around in the scenario must be coordinated, collisions have to be avoided, deadlock situations (such as cyclic waiting situations between forklifts) must be recognized and resolved. In addition, although we did not primarily think of aspects of cooperation when firstly looking at the loading dock domain, it contains quite a few examples for problems where a cooperative approach seems reasonable. Moreover, since the number of forklifts is not fixed, new ones may be added and existing ones may be disposed of. Thus, there is an open planning problem which is very hard to solve.

The problem of perception and mechanical control is still of great relevance for research in robotics [vP93]. The extraction and interpretation of sensoric data is still a considerable problem. The use of video cameras involves all the problems of object and scene recognition. The use of ultrasonic sensors in multi-robot scenarios is problematic, because the existence of many agents equipped with ultrasonic sensors makes the sensors themselves obsolete - the agents "shoot" each other with the sensors (a way to tackle this might be to use different frequences). At present, laser sensors are regarded as the approriate sensor type, in practice. They allow a very precise measurement of distance, and thus of the position of a robot, and are factually insensitive to interference caused by other robots using the same sensoric technology.

The central planning of the activities and the paths for a group of robots is a very serious problem. Experience in robotics (see [Bro86, Lat92, vP93]) has shown that it is in fact not viable. Coordination of robots requires decentralization. This consideration leads us to the following section, where we will discuss in how far concepts and methods from DAI can help us any further.

8.3 DAI Aspects

Some of the problems discussed above are at the same time arguments in favor of a DAI approach. As we have discussed in the previous section, the hard problems imposed by the domain make a decentralized approach reasonable, even more, we feel that such an approach is required to cope with the complexity of the system, which is caused by the variety of the possible interactions occurring among the robots in the scenario.

By a decentralization of information and control, which means giving the forklifts facilities for reasoning and communication, the forklifts are able to avoid collisions, to resolve blocking situations based upon their local point of view and on their own knowledge about the state of the world - in conclusion, the dynamics of the system as a whole can be reduced to the dynamics caused by local encounters among single agents. As has been shown by earlier projects in DAI [CL87, GBH87, KMM93a], this leads to a considerable reduction of problem complexity.

What makes the DAI approach useful in this case, is the inherent distribution of information and control in the loading dock. There are physically distributed entities with a certain complexity - the forklifts. Thus, modeling these as agents is very natural - as natural as modeling them as objects in an object-oriented design approach.

So far, we have motivated the "D" in "DAI" - but, what do we need the "AI" for? Again, there are several answers to this question. Firstly, the open planning problem and the local view confront us with incomplete, possibly even inconsistent knowledge (we should rather

call it *belief*) the agents have about their environment. This, however, is exactly the point where non-AI approaches fail. Research in non-monotonic reasoning, fuzzy theory, or neural nets offers at least first approaches towards this issue, although it has been recognized that there is no easy way out of this problem.

Secondly, in order to behave reasonable, agents need to have commonsense knowledge. A forklift which is in a conflict with another one can greatly benefit from knowing the topology of the environment (for example: *in which direction is the exit from the shelf corridor?*) as well as knowing something about the goals ascribed to other agents (*does the other agent really want to enter the shelf corridor, or is it already on its way back?*). Thus, representing the knowledge agents have about the world, about themselves, and about other agents is a crucial point.

Thirdly, the loading dock is a good example for an application where computers and men might co-exist and collaborate. Human employees might work in the loading dock together with the forklift agents. This makes the use of high-level communication facilities such as speech acts a reasonable approach. Whereas we could argue that a much simpler communication language would be sufficient in the case of inter-robot communication, things should be viewed slightly different in the light of man-machine communication.

Finally, since the forklift agents act in a goal-directed manner, it makes sense to ascribe goals to their behaviour, and to allow the agents to reason about their own goals as well as about the goals of other agents. This information can be used by the agents in order to negotiate cooperative solutions for goal conflicts, for example in the form of a joint plan.

8.4 The Basic Approach

Motivated by the considerations made in the previous section, we will now briefly describe our approach towards modeling the loading dock domain. It will be explained in more detail in chapter 9. The basic idea is to design the scenario as a multi-agent domain. This means that there is no central instance, which might coordinate the activities of the forklifts. Instead, the forklifts are considered to be the only agents in the system. They are equipped with problem-solving facilities, with actoric abilities (they can turn around, move around in the loading dock, take hold of goods, and store them), with perception, and with the ability to communicate with other agents.

Loading and unloading orders are given to the agents (by the human user, or directly by the incoming truck). The agents have patterns of behaviour and planning facilities at their disposal which allow them to achieve their local tasks appropriately. Moreover, they can use both behaviour-based and plan-based methods of interaction in order to avoid and to resolve conflicts with other agents. Behaviour-based methods are often local, plan-based methods are mostly cooperative in a sense that the goals of the other agents are taken into consideration when the planning is done. Coordination using communication is achieved by using communication and negotiation protocols.

In the next chapter, we will describe in more detail how the loading dock has been modeled as a multi-agent scenario, how the agents are designed, which patterns of behaviour and which plan-based mechanisms for action and interaction they use, and how agent interaction in the scenario is tackled. We will show how the agent architecture INTERRAP that has been introduced in the first part of this chapter can be used as a basis for designing the forklift agents in a way that they satisfy the three main requirements which are put forward by the application: *reactivity, goal-directed behaviour*, and the *ability to interact* (cooperate) with other agents.

Chapter 9

Modelling the Loading-Dock using InteRRaP

In this chapter, we describe how the forklifts are modeled as InteRRaP agents. We present the behaviour-based and plan-based mechanisms for action and interaction which are employed by the agents in order to achieve their goals and to cope with situations where conflicts with other agents occur.

9.1 Representation of the World

The world - which means the loading dock itself - is designed according to a raster-based representation. It is represented as a set of fields which are arranged in a rectangular form to the loading dock. If we map this representation to a real-world loading dock, the size of the single squares should be about 2×2 meters.

The individual squares are described as objects which have certain attributes. Squares can be either empty (ground) or non-empty (occupied). If they are non-empty, they can either bear a static object (or a part of such an onject), or an agent may occupy the square. Static objects are boxes, shelves, and the truck. There is a hierarchical relationship between squares and objects, e.g. a shelf object consists of a set of square objects which are topologically related.

There are some constraints which hold in the world, and which have to be respected by the agents when they perform their actions:

- Moves of agents must keep them within the boundaries of the loading dock.
- A box must only be stored at a truck field or at a shelf field, but not on a ground field.
- An agent must not walk through shelves, boxes, other agents, or the truck.
- An agent can keep at most one box at a time.

The Simulated World In the simulation, the "true" state of the world is represented by a *world process*. The world process receives action request by the agents, checks whether performing the action would violate the consistency conditions imposed by the above constraints, and, if this is not the case, updates the state of the world, triggers the visualization of the action in the graphical simulation window, and the agent that the action was executed sucessfully. Moreover, using the world process, a concept of active perception is implemented. This is explained in more detail in section 9.2.2.

9.2 The Forklift Agents

Since the forklifts are the *acting* units in the scenario, modeling them as the agents in a multi-agent system seems a very natural matter. In the following, whenever we speak about agents, we mean the forklift agents. In this section, we will describe the functionality of the agents, their actoric, sensoric, and communicative features, and the structure of their goals and their knowledge. Thus, what we describe in the following basically represents the functionality of the INTERRAP world interface (see figure 4).

9.2.1 Actoric Facilities

Since the agent is to act in a physical environment, it needs facilities to manipulate the environment, to perform physical actions. The forklift can perform the following basic actions, which are directly implemented as primitives in the world interface:

- The agent may move to the square field in front of it, provided that the field is empty. This is represented by a primitive action `walk_ahead`.
- An agent may turn around by an angle α. In our implementation, α can be only $+90°$ or $-270°$. We represent this by actions named `turn_right` and `turn_left`, respectively. There are no constraints for this action, so an agent can always turn around.
- An agent may grasp a box standing in a shelf or on the truck. This is expressed by a primitive action `grasp` the agent may perform. However, grasping something is only possible if the object to be grasped is directly in front of the agent.
- Analogously to the `grasp` action, the agent can put a box either in an empty shelf or on an empty truck field, if that field is directly in front of the agent. This is represented by a `put` action.

Executing in the simulation is realized by sending a request for executing this action to the world process described above.

9.2.2 Sensoric Facilities

The basic sensoric facilities of the forklifts are implemented by means of a simple model of perception which is based on the rastered representation of the world we have discussed above. An integral part of the perception module in the world interface is the *perception buffer*. It contains the information the agent currently perceives. From a logical point of view, this buffer is part of the world model part of the knowledge base. Its importance stems from the fact that it allows the agent to check whether it currently sees something, and thus to draw a distinction whether it believes a fact to be true because it has perceived this fact earlier or because another agent has told it the fact, or because the agent currently perceives the fact - which is interpreted in our model as de facto knowledge[1].

Range of Perception

The basic idea is that agents are equipped with a certain *range of perception*. This specifies fields which can be seen by the agents. The range of perception can be defined in the process of agent configuration. Figure 9.1 shows different examples for ranges of perception. Figure

[1] If we assume noisy perception, we would rather let the perceived information pass through the information assessment module, where a certain mental attitude is associated with it, and treat it as "normal" beliefs with a certain credibility.

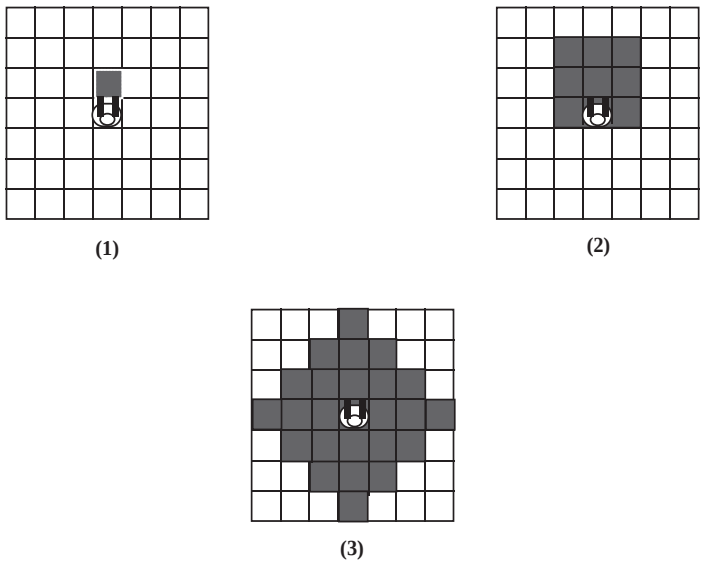

Figure 9.1: Example: Range of Perception

(9.1.1) shows a very simple but important case. The agent can only perceive the square directly in front of it. Thus, the agent resembles to a man who gropes one's way through a totally dark room, having to rely on the sensoric facilities of its hand and of what he has memorized by earlier experience. This case is important because it bears most of the desired complexity of the system which would be achieved by allowing a larger range of perception, but keeps the implementation of perception very economical. Therefore, we started with agents that have a range of perception of only one field. Figure (9.1.2) shows an agent who has a range of perception of 3×3 squares in front of it. This allows an agent to search through a shelf while wlking through the shelf corridor without having to turn around every time it wants to look at a certain shelf square. It also allows to recognize conflicts not only when directly facing another agent. Thus, a very simple form of collision avoidance can be investigated, since the two agents have to be careful not to move at the same time to a field separating one of them from the other.

However, in order to examine methods of collision avoidance in practice, a range of perception as shown in figure (9.1.3) is desired[2]. The length of the range of perception is dependent on the moving direction of the agent. This allows the agent to recognize and to react to potential frontal collisions very early. So far, we have not implemented this solution, since it is very expensive in the simulation. The subject of the complexity of perception is discussed in more detail below.

Perceiving Other Agents:

Since the forklifts do not operate in a real world, but in a simulated one, the problem of becoming aware of what other agents are currently doing is a non-trivial problem, which goes back to the philosophical question whether perception should be viewed rather as an active or as a passive process, i.e. whether the world "tells" an agent that something has changed or whether the agent has to look "actively" for what is going on. Which perspective one prefers is basically a matter of taste.

We decided to solve the problem by implementing an active perception concept which

[2]We thank Prof. Dr. E.von Puttkamer for suggesting us this solution

initiates from the simulation world. Each time an agent performs an action, the new range of perception of the agent is computed and is sent to the agent. On the other hand, the world has to check whether by the consequences of the performed action, something has changed within the range of perception of other agents. If so, the changes are transmitted to these agents, too.

An alternative solution would be that the agents themselves "look" at the world from time to time, for example before they execute an action. This would happen by sending to the world an *update request*. However, in our view, the former view corresponds better to the paradigm of situated agents. For example, when another agent approaches, our agent should at least notice it - and be able to react. This is not guaranteed when we leave the decision whether to look or not to look to the control of the agent. However, as we already noted, this decision is a philosophical one, and it is mainly a matter of personal taste which one is preferred.

Complexity of Perception

It is a consequence of the concept of active perception which is implemented by the simulation world that the computation of the new perception situation is done in a centralized manner by the world process, and is therefore quite expensive. Moreover, the complexity of perception functionally depends on the number of agents in the scenario, and on the range of perception of the individual agents. For any action performed by an agent in the world, the simulation world has

- to compute the new range of perception for the agent and send a description of the fields covered by the range of perception by the agent.
- to check for any other agent in the scenario whether by the action performed, something inside the range of perception of this agent has changed. If so, send this agent what has changed.

