Integrating Agent Interaction into a Planner-Reactor Architecture*

Jörg P. Müller, Markus Pischel†
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken

Abstract

This paper deals with the development of methodologies and architectures for autonomous systems that act and interact in a shared, dynamic environment. We present the agent architecture INTERRAP[1]. INTERRAP extends previous approaches to building layered architectures by a cooperation layer, and thus allows the designer of a system not only to build agents with both reactive and deliberative facilities, but also to provide important mechanisms for communication, coordination, and collaboration with other agents. The components of the architecture and their interplay are described. It is evaluated by the simulation of an automated loading dock, an interacting robots application, where autonomous forklifts have to perform loading and unloading tasks. We show how different functionalities of the forklift agents can be modeled in an adequate manner at different layers of the INTERRAP architecture. Thus, the main contribution of this work is the reconciliation of (D)AI concepts such as goals, plans, conflict resolution, and negotiation with mobile robot research which is absolutely necessary in order to model the lower layers of the agent, such as sensor data processing, obstacle avoidance, navigation, and physical control.

1 Introduction

Our research deals with principles and architectures for the design of autonomous agents which act and interact in a shared, heterogeneous, and dynamic environment. Surviving in such an environment while performing complex tasks, which often exceed the problem-solving capacities of an individual agent, puts strong requirements on agents, such as *situated*, *goal-directed*, *adaptive*, and *efficient* behaviour, as well as facilities for *interaction* and *coordination*. The latter are of special importance in dynamic multi-agent environments. Whereas certain types of *interactions* can often be performed by employing local mechanisms, others require coordinated planning (see

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

†e-mail:{jpm,pischel}@dfki.uni-sb.de

[1]**Int**egration of **R**eactive Behaviour and **Ra**tional **P**lanning

e.g. [KLR+92]), and thus the explicit representation of other agents' beliefs, goals, and plans. Trying to cover these requirements by using a layered architecture with modules working in parallel is a very natural matter. Indeed, looking at currently proposed agent architectures reveals that layered architectures are - rightfully - *en vogue* (see e.g. [Bro86, GL86, Fir92, Fer92, LH92]). However, virtually all approaches focus on the integration of a reactive, procedural component with a deliberative one. Whereas this seems adequate to achieve the requirements of situated, goal-directed, and efficient agents, it does not address issues of interaction and cooperation.

The INTERRAP agent model [MP93], which has been developed at the DFKI, is an approach aimed to fill these gap. By providing a layered architecture, it allows the designer of a system to specify the reactive, procedural, local planning, and interactive capabilities of the agents. The layers are linked by a flexible control mechanism: as tasks get more sophisticated, their control is shifted from lower layers to higher layers (*competence driven*). At the same time, the lower layers remain active, and can thus cope with new events. On the other hand, for task execution, control flows top-down (*activation driven*) from higher layers to lower layers.

According to the characteristics of the application domain, the designer of a system can configure appropriate agent types by mapping certain agent functionalities to the different layers of the architectures. Thus, the scope of the model ranges from describing very simple, purely reactive agents to very sophisticated agents equipped with (cooperative) planning facilities. The objective of our work is to define a shell for the development of autonomous, interacting systems.

The application that has been used to first evaluate our model is the FORKS system, a simulation of an automated loading-dock, where autonomous forklift agents have to load and to unload trucks[2]. Figure 1 gives an idea of the structure of the scenario. In the loading dock, there are shelves which may contain different types of goods. We use a grid-based representation of the loading-dock. The primitive actions the forklifts can do are moving from one square to the next, turning around, grasping and storing goods, and communicating with other agents. Each agent has a certain range of perception, which it can observe. This representation of the world and the agents' actions simplify many of the problems of mechanical control and geometrical trajectory planning while preserving its physical dynamics and its interactive character: forklifts move around autonomously in the dock and have to synchronize and coordinate their actions, they have to avoid collisions, and to resolve blocking situations. Apart from merely local mechanisms for doing this, they should cooperate in order to resolve conflicts (e.g. blockings or threatening collisions), or even to perform transportation tasks.

The paper is structured as follows: In section 2, we give an overall description of the INTERRAP model. In section 3, we provide a description of how reaction and deliberation are achieved by the interplay of the behaviour-based and plan-based component. Section 4 describes the integration of interaction into the INTERRAP model. In section 5, related work in the field is discussed. We finish with a statement of the contribution of this paper and with an outlook of future work in section 6.

[2]See [FKM94] for a further application domain.