Assume that an agent has a range of perception of m squares. Assume further that for each agent, its range of perception is stored as a list of square descriptions by the world. Then, recomputing the range of perception can be done in a time linear to m. Additionally, let there be n agents in the scenario. For each other agent, we have to check whether the action has changed something in its range of perception. This can be done with a complexity of $\mathcal{O}(m * n)$ steps . Since n agents can perform actions concurrently, the total complexity of the active perception algorithm for n agents and a range of perception of m is $\mathcal{O}(n^2 * m)$ in the worst case.

Thus, the implementation of the concept can be regarded as tractable in general. However, the number of agents and the range of perception are critical factors for the efficiency of the concept. The simulation world may become a bottleneck if there are too many agents in the scenario having a large range of perception. In our implementation, we found that a range of perception of 9 squares for about 10 agents can be managed by the simulation world process, if we employ some algorithmic optimization. We will not discuss these improvements in more detail here because this is not the focus of our work.

9.2.3 Communicative Facilities

As we have seen above, perception is modelled as communication with the world. At a higher level, agents also need to communicate with other agents. The world interface provides two basic facilities for sending and receiving messages. An agent can send a message to another agent using a communicative action `send_msg(Id, Ref, From, To, Type, Content)`.

The receiving of a message is implemented asynchronously, i.e. the agent can receive messages at any logical point in time. Type and content of a message are specified at higher layers according to speech act theory. This issue has been discussed in section 4.5.

9.2.4 Tasks and Goals

Up to now, we have described the functionality of an agent in the loading dock. In this section, we will explain what motivates an agent to start its activities. There are two central notions in his context, namely *tasks* the agent receives, and *goals* the agent derives from these tasks.

Tasks

A forklift agent may receive tasks from the human user of the simulation. In a real-world implementation, the tasks could be provided by an order form carried by the truck. Tasks are of of the form

```
transport(Who, What, Where)
```

`Who` specifies the agent to whom the order is addressed. `What` specifies the box which is to be transported. `Where` denotes the destination. A box can either be transported from the truck to the shelf, or from the shelf to the truck.

In our model, we assume that tasks are dedicated to individual agents. Examining models of task decomposition and task allocation is an interesting subject on its own. We investigated it by the example of the transportation domain (see [KMM93b]). In the loading dock example, we are more interested in how the individual agent solves tasks dedicated to it, and study the interactions among agents solving tasks of their own, which are caused by the shared environment.

Goals

The agent uses an intentional structure in order to maintain and control its activities. This structure is the INTERRAP goal hierarchy described in section 5. There are two basic sorts of goals, namely *top level goals*, and *derived goals*. Top level goals are goals which directly emerge from external or internal events but whose origin is not caused by existing goals of the agent. Derived goals are specified according to the *goal expansion* dimension of the goal hierarchy. They describe how a goal can be achieved by decomposing it into subgoals.

The receiving of a task t triggers the emergence of a new top-level goal $goal(t)$ inside an agent. This view gives expression to the close correspondence between patterns of behaviour and goals - a goal emerges inside the agent corresponding to the activation of a pattern of behaviour. Goals and plans for their achievement are the basic objects to be exchanged among the modules of the agent control and thus, they are the objects the agent reasons about. In the following, the patterns of behaviour for forklift agents are described in more detail. The instantiation of the plan-based component is described in section 9.5.

9.3 Behaviour-based Modeling

In this section we will describe the behaviour-based component of a forklift agent in the loading dock scenario. According to section 4.2.2, it consists of a control component, of a unit which maintains a set of patterns of behaviour, and of a resource handler which is the local interface to the agent knowledge base and to the resources the agent has at its disposal. The built-in control mechanism which decides which of a set of active patterns may be executed has been described in section 5. For the description of patterns of behaviour,

we will use the specification language presented in section 4.2.2. Following the goal priority dimension of the agent goal hierarchy (see figure 5.2), we will discuss in an exemplary manner the patterns of behaviour the agent has at its disposal, starting with lower-level, physical goals, up to patterns of behaviour corresponding to social and optimization goals. For reasons of comprehensiveness, the effect of executing a pattern of behaviour (the actual procedure which is executed) is described informally.

9.3.1 Physical Layer

The physical layer of goal prioritization contains the situations the agent has to react to immediately in order to avoid being damaged. Figure 9.2 shows a pattern of behaviour which treats a threatening collision with an other agent that approaches[3]. The effect of the pattern is to have the agent take one step aside in order to avoid this collision whenever this is possible. Note that the general pattern *avoid_collision* does not take into consideration

```
( PoB
      :name avoid_collision
      :static physical
      :participants self; A: agent
      :sit_context curr_pos(self, (X1, Y1)) ∧ curr_pos(A, (X2, Y2))
         ∧ approaches(A, (X1, Y1)) ∧ dist((X1, Y1), (X2, Y2)) ≤ dmin
         ∧ is_neighbour_field((X, Y), (X1, Y1)) ∧ Bel(self, free((X, Y)))
      :mental_context true
      :postcond curr_pos(A, (X1, Y1)) ∨ curr_pos(A, (X, Y))
      :termcond curr_pos(A, (X, Y))
      :failcond ¬∃F.(is_neighbour_field((X, Y), F) ∧ Bel(self, free(F)))
      :exec_descr /* turn around if necessary, and move to field (X, Y). The primitive */
                  /* actions are implemented by sending the world interface a */
                  /* message request(wif, execute(Action)) */
)
```

Figure 9.2: Pattern of Behaviour *avoid_collision*

the agent's goals, but randomly selects a free square. A specialization of this general pattern is shown in figure 9.3. If the agent has the current goal of moving to one of the neighbour fields, and this field is not occupied, then the agent will select this field. The fact that the

```
( PoB
      :name avoid_collision_1
      :super avoid_collision
      :mental_context :add curr_goal(self, goto_field(X, Y))
)
```

Figure 9.3: Specialization of the *avoid_collision* Pattern

pattern of behaviour `avoid_collision_1` is a specialization of the pattern `avoid_collision` is represented by a slot `:super` in the declaration of `avoid_collision_1`. The specialization inherits from its generalization all the contents of the slots. However, these default values may either be overwritten or in the case of the context and condition description, extended by additional conditions. The latter case is indicated by the keyword `:add`. In this case, the

[3]Free variables in the formula are interpreted as being existentially quantified. Shared variable names inside one pattern refer to the same object.

new condition is computed as the conjunction of the condition of the generalization and the conditions for the specialization. In the case of figure 9.3, the mental context is specialized by the new condition `curr_goal(self, goto_field(X, Y))`. Since the mental context in figure 9.2 is `true`, there is no difference to write

:mental_context `:add` $curr\_goal(self, goto\_field(X, Y))$

or only

:mental_context $curr\_goal(self, goto\_field(X, Y))$.

However, if we are going to define a pattern *avoid_collision_shelf* which works exactly the way the *avoid_collision* pattern does, but becomes active only if the conflict happens in a narrow shelf, we can do this by writing

```
( PoB
```
:name *avoid_collision_shelf*
:super *avoid_collision*
:sit_context `:add` $region\_type((X_1, Y_1), shelf)$
...
```
)
```

As a result of this, the situational context for pattern `avoid_collision_shelf` evaluates to

:sit_context $curr\_pos(self, (X_1, Y_1)) \wedge curr\_pos(A, (X_2, Y_2))$
$\wedge\ approaches(A, (X_1, Y_1)) \wedge dist((X_1, Y_1), (X_2, Y_2)) \leq d_{min}$
$\wedge\ is\_neighbour\_field((X, Y), (X_1, Y_1)) \wedge Bel(self, free((X, Y)))$
$\wedge\ region\_type((X_1, Y_1), shelf)$

If we had written

:sit_context $region\_type((X_1, Y_1), shelf)$

instead, the old definition for `sit_context` would have been overwritten by the new one, which obviously would not have been the intended meaning. A reasonable execution description for the pattern of behaviour `avoid_collision_shelf` would be to pass the control for the situation on to the planner by sending a message

```
request(pbc, do(avoid_collision_shelf(self, A))),
```

since it seems reasonable to try to avoid the conflict cooperatively, for instance by finding a joint plan, which is generated by the cooperation component.

9.3.2 Task-oriented Layer

The task-oriented layer of the hierarchy describes the patterns of behaviour which help an agent do its "business as usual", i.e. to perform the transportation tasks. Figure 9.4 shows the top-level pattern of behaviour `treat_transportation_task` of the task-oriented layer. The pattern becomes active if the agent has a transportation task `transport(B, S, D)`, where B is a box, S is the source, D the destination, and if the task is the next one to be performed by the agent. Since a plan is to be used in order to achieve the top-level goal, all the pattern has to do is sending a

```
request(pbc, do(transport(B, S, D)))
```

message to the plan-based component, and then waiting for the acknowledgement of the PBC, which can be either "success" or "fail". Note that the agent's mental context does not play any role for activating the pattern. However, by the activation of the pattern, a corresponding goal `transport(B, S, D)` arises. Thus, although the agent receives tasks from the environment, the BBC and the PBC communicate only on the basis of goals, and not of tasks. The absence of a specific constraint on the mental context in this case entails that even if the agent currently believes that it is not possible to achieve the goal, yet it has to activate the planner and at least try. Only if the planner also fails to find a solution, the pattern may fail. However, if the agent perceives that the goal cannot be achieved (denoted by the predicate `Perc(...)`[4]), for example by perceiving that the box is not located at field S, the behaviour fails. This is an example for the different consequences which arise from the agent *believing* in a fact in contrast to the agent *perceiving* a fact.

```
( PoB
      :name treat_transportation_task
      :static task_oriented
      :participants :local
      :sit_context received_task(transport(B, S, D))
      :mental_context curr_goal(self, transport(B, S, D))
      :postcond Bel(self, ¬possible(transport(B, S, D)) ∨ Bel(self, curr_pos(B, D))
      :termcond Perc(curr_pos(B, D)) ∨ msg_received(pbc, done(transport(B, S, D), ok))
      :failcond msg_received(pbc, done(transport(B, S, D), fail))
                  ∧ Bel(self, ¬possible(transport(B, S, D)))
      :exec_descr  /* pass the whole goal of transporting the box to the */
                   /* plan-based component and wait for the confirmation.*/
)
```

Figure 9.4: Pattern of Behaviour *treat_transportation_task*

As we will see in chapter 11, in the course of performing the transportation task, the PBC activates further lower-level patterns of behaviour in order to achieve some subgoals. One of these patterns is described below, namely the pattern `region_search`. This pattern performs searching a box in a spatially limited region R, which may be, in our case, a shelf of the same type as the box. The pattern does not require that the agent knows something about the location of the box. The pattern is activated when the agent currently has the goal of searching a box which it has to transport to a truck, and if the agent is in a shelf region which has the same type as the box to be searched. The execution of the pattern keeps on either until the agent has found the box and stands in front of it, or until it has examined all reachable fields without success. In the latter case, the agent believes that box B is not in region R (according to the postcondition).

9.3.3 Social Layer

A typical pattern of behaviour at the social layer is shown in figure 9.6: if an agent is asked by another acquainted agent, say A, whether it knows the location of a certain box, say B, and if the agent believes that the box B is at location (X, Y), it replies this to agent A. There are a couple of things worth noting here.

- The `:participants` slot contains a role declaration: apart from the keyword `self` which always denotes the agent who uses the pattern of behaviour, there is a declaration $A : acquainted\_agent$. By declaring a role taxonomy, we can define patterns of

[4] $Perc(p)$ holds true for a formula p if $PERCBUFFER \models p$.

(PoB
:name *region_search*
:static *task_oriented*
:participants :local
:sit_context $curr\_pos(self,(X,Y)) \wedge region((X,Y),R,shelf) \wedge has\_type(R,T)$
:mental_context $Goal(self, search\_box(B)) \wedge has\_type(B,T)$
:postcond $curr\_pos(self,(X',Y')) \wedge region((X',Y'),R,\_) \wedge$
$(\neg found(Box) \Longrightarrow Bel(self, \neg curr\_region(B,R)))$
:termcond $Perc(field((X_1,Y_1),shelf,R)) \wedge curr\_pos(B,(X_1,Y_1))$
$\wedge\ curr\_pos(self,(X_2,Y_2),O) \wedge faces(((X_2,Y_2),O),(X_1,Y_1))$
:failcond $\neg Perc(box(B,F)) \wedge$
$(\forall F'.(field(F',shelf,R) \wedge Bel(self, reachable(F'))) \Longrightarrow looked\_up(F',B))$
:exec_descr `/* Move along the shelf and look whether you see the box, first */`
`/* in one direction, then in the other one. */`
`/* If you have perceived the box, move in front of it */`
)

Figure 9.5: Pattern of Behaviour *region_search*

(PoB
:name *give_box_info*
:static *social*
:participants $self; A : acquainted\_agent$
:sit_context $message\_received(A, query(location(Box)))$
:mental_context $Bel(self, curr\_pos(Box,(X,Y))$
:postcond $Bel(self, Bel(A, curr\_pos(Box,(X,Y))$
:termcond $message\_sent(A, inform(location(Box,(X,Y))))$
:failcond $message\_received(A, retract(query(location(Box))))$
:exec_descr `/* send the other agent a reply containing the current position of the box */`
)

Figure 9.6: Pattern of Behaviour *give_box_info*

interaction where the participants in a certain pattern must fulfil certain role conditions. An alternative way would be to add a new slot - called `:participant_desc` - where predicated required for possible participants in the interaction could be defined. Thus, the role declaration would be replaced by

`:participants` $self; A : agent$
`:participant_desc` $acquainted(self, A)$.

- Again, we could consider the pattern `give_box_info` as a specialization of a more general pattern of behaviour `give_info`, or even more general, `reply_to_question`, which would allow us a hierarchical definition of interactive situations making use of the inheritance of properties.

- In the definition of the pattern, there is no explicit information about how the execution of the *give_box_info* pattern corresponds to the execution of other patterns of behaviour. The actual control structure is according to chapter 5. Using the purely static priority algorithm, an agent would only answer a question of another agent if there are no active goals (patterns of behaviours) at lower levels of the agent's goal hierarchy, i.e. on the `physical` or `task-oriented` layers. This, however, would mean that an agent who has the goal to search a box for itself, would not respond to a question asked by another agent until the goal was no longer active. As discussed in

chapter 5, this undesired behaviour can be omitted by defining for each goal a *degree of satisfaction*, and to compute the relative importance of that goal as a combination of the static priority and the degree of satisfaction[5].

- Finally, note that what figure 9.6 shows is a *reactive, interactive pattern of behaviour*. In previous approaches, "behaviour-based" was mostly unified with "local", "without communication". However, in our approach, answering a simple question which does not require sophisticated planning but which can be done by simply looking up a corresponding fact in the world model part of the knowledge base, can be performed using a pattern of behaviour. In addition, if the pattern of behaviour does not find an immediate response, it may decide to pass control to the planner.

- Note that we do not have to define the case that the agent realizes that it actually does not know the position of the box as a failure condition. This insight would cause the agent to retract the belief which establishes the goal context, and therefore automatically finish the active pattern. If we liked an even clearer separation between conditions that must hold as preconditions and conditions which must hold during the execution of the pattern, we might extend the description of a pattern of behaviour by a *during* condition.

9.3.4 Optimization Layer

The optimization layer of the behavioural hierarchy contains patterns which are used in order to improve the behaviour of the agent in the world. The example we provide here describes how an agent which does not have a current task to perform can explore a region which it did not know before, but which seems interesting to the agent. In our application, a region is interesting if it is a shelf region. Curiosity is an important property of an agent which can be chosen as an option during the configuration of the agent.

```
( PoB
      :name explore_interesting_region
      :static optimization
      :participants :local
      :sit_context ¬received_task(_,_,_) ∧ curr_pos(self,(X,Y)) ∧ in_region((X,Y),R,_)
      :mental_context Bel(self,¬been_to((X,Y))) ∧ Bel(self,interesting(R))
      :postcond curr_pos(self,(X',Y')) ∧ region((X',Y'),R,_)
      :termcond ∀F'.((field(F',_,R) ∧ Bel(self,reachable(F'))) ⟹ Bel(self,been_to(F')))
      :failcond false
      :exec_descr  /* explore all reachable fields in the region and store the information */
)
```

Figure 9.7: Pattern of Behaviour *explore_region*

Apart from the patterns of behaviour described up to now, several other ones exist. For example, there is a pattern of behaviour for the resolution of blockings, for which we implemented several specializations. The pattern of behaviour `region_search` has a specialization `region_memo_search` which can be applied if the agent believes to know the region where the object to be search is[6]. In this case, a more efficient search strategy can be applied.

[5]There are other approaches towards solving this problem. For example, Ferguson proposes in his dissertation to use special control rules to resolve this kinds of inter-layer conflicts.

[6]It can only do so if it has the ability to memorize information about fields it has visited before. This can be defined during the configuration of the agent.

9.4 Behaviour-based Methods

Up to now, we have paid little attention to the methods that are actually applied by the agent for local problem-solving and interaction when executing a pattern of behaviour. We will discuss the issue of behaviour-based interaction in more detail in section **??**, below. In this section, we will give an overview of the behaviour-based methods an agent can employ in order to achieve its local goals.

9.4.1 Randomness

The importance and the theoretical power of making random moves is expressed for the case of an individual walking around randomly in a finite room by the random-walk theorem [Chu74] which says that starting from any point in a finite room, using a random walk strategy, any point can be reached arbitrarily often. Of course, the practical value of this theorem is limited by the fact that it may take *arbitrarily long* to reach a certain point. However, combining it with other methods makes it a powerful mechanism.

The idea of random behaviour is to choose randomly one of a set of alternative actions an agent is able to do in a certain situation. Formally, such a random choice function is defined as follows:

Definition 3 (Random Choice Function) *Let A be an agent, ACT_A be the set of actions which can be performed by A. Let $\mathcal{S}$ be the set of possible situations, $\mathcal{G}$ the set of possible goals. Let $S \in \mathcal{S}$ be the current situation, $ACT_{A,S} \subseteq ACT_A$ be the set of actions A can perform in situation S.*
Then, $f_A : 2^{ACT_{A,S}} \longmapsto ACT_{A,S}$ is a random choice function if f_A is equally distributed, i.e. if for each $a \in ACT_{A,S}$, $p(f(ACT_{A,S}) = a) = \frac{1}{|ACT_{A,S}|}$.

9.4.2 Weighted Randomness

A shortcoming of random strategies is that they are uninformed, i.e. they do not use any knowledge about the world to direct the choice of an agent. By weighting the random function according to the potential usefulness of an action according to the current situation and the current goals of an agent, a more goal-directed acting of an agent may be achieved while the advantages of random interaction methods, such as their nondeterminism, are preserved. For this purpose, a dynamic weighting function has to be introduced which determines the probability with which an alternative is selected from the set of alternative actions an agent is able to execute in a certain situation.

Definition 4 (Weighted Random Choice Function) *Let A be an agent, ACT_A be the set of actions which can be performed by A. Let $\mathcal{S}$ be the set of possible situations, $\mathcal{G}$ the set of possible goals. Let $S \in \mathcal{S}$ be the current situation, $G \subseteq \mathcal{G}$ the set of current goals of the agent.*

a) $\omega : \mathcal{S} \times 2^{\mathcal{G}} \times ACT_A \longmapsto [0,1]$ is a weight function if $\sum_{a \in ACT_A} \omega(a) = 1$.

b) $f_A^\omega : 2^A \times \omega \longrightarrow ACT_A$ is a weighted random choice function with weight function ω if f_A^ω is ω-distributed, i.e. if for each $a \in ACT_A$, $p(f(ACT_A) = a) = \omega(S, a)$.

A random choice function f_A can be regarded a special case of a weighted random choice function with $\omega(a_i) = \omega(a_j)$ for each $a_i, a_j \in ACT_{A,S}$, $\omega(a_k) = 0$ for $a_k \in ACT_A \backslash ACT_{A,S}$.

Due to the knowledge which is implicit in the weight function, results achieved using weighted random methods are clearly superior to results using simple random functions.

Interaction strategies based upon weighted random moves do not suffer from the drawbacks of hill-climbing strategies such getting caught by local optima (cf. next subsection).

However, apart from the problems of using random methods in general for modeling intelligent agents mentioned above, there are two main objectives against using this kind of interactive methods. Firstly, maintaining and adjusting the weight function is a non-trivial task, since it requires knowledge about what are good and what are bad alternatives to take. Secondly, encoding all the problem-solving knowledge into a single function seems no good AI style. Small changes in the domain may require redesigning the weight function from scratch.

Therefore, in a next step, we introduce heuristic methods which make explicitly use of the toplogical knowledge of the agents.

9.4.3 Heuristic Methods

The use of heuristic rules and methods in order to come to a decision is one of the basic problem-solving techniques in classical AI [Win84, RK91]. Many of these techniques fit into our concept of behaviour-based concepts since they define the behaviour of the agent in direct dependence on its local knowledge, on the external situation, and on its mental context. In this section, we present two different kinds of these heuristics, namely hill-climbing-like methods, and the simulated annealing heuristics. The latter are able to avoid the typical drawbacks of hill-climbing by allowing moves that lead to a temporary deterioration of the agent's situation.

The Gradient Field Method

Using the gradient field method for making the agents' behaviour goal-directed has been proposed by Steels [Ste90]. In robotics, it is well-known as a model for robot path planning (see also [Lat92], where it is named *potential field method*). Indeed, this method is a standard search strategy in AI where it is also known by the name "steepest-ascent hill climbing"[RK91]. Actually, it is a deterministic version of the weighted random choice method described above. In any situation, an agent follows the steepest ascent of a given gradient function. Whereas by dropping the random part, the gradient model is appropriate to explain the behaviour of an agent trying to maximize its local utility, it has quite a few serious drawbacks:

- By eliminating the random element, it suffers from the typical hill-climbing problems such as local optima, plateaux, and ridges. In order to tackle these problems, however, a combination of gradient search and random strategies may be used.
- Again, the whole knowledge of an agent is encoded in one single function, namely the gradient, which may be criticized.
- Computing and maintaining the gradient in a dynamic environment, where also the goal an agent pursues may alter its location, is computationally expensive, since it requires the simulation of a diffusion mechanism, which has to be performed for any point in the scenario. This is a considerable overhead.

Simulated Annealing

The simulated annealing strategy [KJV83] is a variation of the gradient-method in a sense that under certain circumstances, some downward moves are allowed. This method requires the maintenance of an *annealing schedule* which is crucial for its success, and the computation of probabilities for transitions into a higher state. Thus, simulated annealing

avoids some of the drawbacks of standard hill-climbing. However, it is nothing more than a heuristic and is not safe from the problems which are also inherent to the above approaches.

9.4.4 Discussion

In conclusion, according to our experience, the behaviour-based methods pursued up to now suffer from two main drawbacks, the *lacking cognitive adequacy* and the *bounded scope*. The problem of lacking adequacy means it is not intuitive that the criteria of these functions are appropriate to model the complex decision processes and preference structures in an intelligent agent. The bounded scope of these methods seems to be a very serious problem if the methods are used to achieve cooperative behaviour. However, where the behaviour of agents is not too complex, and where no sophisticated mechanisms of interaction occur, the methods under consideration up to now offer a convenient way to describe action and interaction of agents. In the following section, we will present the modeling of the plan-based component of the forklift agents.

9.5 Plan-Based Modeling

9.5.1 Single-agent Plans

A (single-agent) plan can be described by a set of plan steps and a set of relationships (constraints) among these plan steps. The type of constraints we will restrict ourselves to for single-agent plans are precedence constraints describing that some plan steps have to be executed before others. However, in the general framework, arbitrary predicates among the plan steps are allowed.

Definition 5 (Single-Agent Plan) *A single-agent plan P is defined as $P = (S, C)$, where*

- $S = \{s_1, \ldots, s_n\}$ *is a set of individual plan steps.*
- $C = \{c_1, \ldots, c_k\}$ *is a set of predicates, denoting the constraints among elements of S, $c_i \subseteq S^j$, $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, n\}$.*

Representation of Single-Agent Plans; the Plan Language $\mathcal{P}_0$ Single-agent plans are represented by sequences of plan steps. The sequence represents the precedence constraints among the plan steps in the single agent plan.

Definition 6 *Let a be an agent. Let $S = \{s_1, \ldots, s_n\}$ be a set of primitive plan steps. Then a plan P_a for agent a is represented as $P_a = (Name, Type, Body)$ with*

- `Name` *denotes the reference name of the plan.*
- `Type` $\in \{s, b\}$ *denotes the execution type of the plan step (b stands for "executable pattern of behaviour", s stands for "(skeletal) plan".)*
- `Body`$\in \mathcal{P}_i$ *is a well-formed plan body with respect to a plan language $\mathcal{P}_i$.* `Body` *denotes the actual description of the plan steps.*

In the following, we will define the plan language $\mathcal{P}_0$ which describes admissible bodies of single-agent plans. $\mathcal{P}_0$ is a propositional plan language; it does not permit the use of individual variables.

Definition 7 ($\mathcal{P}_0$) *Let $S = \{s_1, \ldots, s_n\}$ be a nonempty set of primitive plan steps. The plan language $\mathcal{P}_0^S$ is defined as follows:*

- $- \in \mathcal{P}_0^S$ *(empty plan body).*
- $s \in \mathcal{P}_0^S$ *for a primitive plan step* $s \in S$*.*
- *Let* $s_1, s_2 \in \mathcal{P}_0^S$*. Then* $s_1, s_2 \in \mathcal{P}_0^S$ *(sequential composition of plan steps).*
- *Let* $s_1, s_2 \in \mathcal{P}_0^S$*. Then* $s_1; s_2 \in \mathcal{P}_0^S$ *(disjunctive composition of plan steps).*
- *Let* $s_1, s_2 \in \mathcal{P}_0^S$*, let* e *be an arbitrary predicate. Then* **if** e **then** s_1 **else** $s_2 \in \mathcal{P}_0^S$ *(conditional branch).*
- *Let* $s \in \mathcal{P}_0^S$*, let* e *be an arbitrary predicate. Then* **while** e **do** $s \in \mathcal{P}_0^S$ *(while-loop).*

We will use single-agent plans of type $\mathcal{P}_0$ as the basic building blocks in order to model the basic plan steps of joint plans in section 10.2.2.

9.5.2 Planning in the Loading Dock

At the plan-based layer of a forklift agent, the plans for the complex transportation tasks performed by the agent are defined. There are two top-level tasks a forklift agent can execute: firstly, it can load a truck, i.e. fetch a box from a shelf, carry it to the truck, store it on the truck, and return to the forklift's wating zone. Secondly, it can unload a truck, i.e. fetch a specified box from the truck, carry it to the corresponding shelf, store it in the shelf, and again return to the waiting zone. As soon as the agent receives such a task, a pattern of behaviour is activated, and thus, the agent adopts the goal of performing the task. The agent's *plan library* contains skeletal plans for achieving the transportation goal. In chapter 11, an example is provided which uses a slightly simplified representation of the representation of the goals.

Apart from the ability to exploit the presence of skeletal plans stored in a plan library, the forklift agent is equipped with elementary modeling facilities. These help the agent cope with situations for which neither a skeletal plan nor an executable pattern of behaviour are specified. In our implementation, modeling means the from scratch generation of new plans. Currently, this feature is available only for small subgoals of an agent, such as for example for planning dynamically how to move from one landmark to another. For this purpose, apart from a pattern of behaviour, a set of plan steps are generated dynamically which describe how the goal of reaching a landmark is splitted in a sequence of goals describing moves between fields.

At a later stage of the development of the system, the crude modeling facilities provided by the INTERRAP model so far can be extended by more sophisticated mechanisms for model generation and for describing the agent's focus of attention[7].

[7] Ferguson has even joined predictive and reflective modeling facilities of an agent in a seperate layer of the agent architecture [Fer92]. In contrast to this, INTERRAP associates modeling facilities with different layers of the agent, namely with the plan-based layer, as far as local modeling is concerned, and with the cooperation layer, as far as the treatment of conflicts is concerned.

Chapter 10

Agent Interaction

There are several interesting aspects of interaction among agents in the loading dock. Most of the interaction taking place in this scenario are of physical nature, i.e. interactions which are caused by the movements of the forklifts in the loading dock. Therefore, the situational context dominates the description of most of the patterns of interaction, e.g. the ones for avoiding collisions or for resolving blockings. However, since the forklifts have goals to achieve, also the goal context has to be taken into consideration. This will be outlined in the following by means of a small example.

10.1 An Example

In figure 10.1, two typical situations of interaction between two forklift agents are shown, as they occur in the loading dock. In figure (10.1.1), there is a potential frontal collision

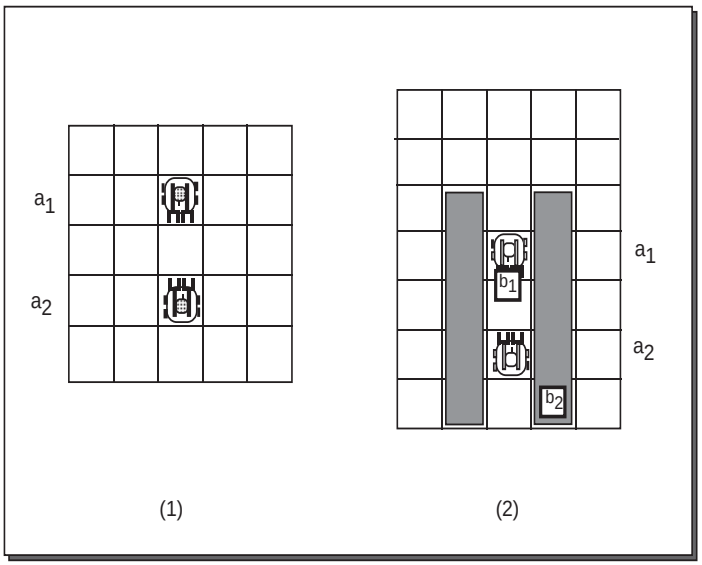

Figure 10.1: Interaction in the Loading Dock

between two forklifts. Figure (10.1.2) shows a typical deadlock situation caused by goal conflicts: agent a_1 wants to take box b_1 and blocks agent a_2's way out of the shelf area. a_2 has just finished his task to put box b_2 to the shelf. In both situations, behaviour-based as well as plan-based mechanisms of interaction may be used. However, compared to situation (10.1.1), situation (10.1.2) seems less suited to be tackled by behaviour-based methods such

as making *random moves*, since a solution to the conflict requires taking the goals of the agents into consideration. For example, agent a_1 wants to reach a field behind agent a_2 in order to put a box, there. So why should a_1 move out of the corridor before it has reached its goal? The fact that both agents strive after minimizing their local costs (which might be given by the number of actions they perform in order to reach a goal) often requires the use of mechanisms for negotiation and persuasion. During this negotiation, a joint plan for the agents can be devised. For example, a clever solution for situation (10.1.2) is a cooperative one: agent a_2 should do the job of transporting box b_1 to the shelf for a_1. On the other hand, negotiating a joint plan for collision avoidance in the first case seems like using a sledgehammer to crack a nut.

10.2 Mechanisms of Interaction

In the loading dock domain, agents use several both plan-based and behaviour-based methods to coordinate their activities. In the following, we will provide a short sketch of these mechanisms. For a more detailed discussion, we refer to [Mül93].

10.2.1 Behaviour-based Mechanisms of Interaction

Behaviour-based methods of interaction are methods which do not require devising a plan. In most cases, these methods have local nature, i.e. they do not involve communication between agents. However, as we will see, behaviour-based "public" mechanisms exist. Basically, there are two behaviour-based methods of interaction used in the loading dock domain, randomness and what we call "spontaneous communication". These will be discussed in the following.

Randomness

The use of randomly chosen actions has been discussed before in the context of the description of the behaviour-based component. Also when it comes to deal with interactive situations, i.e. situations where the activites of several agents have to be coordinated, randomness is an important tool. Random actions are used in the loading-dock in order to resolve blocking conflicts among forklifts, if other, more sophisticated methods - such as the generation of joint plans which is described in the following section - fail. Randomness has turned out to be very useful as a means of coordination in symmetrical situation (see figure 10.1 for an example). As first results show, randomness should *not* be employed as the default technique when programming the forklift agents. For example, whereas three forklifts started simultaneously in the scenario can perform their tasks (each should load a box to the truck) within a few minutes (depending on their a priori knowledge) if they are equipped with planning facilities, in the case of agents which are only moving randomly it took half an hour - until the first of them had successfully finished its task.

Spontaneous Communication

By spontaneous communication, we mean communication between agents which is triggered by simple situative and/or mental patterns, and for the enforcement of which there is not much deliberation and refelction necessary. In this section, we will describe two applications of spontaneous communication between agents: its use for a goal-driven exchange of information, and for conflict resolution.

Exchange of Information A further behaviour-based mechanism of interaction that is used by the forklift agents has been mentioned in the above section 9.3. In figure 9.6, a pattern of behaviour was shown that describes how an agent could provide another agent with information upon request. On the other hand, a pattern of behaviour `query_box_info` is implemented which allows an agent who has a transportation task and who searches a box to ask another agent whether that one knows the box. The definition of the pattern `query_box_info` is shown in figure 10.2. Since the pattern is triggered by the presence

(PoB
:name $query\_box\_info$
:static $social$
:participants $self; A : agent$
:sit_context $Perc(agent(A,(X,Y))$
:mental_context $Goal(self, search\_box(B)) \wedge \neg Bel(self, curr\_pos(B,(\_,\_)))$
:postcond $message\_received(A, inform(location(B,(X,Y))) \Longrightarrow Bel(self, curr\_pos(Box,(X,Y)))$
:termcond $message\_received(A, inform(location(B,(X,Y))))$
:failcond $message\_received(A, inform(dont\_know(location(B))) \vee timeout$
:exec_descr send the other agent a query $query(location(B))$ and wait for an answer */
)

Figure 10.2: Pattern of Behaviour *query_box_info*

of another agent, we would prefer to classify it as a behaviour-based one, although the mental context of the agent also plays a role - the pattern is only activated if the agent has the goal of searching a box and does not know the location of the box. Note that the second part of the mental context - that the agent's knowledge base does not contain a belief of the form $Bel(self, curr\_pos(B,(\_,\_)))$ in fact requires the agent to draw knowledge base inferences. Therefore, we could say that the pattern `query_box_info` is not purely behaviour-based. However, since the inferential abilities of our agents are currently restricted to simple knowledge base matching (see chapter 3.3 for a discussion of this point), the agent can check very quickly whether it believes in a certain fact. Thus, we can argue that the pattern is rather behaviour-based than requiring actual deliberation.

Conflict Resolution A second application of spontaneous communication is employed for conflict resolution. In the case of a blocking conflict that cannot be resolved by simply applying random moves, before agents get involved in a possibly lengthy negotiation process, they try to resolve the conflict by giving signals to each other, which we regard as a special form of spontaneous communication. An agent who realizes that there is a blocking conflict with another agent first sends this agent a request `go_away`[1] . This is clearly behaviour-based, since the communicative act is triggered by the existence of a blocking conflict.

The recipient of the `go_away` message checks whether moving away from its current position would clash with its local goals. If this is not the case, the agent will move away from the field; otherwise, it will reject the request. Note, however, that the process of interpreting the `go_away` request and checking whether there are possible goal conflicts can be a very complex one. Thus, the term "behaviour-based" is suitable for representing the action of sending a signal, but not for characterizing its interpretation.

[1] In a road traffic scenario, this could be reached by a horn signal. However, since the semantics of that signal is *"Watch out! I have the goal to get to the place where you are standing!"*, we prefer using a speech act in this case.

10.2.2 Plan-based Interaction via Joint Plans

In this section, we will develop the concept of *joint plans* as a means of deliberate interaction among autonomous agents. We will outline how agents can agree on the execution of a joint plan by a process of negotiation. Due to reasons of space, we will only give a brief overview of the subject. A detailed definition of joint plans and joint plan negotiation can be read in [Mül93].

Joint Plans

Intuitively, joint plans are plans that may be devised and/or executed by several agents. Depending on the complexity of the underlying plan language, joint plans can have very different shapes: in the most simple case, they are defined as sequences of actions which are labelled by the name of the agent who performs the respective action. Extensions of this language can be made by allowing non-linear plans (see e.g. [Sac75] for the case of single-agent plans), by introducing variables, by specifying roles for of agents, and by allowing constructs for parallelism in multi-agent plans. In [Mül93], we define plan languages which contain these extensions.

A single-agent plan consists of a set of plan steps and of a set of constraints among these plan steps (see chapter. A multi-agent plan extends this theory by a set of agents which label the plan steps, and by providing a set of constraints among the actions of different agent. A joint plan JP can be formally defined as follows:

Definition 8 (Joint Plan) *A joint plan is a quadruple $JP = (A, S, F, C)$, where*

- $A = \{a_1, \ldots, a_m\}$ *is a set of agents.*
- $S = \{s_1, \ldots, s_n\}$ *is a set of plan steps.*
- $F : A \rightarrow 2^{M(S)}$ *is a function which maps each agent in A to the subset of plan steps in S executed by that agent. Since elements of S may occur multiply in one plan, the range of F is the multiset M of S.*
- $C = C_l \cup C_g$ *is a set of predicates denoting the plan constraints. The components of C are interpreted as follows:*
  - $C_l = C_{a_1} \cup \ldots \cup C_{a_m}$ *is the set of local constraints among the plan steps of $a_1, \ldots, a_m$. $C_{a_i} \subseteq S^k_{a_i}$, $k \leq |S_{a_i}|$.*
  - $C_g = \{C_{g1}, \ldots, C_{gm}\}$, $m \geq 1$, $C_{gi} \subseteq S_{a_i} \times \ldots \times S_{a_j}$, $1 \leq i \leq j \leq m$ *is a set of global constraints, i.e. constraints between plan steps of different agents $a_i, \ldots, a_j$.*

Representation of Joint Plans

In this section, we present the representation of joint plans used in the loading dock scenario. Definition 9 shows the general structure of a joint plan

Definition 9 *Let $A = \{a_1, \ldots, a_k\}$, $k \geq 1$, be a set of agents. Then a joint plan JP is represented as $JP = (Name, Type, Body)$ with*

- `Name` *denotes the reference name of the plan.*
- `Type` $\in \{s, b\}$ *denotes the execution type of the plan step (b stands for "executable pattern of behaviour", s stands for "(skeletal) plan".)*
- `Body`$\in \mathcal{P}_i$ *is a well-formed plan body with respect to a plan language $\mathcal{P}_i$.* `Body` *denotes the actual description of the plan steps.*

Note that the plan language that describes the plan body is not prescribed by definition 9. Rather, it can be provided as a parameter.

We start the definition of our plan language with the set $S = \{s_1, \ldots, s_k\}$ of *primitive plan steps*. The $s_i \in S$ are atomic propositional formula. In the following, we define the exemplary plan language $\mathcal{P}_2$. It defines one step of a joint plan as a list of single agent plans, one plan being provided for each agent participating in the execution of the joint plan. A language for bodies for single-agent plans has been specified by definition 7. For the case of joint. multi-agent plans, we extend this basic language by an additional language construct | which allows us to represent plan steps which are executed simultaneously by two agents[2]. This leads to the definition of plan language $\mathcal{P}_0'$:

Definition 10 (Plan Language $\mathcal{P}_0'$) *Let $a_i, a_j \in A$ be agents. Let plan language $\mathcal{P}_0$ be according to definition 7. Then, the plan language $\mathcal{P}_0'$ is defined recursively as follows:*

- $s \in \mathcal{P}_0 \longrightarrow s \in \mathcal{P}_0'$.
- *Let $s, t \in \mathcal{P}_0'$. Then, $s_i|t_j \in \mathcal{P}_0'$ for agents a_i, $a_j \in A$ (simultaneous composition of plan steps).*

$s_i|t_j$ denotes a composite plan step which as defined by agents a_i and a_j simultaneously executing the plan steps s and t, respectively.

Now, we can define plan language $\mathcal{P}_2$ for joint plans. The semantics of one step of a joint plan is that it describes sequences of actions which can be performed *concurrently* by the different agents taking part in the plan.

Definition 11 ($\mathcal{P}_2$) *Let $A = \{a_1, \ldots, a_m\}$ be a set of agents, $S = \{s_1, \ldots, s_n\}$ be a set of primitive plan steps. The plan language $\mathcal{P}_2$ is defined as follows:*

- $[] \in \mathcal{P}_2$ *(empty plan step).*
- *Let $l_1, \ldots, l_m$ be plan bodies of type $\mathcal{P}_0'$, so that for each l_i, $i \leq m$*
  - *agent a_i performs plan body l_i (i.e. all plan steps in l_i are labelled by agent a_i, or*
  - *l_i contains only plan steps labelled by agent a_i* and *simultaneous composition plan steps $s_1 \| s_2$. In the latter case, agent a_i performs either s_1 or s_2.*

  Then $[l_1, \ldots, l_m] \in \mathcal{P}_2$

The separation between two plan steps expresses that there are precedence constraints among these plan steps (see also the example in chapter 11).

Operationalization

Joint plans are operationalized in the cooperation component (see section 4.2.5). The component itself contains a joint plan generator which can devise joint plans from scratch. Moreover, a library of joint plans for standard situations is contained in the cooperation knowledge level of the hierarchical KB. An example for this can be found in chapter 11. Since the joint plans cannot directly be executed by the plan-based component, they have to be transformed into a sequence of single-agent plan by the *joint plan translator* component of the CC (see chapter 4.2.5). For details of this process, we refer to [Mül93]. Agents agree on a joint plan for a certain pattern of interaction by a process of negotiation. This is described in the following section.

[2]The operator | can be easily extended to plan steps executed simultaneously by $n > 2$ agents.

Negotiation on Joint Plans

Since we deal with autonomous agents, the agreement on a joint plan has to be reached by an iterative process of negotiation. The process of devising and executing a joint plan is described as a process in six phases, namely

1. The initiation and information phase.
2. The plan generation phase.
3. The plan negotiation phase.
4. The plan confirmation phase.
5. The plan execution phase.
6. The execution evaluation phase.

After the subject of negotiation has been agreed on and the goals of the agents which are affected have been exchanged (phase 1), plan negotiation itself starts with the generation of an initial plan (phase 2). The proposed plans are evaluated by the agent (using the plan evaluation module in the plan-based component, see chapter 4.2.4). Plans can be rejected, accepted, modified, and refined until either a mutually agreed on plan has been devised, or until no alternatives are left. The speech acts used for negotiation are the ones described in [SS93].

An additional quality can be gained by splitting up the plan negotiation phase in three subphases; during the first phase, agents agree on an uninstantiated skeletal plan, i.e. on what is to be done in general. In phase two, the allocation of roles is negotiated (who shall do what?). Finally, phase three leads to an agreement on the framework conditions such as cost and time conditions. Backtracking between these phases may occur. This structuring of the plan negotiation process helps reducing the high complexity of plan evaluation. As we know from human joint planning (such as project planning), it is often useful to know what a reasonable plan looks like, even before it is clear who will take which role in that plan. However, this kind of negotiation requires a plan language of type $\mathcal{P}_4$ which provides variables for roles and objects (see [Mül93]).

Chapter 11

An Example

labelexample In this chapter, we will explain by means of an example how the concepts presented in this report allow the InteRRaP agent to cope both with its routine tasks and with unforeseen events in an intelligent and flexible manner. For this purpose, let us consider the following example: an agent, say `agent`$_i$, gets an order to load a truck, say t_1 with a certain box, say b_{23}. This happens by sending the agent a request message

`request(agent`$_i$`, load_truck(`t_1`,` b_{23}`))`.

In the following, we will provide a trace of planning and plan execution achieved by the InteRRaP model.

11.1 The Plan Library

Let us first consider what the plan library of the agent looks like. In section 4.2.4, we defined the library by a list of entries which can be referenced by a name of a goal. Figure 11.1 shows a fragment of the library for the current goal. In the example, the plan library

```
lpb_entry(load_truck(T, B), s,
        [do(fetch_box(B)), do(store_box(B, T))])
lpb_entry(fetch_box(B), s,
        [rr(box_position(B, ?Pos)), do(goto_landmark(Pos)), do(get(B))]).
lpb_entry(fetch_box(B), b,
        [pob(random_search(B))]). ;; pattern of behaviour
        ...
lpb_entry(goto_landmark(L1), b,
        [pob(goto_landmark_beh(L1))])
lpb_entry(goto_landmark(L1), s,
        [rr(where_am_i(?L0)), do(gen_moves(L0, L1))])
        ...
```

Figure 11.1: Exemplary Plan Library

embodies a very simple plan language which represents a single-agent plan as a sequence of subplans, resource requests, and patterns of behaviour. The plan library specifies a tree-like plan structure. Note that for several plan steps, there exists more than one possibility to continue. For example, the plan step `goto_landmark` can be treated either by further planning (basically by generating a list of moves from one field to the next), or by activating a pattern of behaviour `goto_landmark_beh` which can be directly executed.

11.2 Performing Routine Tasks: Trace of Planning and Execution

Figure 11.2 shows the overall processing of the order request by agent i. The original request

`request(agent`$_i$`, load_truck(`t_1`, `b_{23}`))`

is handled by a pattern of behaviour `treat_order_beh` located in the BBC. Due to the limited space, we will not explain the structure of the patterns of behaviour and the way they work in more detail. Let us here assume that the pattern of behaviour recognizes that the whole goal should be treated by a planner[1]. Thus, as shown in figure (11.2.1), the BBC sends a `request(pbc, do(load_truck(`t_1`, `b_{23}`)))` to the PBC. Now, the planning process inside the PBC starts (see below for a more detailed description). Since the PBC has been called by a `do` command, plans are (1) devised and (2) their execution is monitored by the planner.

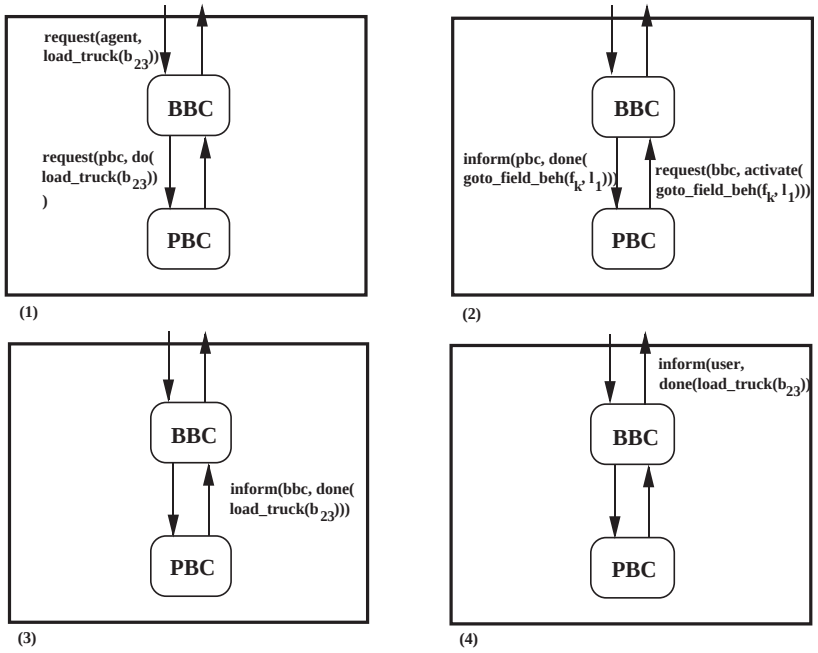

Figure 11.2: Example: Processing the `load_truck` Goal

This monitoring is shown exemplarily in figure (11.2.2): whenever the planner has found an executable pattern of behaviour or a primitive action (see section 4.2.4), it activates the corresponding pattern of interaction. In the example, the pattern of behaviour for moving to a certain landmark is activated by sending a message

`request(bbc, activate(goto_field_beh((7, 4)))).`

The pattern of behaviour is executed and reports its (successful) termination by sending a message

`inform(pbc, done(goto_field_beh((7, 4))))`

to the PBC.

[1] In fact, the behaviour could call the planner only for a part of the goal.

Finally, the PBC has planned and processed the whole `load_truck` task. It reports this to the calling pattern of behaviour `treat_order_beh`. This is shown in figure (11.2.3). Control shifts back to the BBC, and, as displayed in figure (11.2.4), a message is sent to the user denoting that the task has been successfully completed.

Up to now, we have seen the collaboration between PBC and BBC from a global point of view. In the following, we will explain the processing inside the PBC in more detail. Figure 11.3 shows the trace of the planning process for the goal `load_truck(`t_1`, `b_{23}`)`.

```
/* In the following, PC means planning control, PE plan evaluator, PG plan generator, RH resource handler */
/* ?X means that variable X is output variable.  curr. goal shows the top of the goal stack.  */

BBC → PBC:      request(pbc, do(load_truck(t1, b23))).
curr. goal (PC): do(load_truck(t1, b23)).
PC → PG:        make_plan(load_truck(t1, b23), ?Planlist).
PG → PC:        Planlist = [[do(fetch_box(b23)), do(store_box(t1, b23))]].
curr. goal (PC): do(fetch_box(b23)). /* Evaluation is omitted, since no alternative plans */
PC → PG:        make_plan(fetch_box(b23), ?Planlist).
PG → PC:        Planlist = [[rr(box_position(b23, ?Pos)), do(goto_landmark(Pos)), do(get(b23))],
                            [pob(random_search(b23))]].  /* Planlist = [p1, p2] */
PC → PE:        evaluate([p1, p2], ?EvPlans).
PE → PC:        ?EvPlans = [(p1, 10), (p2, 3)].  /* Plan p1 has higher utility than plan p2 */
curr. goal (PC): rr(box_position(b23, ?Pos)).  /* First subgoal of p1 */
PC → RH:        box_position(b23, ?Pos).
RH → PC:        Pos = (7, 4)).
curr. goal (PC): do(goto_landmark((7, 4))).
PC → PG:        make_plan(goto_landmark((7, 4), ?Planlist)).
PG → PC:        Planlist = [[pob(goto_landmark_beh((7, 4)))], [rr(where_am_i(?ActPos)),
                            do(gen_moves(ActPos, (7, 4)))]]. /* Planlist = [p3, p4] */
PC → PE:        evaluate([p3, p4], ?EvPlans).
PE → PC:        ?EvPlans = [(p3, 12), (p4, 5)].  /* Plan p3 has higher utility than plan p4 */
curr. goal (PC): beh(goto_landmark_beh((7, 4))).  /* Pattern of behaviour has been selected */
PBC → BBC:      request(bbc, activate(goto_landmark_beh((7, 4)))).
        ...
BBC → PBC:      inform(pbc, done(goto_landmark_beh((7, 4)))).
        ...
PBC → BBC:      inform(bbc, done(load_truck(t1, b23))).
```

Figure 11.3: Example: the Interplay between BBC and PBC

11.3 The Handling of Unforeseen Events

So far, we have seen how INTERRAP agents can perform their tasks using plans and patterns of behaviour, and how the use of these concepts can be combined to yield an appropriate overall behaviour of the agent. In chapter 6, we discussed issues of dynamic planning in the INTERRAP model. In this section, we will provide a detailed example for how the INTERRAP model supports dynamic planning, i.e. what makes the model especially suited to tackle highly interactive and dynamic domains such as the robotics domains described in [MP93]. In the following, this issue will be informally discussed by means of the above example.

Let us have a look at the *goto_landmark* plan step in the above example. As is shown in figure 11.1, agent *i* has two alternatives for reaching a given landmark. Firstly, it can use a pattern of behaviour, i.e. at a certain level of abstraction, this is a routine task for the agent which does not require deep reflection. Secondly, the agent can further decompose the plan step by first determining its actual location and then planning a list of moves the last of which leads to the goal landmark (this is accomplished by the predicate `gen_moves`).

The choice the agent makes between these two alternatives is crucial for how it can master the task. On the one hand, if it chooses the plan, after a brief planning time, it can walk straight ahead to the goal landmark. A pattern of behaviour for this task is likely to be less goal-directed; for example, it may use a weighted random function[2]. However, let us now assume that another agent, say j, blocks the way of agent i, and let us first consider the case that the agent has chosen the plan-based processing of the `goto_landmark` subgoal. In this case, i notices that a specific plan step `goto_field(From, To)` cannot be executed, since field `To` is not free. As a consequence of this, the PBC has to be activated again. The original plan must be modified, an alternative way has to be planned, or a negotiation with agent j is initiated where a joint plan to resolve the conflict is devised. This is, of course, a very costly and complex process considering the fact that also the new plan is likely to be interrupted.

What we would prefer is to shift some intelligence into the execution. This is done by defining a pattern of behaviour `goto_landmark_beh` which *somehow* finds a way to the goal landmark, and which is able to handle some exceptional situations from within it. For example, while walking straight ahead, a common exceptional situation is that another agent blocks the way. Thus, the behaviour can cope with this situation by trying to take a square aside randomly. In most cases, applying this strategy will be sufficient to resolve the conflict. What is important is that the pattern of behaviour is able to recognize situations it cannot cope with, i.e. it has *self-monitoring capabilities*. In these situations, for instance if two agents block their ways in a narrow shelf, the BBC calls the PBC[3] by sending a request message

```
do(resolve_shelf_conflict((self, Agent)))
```

to the PBC. The *pbc_control* recognizes that solving a blocking conflict requires a joint plan. Therefore, it calls the cooperation component. This one devises a joint plan, and initiates a negotiation with the other agent on the join plan. In the simplest case, negotiation may consist of directly accepting the first plan proposed. However, an iterated process of plan modification and plan refinement may be necessary in order to come to a mutually accepted plan. The resulting joint plan is translated by the CC in a single-agent plan which is augmented by synchronization commands. For example, as already shown in chapter 6, a very simple joint plan for two agents i and j changing places is

```
JP = [[[walk_aside(i, north)], []],
      [[walk_ahead(i)], [walk_ahead(j)]],
      [[walk_aside(i, south)], []]].
```

The plan `JP` has three phases. Phase describes that agent i must first walk aside. In phase two, i and j may concurrently walk ahead. Phase three, where agent i steps to the other side again, can only start after agent j has executed the `walk_ahead` action. This joint plan is translated into the following single agent plan P for agent j:

```
P = [walk_aside(north),
     send_synch(j, ready),     ;; send message to agent j
     walk_ahead,
     on receive(j, ready) do  ;; wait until j has done walk_ahead
        walk_aside(south),
        send_synch(j, ready)].
```

[2]We use randomness as an important means to keep the system deadlock-free.

[3]This is what we mean by *competence driven* activation.

The synchronization commands ensure that the precedence constraints expressed by the joint plan are respected in the execution. This plan is passed to the PBC which executes it by again activating appropriate patterns of behaviour. Figure 11.4 provides an overview of how the shelf conflict situation is processed by shifting control and sending messages between the behaviour-based, the plan-based, and the cooperation component, starting from the realization of the situation to the execution of the mechanims for conflict resolution that has been chosen by the cooperation component of the agent. In figure (11.4.1), the conflict

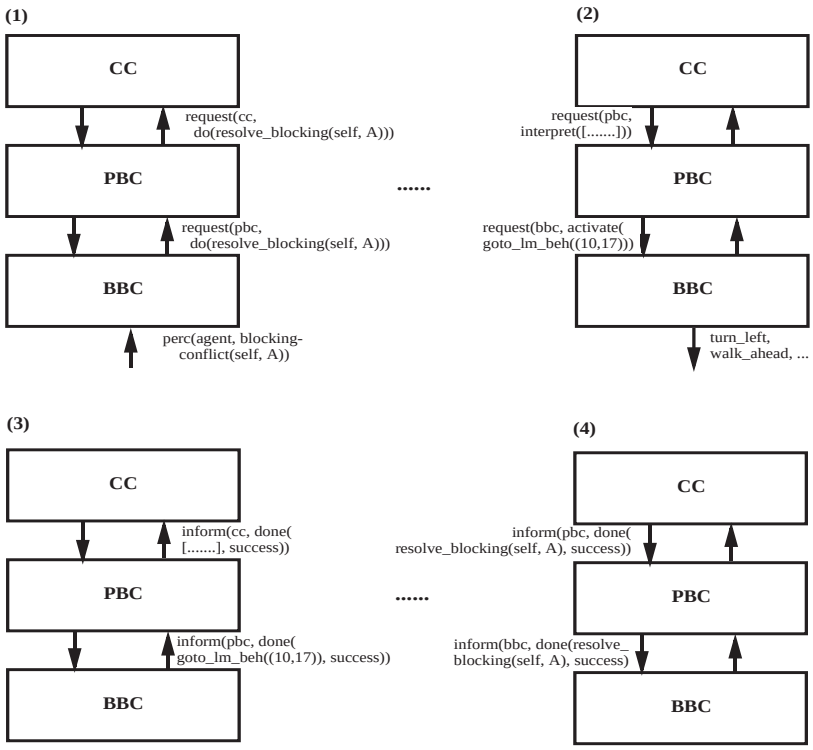

Figure 11.4: Example: Resolution of a Conflict Situation

is recognized by pattern of behaviour in the BBC. Control is passed to the PBC which recognizes that solving the conflict is beyond its capacities, and shifts control up to the CC. The CC generates a joint plan, and passes this plan to the PBC in the form of a sequence of synchronized single agent plans. Figures (11.4.2) and (11.4.3) show how one such plan (which may also be a partial protocol) is processed by the PBC. The PBC interprets the plan, and, if the current plan step is an executable pattern of behaviour, activates the BBC which executes the behaviour (by sending messages to the world interface - this has been omitted in the figures) and reports success or failure to the PBC. This continues until the plan is processed which is again reported to the CC by the PBC. Finally, as shown in figure (11.4.4) the whole joint plan is processed. The CC has performed its original goal, and shifts back control to the PBC, then from there to the BBC. The conflict has been resolved, and the agent can keep on with performing its standard task - loading or unloading the truck.

11.4 Discussion

In conclusion, patterns of behaviour are flexible, but maybe less efficient mechanisms, whereas plans lead to intelligent solutions, but are likely to fail in a dynamic environment. The crucial point is to find the appropriate granularity, i.e. to find criteria for deciding what

should be done by defining a pattern of behaviour, and what should rather be performed by devising and executing a plan. Apart from intuitive criteria, which are useful but definitely insufficient, empirical results gained by testing different configurations and decision strategies in a concrete scenario will provide valuable information. We refer to the following section for a discussion of that issue.

Chapter 12

Implementation and Preliminary Results

In this section we will provide an overview of the loading dock implementation and report preliminary results we have obtained using the actual implementation.

12.1 The Test-bed

A prototype of the loading dock scenario described in section 8 has been developed in our group. Starting from two separate implementations - a purely behaviour-based agent written in MAGSY [FW92] (a multi-agent extension of OPS-5) and a purely plan-based agent written in PROLOG - we now start merging the features to a single agent which incorporates both behaviour-based and plan-based facilities. The testbed runs under UNIX on a network of SUN SPARC stations. Agents can be run distributed on any available local machine. Thus, there is no real limitation on the number of agents except the message transmission capacity of the local net.

12.1.1 The Agent

In order to reduce the process communiation overhead, we refrained from implementing each of the control modules as a physical process on its own. Rathermore, the agent is physically defined by two UNIX processes, a MAGSY process which contains the world interface and the behaviour-based component modules, and by a PROLOG process which realizes the plan-based component and the cooperation component. This partition is straightforward since there is a very tight coupling of the patterns of behaviour and the primitives for the execution of actions and for the sending of messages contained in the world interface on the one hand, and a similarly tight conceptual connection between the (local) plan-based component and the (interactive) cooperation component, on the other hand. Together, the two latter modules form the rational, deliberate part of the agent.

The communication primitives used by the agents are implemented in $C++$ on top of UNIX process communication facilities using port sockets. The underlying low-level protocol is TCP/IP. This basis allows agents written in different languages to communicate when run as processes under UNIX. What is required is only a communication layer which transforms messages from one language into a knowledge interchange language, and back from the interlingua to local language format. This allows the agent to form their messages using high-level language structs (speech acts). So far, these modules are available for MAGSY (behaviour-based part of the agent), PROLOG (plan-based part of the agent), and LISP (interface to the KRIS system which is to be used to model parts of the world knowledge

and of the cooperation knowledge). A module for the OZ concurrent constraint language [WHS93] is under development.

12.1.2 The Simulation World

The simulation world is represented by a PROLOG process. It contains the current, "objective" state of the world, and serves to determine whether actions performed by agents have succeeded or not. Moreover, it maintains the visualization of the scenario via a graphical user interface. Agents perform actions by sending messages to the world process. The world computes its new state, and acknowledges the performance of the action. Moreover, it sends to agents changes in their range of perception caused by actions they or other agents have performed.

12.2 Preliminary Experiences

In this section, we describe a series of tests we did using our experimental testbed. Section 12.2.1 describes the experimental setting. In section 12.2.2, the results of the experiment are provided. They are discussed in section 12.2.3.

12.2.1 The Experimental Setting

The aim of the experiment we did was to find out how agents having different facilities and abilities can perform certain kinds of tasks in the loading dock, and how the performance of the agents was influenced by varying their number. Since the physical coupling between the MAGSY and the PROLOG part is not yet ready to use, we performed our experiments using the MAGSY part of the agent, and allowed the MAGSY agent - in some configurations - to use a landmark-based algorithm for path planning written in C. The parameters of the experiment are explained in the following.

The Statical Setting: In the loading dock, there are shelves of different types ($\{red$, $blue$, $yellow\}$), and a set of boxes distributed in the shelves or on the truck. Boxes have types corresponding to the shelves. In the experiment, we do not address the boxes by their identities but by their colour. For example, an agent receives the task of loading a blue box onto the truck. This means that it can pick up the first blue box it sees. The loading dock itself has a size of 15×20 squares. There is one truck and six shelves of size 2×6 squares, two of which are red, blue, and yellow, respectively.

The experiment was run on six SUN SPARC stations at the maximum (in the case of nine forklifts).

Number of Agents: We carried through our experiment with three different numbers of agents, namely 3, 6, and 9 agents. Each of the agents has a range of perception of one field, i.e. it can perceive the square just in front of it. Moreover, agents in our experiment do not make use of their facilities of gaining information about the location of boxes by communicating with other agents. This will be subject to a later series of tests.

Agent Configuration: We experimented using five different types of agents. These are explained in the following.

Type 1: Complete Static and Dynamic Knowledge Type 1 agents have complete knowledge about the *initial* state of the world. This includes the position of shelves and trucks as well as the knowledge where the boxes stand in the scenario. However, the dynamic

knowledge is updated only by perception. Thus, if agent a_1 believes that a blue box is at square $(12, 16)$, but agent a_2 has removed the box from there, then a_1 will not realize this until it reaches the field and preceives that the blue box is no longer there. Type 1 agents use a path planner in order to find their way to the place where they believe there is a box. If they do not find a box there, they plan their way to where they believe there might be another box. Finally, if they do not find a box, at all, they start an exhaustive search through all appropriate shelves.

Type 2: Complete Static Knowledge In contrast to type 1 agents, type 2 agents have static knowledge, but no initial knowledge about the location of boxes. Therefore, all an agent can do if it receives a task "load a blue box on the truck", is going to a blue shelf and starting an exhaustive search (see the pattern of behaviour *region_search* defined in chapter 9) until it has found the box. For finding their way to the shelves, agents use a path planner.

Type 3: Weighted Randomness Type three differs from type two in that it does not employ a path planning algorithm for reaching a goal poin. Instead, it uses the weighted randomness method presented in chapter 9. Thus, it choses its next action so that it gets nearer to its goal with a high probability (75%) in the experiment.

Type 4: Weighted Randomness with Curiosity Type 4 agents are like type three agents, with the difference that they are curious. If they discover an interesting region, they start searching in this region (see also the pattern of behaviour *explore_region*). In the experiment, a shelf is regarded an interesting area.

Type 5: Random Walker Finally, type 5 agents are random walkers. These agents start without having any knowledge about the scenario, but are able to perceive the world, to memorize what they see, and to construct a model of the world. Thus, the longer they are moving through the scenario, the more they perceive, and the more efficient methods they can use. Note that the curiosity feature is very important in order to improve the performance of type 5 agents, since it allows them to explore interesting regions much faster. In our experiment, however, agents remain random walkers - they keep on making random moves until they have found the box[1].

Task Characteristics: We did the experiment with two different load characteristics. In the first case, the position of the boxes and the transportation tasks were randomly generated, and thus, equi-distributed. In the second case, we experimented with a clustered structure. Agents had to search boxes located in a small area. What we expect is that the probability of conflicts between agents is much higher in the second type of scenario setting.

Measure of Performance: In the experiment, the following variables were measured in order to judge the performance of the system:

- The CPU time needed by the forklift agents in order to perform the task.
- The number of primitive actions $\{walk\_ahead, turn\_left, turn\_right, get\_box, put\_box\}$ which were carried out by the agents in order to reach their goals.
- The number of conflicts among the agents. In our experiment, an agent notices a conflict with another agent if that one stands on the field in front of it. Since no

[1] Or until the next system crash appears - whatever comes earlier!

communication was used in the experiments, the conflict rate was computed from an agent-based point of view.

For each variable, we recorded the minimum, maximum, and average value per agent as well as the sum over all the agents.

12.2.2 Results

Table 12.1 displays the run-time results for three agents. The columns show the results for the different configuration types presented above. T_1 means type 1 agents, T_2 type 2 agents, and so on. For agents of type 1 and type 2, we experimented with clustered (denoted by *cl.*) and equidistributed (denoted by *eq.* in the table) task characteristics. For type 3, 4, and 5 agents, we only show the results in the case of equidistribution.

$n = 3$	T_1		T_2		T_3	T_4	T_5
tasks	*eq.*	*cl.*	*eq.*	*cl.*	*eq.*	*eq.*	*eq.*
CPU time							
min	30.5	54.6	38.4	99.8	38.8	60.7	300.63*
max	67.9	148.8	192.6	230.1	192.2	515.8	-
sum	161.9	282.2	378.9	524.5	373.9	942.1	-
avg	54.0	94.1	126.3	174.8	124.6	314.1	
#Steps							
min	35	53	38	92	30	40	245*
max	71	165	277	214	167	275	-
sum	174	296	470	519	308	557	-
avg	58	98.7	156.7	173	102.7	185.7	-
# Conflicts							
min	0	1	0	3	0	0	0*
max	3	4	3	6	0	0	-
sum	5	9	5	14	0	0	-
avg	1.7	3	1.7	2.8	0	0	-

Table 12.1: Experiment for Three Forklift Agents

The values for the random walker (T_5) are furnished with an asterisk (*). They mean the solution achieved by the *first* agent who was able to achieve its goal using the random walk strategy. Of course, the exact numbers are only interesting in comparison with the performance of the other agent configurations.

Figure 12.2 shows the results of the experiments carried out for six agents. For this test set, we have compared the behaviour of agents of type 1 and type 2, using a both clustered and equi-distributed test sets.

In table 12.3, the results of the experiment for nine forklift agents can be found. As in the case of six agents, we restricted ourselves to agent types 1 and 2.

12.2.3 Discussion

There are some intereting remarks which can be made as regards the tests described above.

- Obviously, type 1 agents perform best of all, whereas the random walk agents have a very poor performance. This shows what we expected, namely that randomness is not a good default strategy - provided that the agent has some knowledge available.

$n = 3$	T_1		T_2	
tasks	*eq.*	*cl.*	*eq.*	*cl.*
CPU time				
min	26.2	78.2	25.3	74.5
max	201.4	207.6	349.56	225.5
sum	494.7	808	887.4	740.0
avg	82.5	134.7	147.9	123.3
#Steps				
min	27	75	31	65
max	204	201	327	200
sum	508	763	806	679
avg	84.7	127.2	134.3	113.2
# Conflicts				
min	1	2	1	2
max	4	8	5	5 -
sum	13	29	17	22
avg	2.2	4.8	2.8	3.7

Table 12.2: Experiment for Six Forklift Agents

$n = 3$	T_1		T_2	
tasks	*eq.*	*cl.*	*eq.*	*cl.*
CPU time				
min	32.0	48.7	28.1	66.6
max	155.6	321.1	408.4	627.0
sum	872.1	1289.7	1631.7	2451.8
avg	96.6	143.3	181.3	272.4
#Steps				
min	38	54	32	53
max	125	324	405	563
sum	773	1249	1609	2132
avg	85.9	127.2	178.8	236.9
# Conflicts				
min	1	1	1	3
max	7	18	9	25
sum	29	74	37	87 -
avg	3.2	8.2	4.1	9.7

Table 12.3: Experiment for Nine Forklift Agents

- Type four agents take a longer time in order to achieve their goal than type three agents. The property of curiosity, which is very important if the agent has to orient itself in an unknown area, does not pay off in the experiment, since it requires goal-directed behaviour.

- In general, agents behave worse if tasks are clustered than if tasks are equidistributed.

- The intelligent strategies yield relatively bad results in the clustered case compared to their behaviour in the non-clustered case. This is because agents find their way to the destination quite quickly, and thus, all of them will arrive in the shelf area at about the same time.
- The number of conflicts per agent seems to grow approximately linear with the number of agents in the non-clustered case. In the clustered case, the number increases much faster. This is shown graphically in figure 12.1 for the case of type 1 and type 2 agents.
- The agents using randomness and weighted randomness are less sensitive to conflicts than the plan-based agents.

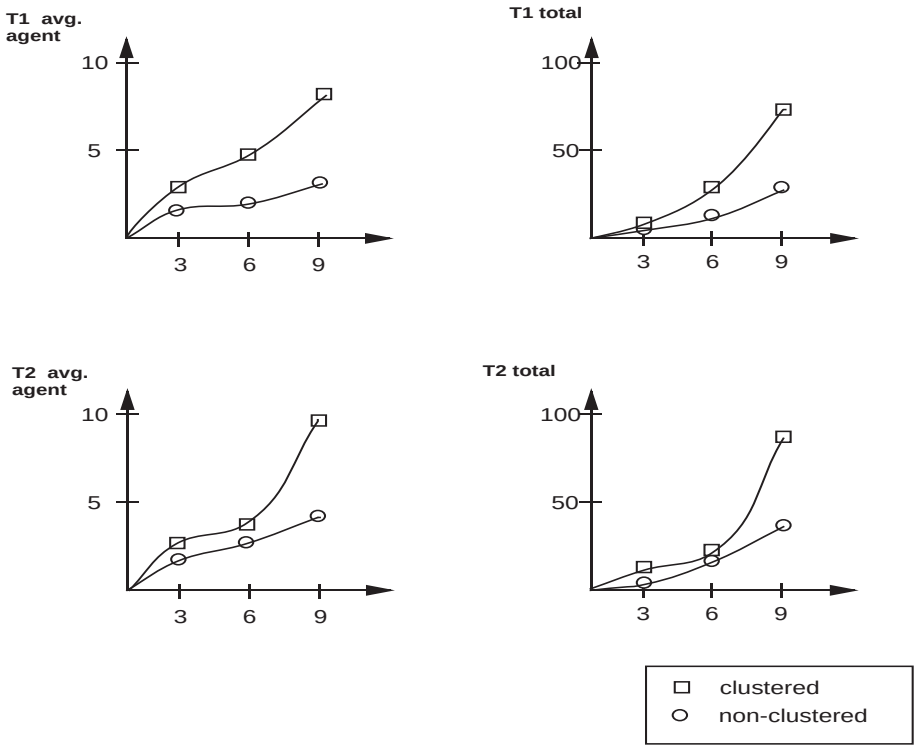

Figure 12.1: Conflict Rates for Different Numbers of Agents

In conclusion, this first test series has shown the results of different agent configurations in the loading dock. The relationship between the number of agents in the scenario and the number of conflicts has been investigated. However, the domain offers a wid variety of further possible experiments. Firstly, the combination of behaviour-based and plan-based methods is an ideal area for empirical investigation. Secondly, up to now, we have only regarded homogeneous agent scoieties, i.e. all the agents started with the same configuration. It would be interesting to see how different types of agents behave in direct competition, and how the system as a whole is affected by varying the composition of the agent society. For example, it could be interesting to see whether we could confirm the empirical results of Kephart et al. [KHH89] that it is sometimes better to have not only smart agents[2]. These kinds of experiments will be the subject of our future work.

[2]In our current experiment, we already noticed a phenomenon which can be interpreted this way: if we assume a clustered task characteristics, in societies consisting of smart agents (types 1 and 2) conflict probability tends to be higher than in societies with less "intelligent" agents; this is due to the more goal-directed behaviour of the former ones.

Chapter 13

Conclusion and Outlook

In this report, we have presented the INTERRAP agent model and have evaluated it by means of a robotics application, the loading dock domain. By our work, we have made the following contribution: INTERRAP makes it possible to flexibly combine features of behaviour-based and plan-based agents. Thus, the designer of a MAS can choose among a large spectrum of mechanisms available for action and interaction, depending on the requirements imposed by the application domain. Especially, we offer *patterns of behaviour* for modeling reactive agents and for modeling procedural "routine tasks", as well as a planning mechanism for single- and multi-agent planning. By defining a flexible interplay between a behaviour-based and a plan-based component, our agents are robust and can cope with many situations.

We presented a two-dimensional goal hierarchy as a data structure which allows an agent to decide what goals to pursue next. But, what is more, the goal hierarchy can be regarded as a first step to an agent development tool for the designer of a multi-agent system. Our long-term vision is that the goal hierarchy will represent a generic agent model, which can be instantiated by a designer by defining possible goals, their priorities, and their implementation.

Of course much work remains to be done: Firstly, up to now, there is no general theory of the relationship between the situational context, the mental context, and the possible goals of an agent (i.e. the preselection module in figure 3). Whereas the situational context can be implemented in a very elegant manner using the forward-reasoning facilities of OPS-5, it is not yet clear how the current goals of an agent interfere with new goals.

Secondly, in this paper we have said much about the decision as to which goal to pursue, but only little about the decision as to which mechanism should be chosen for the execution. We are implementing with several (both behaviour-based and plan-based) mechanisms of action and interaction in the implementation of the loading dock; first results will be obtainable, soon (see [Mül93]).

Finally, the task of finding a good compromise between solving problems by using behaviour-based and using plan-based mechanisms is a challenge. First experiments confirm that, the more plan-based our agents are, the more we are faced with standard AI problems such as the frame problem. On the other hand, the more reactive (in the sense of "behaviour-based") they are, the less intelligent they behave in (possibly highly constrained) "standard" situations requiring more or less fixed sequences of actions, as well as in complex interactions with other agents. Here, the concept of INTERRAP pays off since it allows us to experiment with various mechanisms for solving problems at different levels in the problem hierarchy (see figure 11.1). A great deal of our future work will consist in extensively experimenting with different configurations of the scenario. Our hope is to discover more general rules which describe the scope and the limitations of behaviour-based and plan-based approaches.

The work presented in this paper has been supported by the German Ministry of Research and Technology under grant ITW9104.

Bibliography

[AH85] G. Agha and C. E. Hewitt. Concurrent Programming Using ACTORS: Exploiting Large Scale Parallelism. Technical Report AI Memo 865, MIT, 1985.

[AHT90] J. F. Allen, J. Hendler, and A. Tate. *Readings in Planning.* Morgan Kaufmann, San Mateo, 1990.

[AKPT91] J. F. Allen, H. A. Kautz, R. N. Pelavin, and J. D. Tenenberg. *Reasoning About Plans.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[ALS89] N. M. Avouris, M. H. V. Liedekerke, and L. Sommaruga. Evaluating the CooperA Experiment: the Transition from an Expert System Module to a Distributed Artificial Intelligence Testbed for Cooperating Expert Systems. In *Proc. of the 9th Workshop on Distributed AI*, 1989.

[Axe84] R. Axelrod. *The Evolution of Cooperation.* Basic Books, 1984.

[BAF+87] R. Bisiani, F. Alleva, A. Forin, R. Lerner, and M. Bauer. The Architecture of the AGORA Environment. In M. N. Huhns, editor, *Distributed Artificial Intelligence*, pages 99–118. Pitman / Morgan Kaufmann, San Mateo, CA, 1987.

[BBH+90] F. Baader, H.-J. Bürckert, B. Hollunder, W. Nutt, and J. H. Siekmann. Concept logics. In Lloyd, editor, *Proceedings of Symposium on Computational Logic*, ESPRIT Basic Research Series, pages 177–201. Springer, 1990.

[BFS91] T. Bouron, J. Ferber, and F. Samuel. MAGES: A Multi-Agent Testbed for Heterogeneous Agents. In *Decentralized A.I. 2.* North-Holland, 1991.

[BG88] A. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, Los Angeles, CA, 1988.

[BH90] F. Baader and B. Hollunder. Kris: Knowledge representation and inference system; system description. Technical Report Technical Memo TM-90-03, DFKI, Kaiserslautern, 1990.

[BH92] F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning*, Cambridge, Mass., 1992.

[BIP88] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, November 1988.

[BKL92] A. Bernardi, C. Klauck, and R. Legleitner. PIM: Planning in Manufacturing using Skeletal Plans and Features. Technical Report RR-92-19, German Research Centre for Artificial Intelligence, 1992.

[BM91] H. J. Bürckert and J. Müller. RATMAN: Rational Agents Testbed for Multi Agent Networks. In *Decentralized A.I. 2*. North Holland, 1991.

[BM92] S. Bussmann and H. J. Müller. A Negotiation Framework for Cooperating Agents. In *Proc. of the 2nd Workshop on Cooperating Knowledge Based Systems*. Keele, GB, September 1992.

[Bro86] Rodney A. Brooks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, volume RA-2 (1), April 1986.

[Bro91] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[BS92] B. Burmeister and K. Sundermeyer. Cooperative problem-solving guided by intentions and perception. In E. Werner and Y. Demazeau, editors, *Decentralized A.I. 3*. North-Holland, 1992.

[CGHH89] P. R. Cohen, M. L. Greenberg, D. M. Hart, and A. E. Howe. Trial by fire: Understanding the design requirements for agents in comlex environments. *AI Magazine*, 10(3), 1989.

[Chu74] K. L. Chung. *Elementary Probability Theory with Stochastic Processes*. Springer, New York, 1974.

[CL87] D. D. Corkill and V. R. Lesser. Distributed problem solving. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 245–251. John Wiley and Sons, New York, 1987.

[CL88] Daniel D. Corkill and Victor R. Lesser. The distributed vehicle monitoring testbed: A tool for investigating distributed problem solving networks. In Robert Engelmore and Tony Morgan, editors, *Blackboard Systems*. Addison-Wesley Publishing Company, 1988.

[CL90] P. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42, 1990.

[CML88] Susan E. Conry, Robert A. Meyer, and Victor R. Lesser. Multistage negotiation in distributed planning. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 367–384. Morgan Kaufmann Publishers, 1988.

[DCJL91] J. Doran, H. Carvajal, Y. J.Choo, and Y. Li. The MCS Multi-Agent Testbed: Developments and Experiments. In *CKBS-90 - Proc. of the International Working Conference on Cooperating Knowledge-Based Systems*. Springer-Verlag, 1991.

[DL86] E. H. Durfee and V. R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proc. of the Fifth National Conference on Artificial Intelligence*, pages 58–64. Philadelphia, Pennsylvania, USA, 1986.

[DL89] E. H. Durfee and V. R. Lesser. Negotiating task decomposition and allocation using partial global planning. In *Distributed Artificial Intelligence, Volume II*, pages 229–244, San Mateo, CA, 1989. Morgan Kaufmann Publishers, Inc.

[DM90] E. H. Durfee and T. A. Montgomery. A hierarchical protocol for coordination of multiagent behaviour. In *Proc. of the 8th National Conference on Artificial Intelligence*, pages 86–93. Boston, MA, 1990.

[DM91] E. H. Durfee and T. A. Montgomery. Coordination as distributed search in a hierarchical behaviour space. *IEEE Transactions on Systems, Man, and Cybernatics, Special Issue on DAI*, 21(6), December 1991.

[DS83] R. Davis and R.G. Smith. Negotiation as a metaphor for distributed problem solving. In *Artificial Intelligence, 20(1)*, pages 63–109, 1983.

[EHRLD80] L.D. Ermann, F. Hayes-Roth, V.R. Lesser, and D.R.Reddy. The HEARSAY-II speech understanding system: Integrating Knowledge to resolve uncertainty. In *Computing Surveys 12(2)*, pages 213–253, 1980.

[ER92] E. Ephrati and J. Rosenschein. Reaching agreement through partial revelation of preferences. In *Proc. of ECAI-92*, pages 229–233. Vienna, August 1992.

[FD90] T. Fraichard and Y. Demazeau. Motion planning in a multi-agent world. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I.*, pages 137–153. North-Holland, 1990.

[Fer89] J. Ferber. Eco-problem solving: How to solve a problem by interactions. In *Proc. of the 9th Workshop on DAI*, pages 113–128, 1989.

[Fer92] I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, Computer Laboratory, University of Cambridge, UK,, 1992.

[FHN71] R. E. Fikes, P. E. Hart, and N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving. *Artificial Intelligence*, 2:189–208, 1971.

[Fin87] J. Finger. *Exploiting Constraints in Design Synthesis*. PhD thesis, Stanford University, 1987.

[FKM+93] K. Fischer, N. Kuhn, H.-J. Müller, J. P. Müller, and M. Pischel. Sophisticated and distributed: The transportation domain. In *Proc. of MAAMAW-93*, Neuchatel, CH, August 1993. Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World.

[FL77] Richard D. Fennell and Victor R. Lesser. Parallelism in AI Problem Solving: A Case Study of HEARSAY-II. In *IEEE Transactions on Computers*, pages 98–111, February 1977.

[For82] C.L. Forgy. Rete — a fast algorithm for the many pattern – many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[FW92] K. Fischer and H. M. Windisch. MAGSY- Ein regelbasiertes Multi-Agentensystem. In H. J. Müller, editor, *KI1/92, Themenheft Verteilte KI*. FBO-Verlag, 1992.

[GBH87] L. Gasser, C. Braganza, and N. Hermann. MACE: A Flexible Testbed for Distributed AI Research. In M. N. Huhns, editor, *Distributed Artificial Intelligence*. Pitman / Morgan Kaufmann Publisher, 1987.

[GI89] M. P. Georgeff and F. F. Ingrand. Decision-making in embedded reasoning systems. In *Proc. of the 6th International Joint Conference on Artificial Intelligence*, pages 972–978, 1989.

[GL86] M. P. Georgeff and A. L. Lansky. Procedural knowledge. In *Proc. of the IEEE Special Issue on Knowledge Representation*, volume 74, pages 1383–1398, 1986.

[GL87] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proc. of the 6th National Conference on Artificial Intelligence*, 1987.

[Gor77] T. Gordon. *Leader Effectiveness Training L. E. T.* Peter H. Wyden, New York, 1977.

[Had93] A. Haddadi. Eine Hybride Architektur für Mehragentensysteme. In H. J. Müller, editor, *Beiträge zum Gründungsworkshop der Fachgruppe VKI*, pages 1–10. DFKI, Saarbrücken, 1993.

[Hew73] C. Hewitt. A Universal Modular ACTOR Formalism for AI. In *Proc. of IJCAI-73*, pages 235–245. Morgan Kaufmann, 1973.

[Hew77] C. Hewitt. Viewing control structures as patterns of message passing. *Artificial Intelligence*, 8(3):323–364, 1977.

[Hew85] C. E. Hewitt. The challenge of open systems. *Byte*, 4(10), 1985.

[HI91] C. Hewitt and J. Inman. DAI betwixt and between: From "intelligent agents" to open systems. *IEEE Transactions on Systems, Man, and Cybernetics (Special Section on DAI)*, 21(6):1409–1419, November/December 1991.

[HM90] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[HM92] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[HR89] B. Hayes-Roth. Making intelligent systems adaptive. Technical Report STAN-CS-88-1226, Stanford University, Stanford, CA, October 1989.

[Hub88] B. Huberman. *The Ecology of Computation.* Elsevier Sciences Publication, North-Holland, 1988.

[Jen92] N. R. Jennings. *Joint Intentions as a Model of Multi-Agent Cooperation.* PhD thesis, Queen Mary and Westfield College, London, August 1992.

[Kae90] L. P. Kaelbling. An architecture for intelligent reactive systems. In J. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 713–728. Morgan Kaufmann, 1990.

[Kau91] H. A. Kautz. *A Formal Theory of Plan Recognition and its Implementation*, pages 69–126. Morgan Kaufmann Publishers, Inc., San Mateo, Calif., 1991.

[KHH89] J. O. Kephart, T. Hogg, and B. A. Huberman. Dynamics of Computational Ecosystems: Implications for DAI. In L. Gasser and H. M. Huhns, editors, *Distributed Artificial Intelligence, Volume II.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1989.

[KJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598), 1983.

[Kle90] M. Klein. A computational model of conflict resolution in cooperative design systems. In *Proc. of the 10th International Workshop on DAI*, Austin, Texas, 1990. MCC.

[KMM93a] N. Kuhn, H. J. Müller, and J. P. Müller. Simulating cooperative transportation companies. In *Proceedings of the European Simulation Multiconference (ESM-93)*, Lyon, France, June 1993. Society for Computer Simulation.

[KMM93b] N. Kuhn, H. J. Müller, and J. P. Müller. Task decomposition in dynamic agent societies. In *Proceedings of the International Symposium on Autonomous Decentralized Systems (ISADS-93)*, Tokyo, Japan, 1993. IEEE Computer Society Press.

[KR76] R. L. Keeney and H. Raiffa. *Decisions With Multiple Objectives.* John Wiley and Sons, New York, 1976.

[KT93] R. Kakehi and M. Tokoro. A negotiation protocol for conflict resolution in multi-agent environments. In *Proceedings of ICICIS-93*, pages 185–196. Rotterdam, 1993.

[KvM91] T. Kreifelts and F. v. Martial. A negotiation framework for autonomous agents. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I. 2.* North-Holland, 1991.

[Lat92] J. P. Latombe. How to move (physically speaking) in a multi-agent world. In Y. Demazeau and E. Werner, editors, *Decentralized A.I. 3.* North-Holland, 1992.

[Lau93] A. Laux. Representing belief in multi-agent worlds via terminological logics. In H.-J. Bürckert, editor, *Beiträge zum Workshop* Modellierung Epistemischer Propositionen*, Fachtagung Künstliche Intelligenz*, Berlin, September 1993. unpublished.

[LBS92] A. Lux, F. Bomarius, and D. Steiner. A model for supporting human computer cooperation. In *AAAI Workshop on Cooperation among Heterogeneous Intelligent Systems*, July 1992.

[Len75] D. B. Lenat. Beings: Knowledge as interacting experts. In *Proc. of IJCAI-75.* Morgan Kaufmann, 1975.

[Les91] V. R. Lesser. A Retrospective View of FA/C Distributed Problem Solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1347–1362, 1991.

[Lew35] K. Lewin. *A dynamic theory of personality.* New York, 1935.

[LH92] D. M. Lyons and A. J. Hendricks. Planning, reactive. In *Encyclopedia of Artificial Intelligence*, pages 1171–1181. John Wiley, 2nd edition, 1992.

[LR57] R. D. Luce and H. Raiffa. *Games and Decisions: Introduction and Critical Survey.* John Wiley and Sons, 1957.

[Mal87] T. W. Malone. Modelling coordination in organizations and markets. *Management Science*, 33:1317–1332, 1987.

[McD90] D. McDermott. Planning reactive behaviour: A progress report. In J. Allen, J. Hendler, and A. Tate, editors, *Innovative Approaches to Planning, Scheduling, and Control*, pages 450–458. Morgan Kaufmann, San Mateo, CA, 1990.

[MH90] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In J. Allen, J. Hendler, and A. Tate, editors, *Readings inPlanningg*, pages 393–435. Morgan Kaufmann, San Mateo, CA, 1990.

[MP93] J. P. Müller and M. Pischel. Modelling Robot Societies using InteRRaP. In *Working Notes of the IJCAI Workshop on Dynamically Interacting Robots*, Chambery, France, August 1993. Forthcoming.

[Mül93] J. P. Müller. Towards a model for flexible agent interaction. Technical Report RR-93-01, German Research Centre for Artificial Intelligence, Saarbrücken, (Forthcoming), 1993.

[Oga91] G. H. Ogasawara. A distributed, decision-theoretic control system for a mobile robot. *ACM SIGART bulletin*, 2(4):140–145, 1991.

[OY82] C. Ó'Dúnlaing and C. K. Yap. A retraction method for planning the motion of a disc. *J. of Algorithms*, 6, 1982.

[Par93] H. Van Dyke Parunak. Industrial applications of multi-agent systems. In *Proc. of INFAUTOM-93*, Toulouse, February 1993. Association Colloque SUP'AERO.

[PR90] M. E. Pollack and M. Ringuette. Introducing the tile-world: Experimentally evaluating agent architectures. In *Proc. of the Conference of the American Association for Artificial Intelligence*, pages 183–189, 1990.

[RG85] Jeffrey S. Rosenschein and Michael R. Genesereth. Deals among rational agents. In *IJCAI-85*, pages 91–99, 1985.

[RG91] A. S. Rao and M. P. Georgeff. Modeling Agents Within a BDI-Architecture. In R. FIkes and E. Sandewall, editors, *Proc. of the 2rd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, Cambridge, Mass., April 1991. Morgan Kaufmann.

[RK91] E. Rich and K. Knight. *Artificial Intelligence*. McGraw Hill, 2nd edition, 1991.

[Ros85] Jeffrey S. Rosenschein. *Rational Interaction: Cooperation among Intelligent Agents*. PhD thesis, Stanford University, 1985.

[Sac75] Earl D. Sacerdoti. The nonlinear nature of plans. In *IJCAI-75*, pages 206–218, 1975.

[Sch92] A. Schupeta. COSMA: Ein verteilter Terminplaner als Fallstudie der Verteilten KI. In H. J. Müller and D. Steiner, editors, *Workshop Kooperierende Agenten*, pages 50–55. DFKI Document D-92-24, 1992.

[Sea69] J. R. Searle. *Speech Acts*. Cambridge University Press, 1969.

[SF89] A. Sathi and M. S. Fox. Constraint-directed negotiation of resource reallocations. In Less Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, pages 163–193. Morgan Kaufmann, San Mateo, California, 1989.

[SH90] J. Sanborn and J. Hendler. A model of reaction for planning in dynamic environments. *International Journal of Artificial Intelligence in Engineering*, 6(1):41–60, 1990.

[Sho93] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[SMH90] D. Steiner, D. Mahling, and H. Haugeneder. Human computer-supported cooperative work. In *Proc. of the 10th International Workshop on DIstributed Artificial Intelligence*. MCC Technical Report Nr. ACT-AI-355-90, 1990.

[SRSF90] K. P. Sycara, S. Roth, N. Sadeh, and M. Fox. An investigation into distributed constraint-directed factory scheduling. In *Proc. of CAIA-90*, 1990.

[SS93] P. Sablayrolles and A. Schupeta. Conflict Resolving Negotiation for COoperative Schedule Management Agents (COSMA). Technical Report Technical Memo TM-93-02, DFKI Saarbücken, May 1993.

[ST92] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agent societies. In *Proc. of AAAI-92*, pages 276–281, 1992.

[Ste90] L. Steels. Cooperation between distributed agents through self-organization. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I.*, pages 175–196. North-Holland, 1990.

[Suc87] L. A. Suchman. *Plans and Situated Actions.* Cambridge Universtiy Press, Cambridge, 1987.

[Syc87] K. P. Sycara. *Resolving Adversarial Conflicts: An approach integrating case-based and analytic methods.* PhD thesis, Georgia Institute of Technology, Atlanta, Georgia, June 1987.

[Syc88] K. P. Sycara. Resolving goal conflicts via negotiation. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 245–250, St. Paul, Minnesota, August 1988.

[Syc89] K. P. Sycara. Multiagent compromise via negotiation. In L. Gasser and M. N. Huhns, editors, *Distributed Artificial Intelligence, Volume II*, pages 119–137. Morgan Kaufmann, San Mateo, California, 1989.

[vM90] F. von Martial. Interactions among autonomous planning systems. In Y. Demazeaau and J.-P. Müller, editors, *Decentralized A. I.*, pages 105–120. North-Holland, 1990.

[vP93] E. von Puttkamer. Kaiserslautern university, mobile robotics group. personal communication., 1993.

[Wal91] P. Wallich. Silicon babies. *Scientific American*, December 1991.

[WD92] E. Werner and Y. Demazeau, editors. *Decentralized A.I. 3.* North-Holland, 1992.

[WHS93] J. Würtz, M. Henz, and G. Smolka. Oz - a programming language for multi-agent systems. In *Proc. of the International Joint Conference on AI.* IJCAI-93, 1993.

[Wil88] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, CA, 1988.

[Win75] T. Winograd. Frame representations and the declarative / procedural controversy. In D.G. Bobrow and A. Collins, editors, *Representation and Understanding - Studies in Cognitive Science*, pages 185–210. Academic Press, 1975.

[Win84] P. H. Winston. *Artificial Intelligence.* Addison-Wesley, 2nd edition, 1984.

[Wit92] T. Wittig. *ARCHON: An Architecture for Multi-Agent Systems*. Ellis Horwood, 1992.

[Woo92] M. Wooldridge. *On the Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, UMIST, Department of Computation, Manchseter, UK, 1992.

[WS92] W. A. Woods and J. G. Schmolze. The KL-ONE Family. In F. W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992.

[ZR89] G. Zlotkin and J. S. Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domains. In *Proc. of the Eleventh IJCAI*, pages 912–917. Detroit, Michigan, August 1989.

[ZR91] G. Zlotkin and J. S. Rosenschein. Negotiation and goal relaxation. In Y. Demazeau and J.-P. Müller, editors, *Decentralized A.I.2*, pages 273–286. North-Holland, 1991.

[ZR92] G. Zlotkin and J. S. Rosenschein. A domain theory for task oriented negotiation. In A. Cesta, R. Conte, and M. Micheli, editors, *Pre-Proceedings of MAAMAW-92*, July 1992.