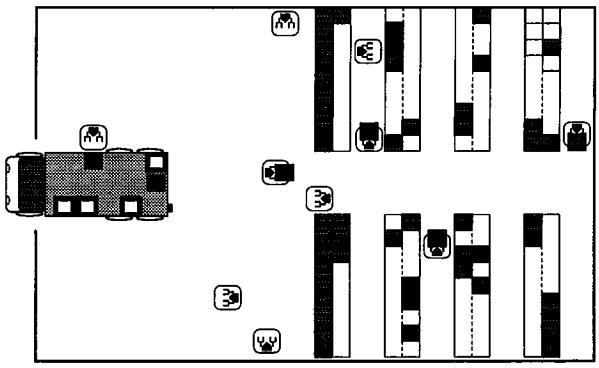

Figure 1: The Loading-Dock

2 The INTERRAP Agent Model

In this section, we explain the key ideas of the INTERRAP agent model and its basic functional structure. The main idea of INTERRAP is to define an agent by a set of functional layers, linked by a activation-based control structure and a shared hierarchical knowledge base. Figure 2 overviews the INTERRAP agent model. It consists

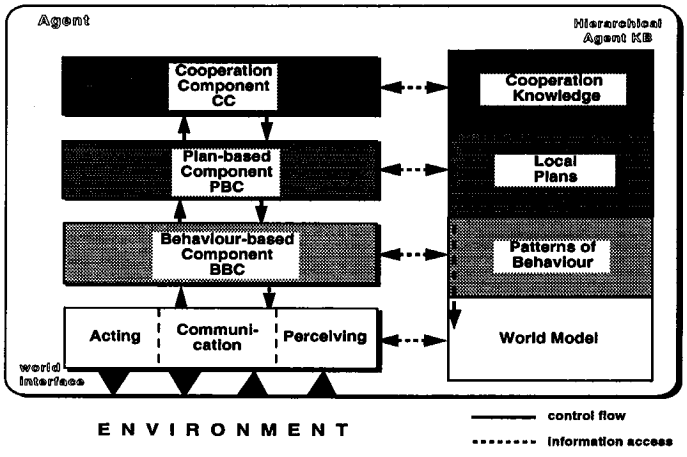

Figure 2: The INTERRAP Agent Model

of five basic parts: the world interface (WIF), the behaviour-based component (BBC), the plan-based component (PBC), the cooperation component (CC), and the agent knowledge-base. The *WIF* contains the agent's facilities for perception, action, and communication. The *BBC* implements and controls the basic reactive behaviour of the agent as well as its procedural knowledge (abstract actions). It is based on the concept of *patterns of behaviour*. These allow an agent to react flexibly to its environment, and to perform routine tasks without using an explicit symbolic representation of how the task is to be performed. 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. Planning helps 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 ([KLR+92]) to cope with special situations. This functionality is provided by the cooperation component CC.

The *knowledge base* is structured in a hierarchical manner. It consists of four layers which basically correspond to the structure of the agent control, representing the agent's world model, the patterns of behaviour, local plans / goals, and knowledge of / strategies for cooperation, respectively. The basic idea is that information is passed only from lower layers of the knowledge base to higher layers. 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.

In the following section, we introduce the loading-dock application. In the following sections, the modules of INTERRAP and their interplay are explained in more detail.

3 Integrating Reaction and Deliberation

In this section, we describe the interplay between planning and execution in the INTERRAP model. For this purpose, the world interface, behaviour-based and plan-based layers are examined in detail. Wherever necessary, the description will be enhanced by examples taken from the loading-dock application.

3.1 The World Interface

In the *world interface*, the agent's facilities for perception, action, and communication are modeled. 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 our simulation system, perception is implemented as follows: each agent has a configurable range of perception ($n \times m$ squares). Since a change within this range of perception can be always associated with a certain square, each time something within the perceived fields of an agent changes, the simulation world sends the agent a description of what has changed within this square - at a symbolic level. In a real robot environment, this knowledge has to be obtained by a complex hierarchical process of transformation of the sensor data obtained by e.g. video camera, infra-red, laser, or ultrasonic sensors. The *actions* component controls the effectoric capabilities of the agent. Obviously, these capabilities are very domain-dependent. In the case of our forklift robots, these actions are *walk_ahead*, *turn_left*, *turn_right*, *put_box*, *grasp_box*.The *communication* unit bears the agent's communicative facilities. It controls the physical realization of sending and receiving messages. Since we allow heterogeneous agents, outgoing messages have to be transformed from the agent's internal language into a format understood by all agents. Analogously, incoming messages must be transformed into the local agent language. This transformation is done by a *translator module.*

3.2 The Behaviour-based Component

The *BBC* implements the basic reactive behaviour and the procedural knowledge of the agent. It consists of a control component *bbc-control* and of a set of patterns of behaviour together with a priority mechanism which are represented and maintained in the *pattern maintenance unit.* The *bbc-control* serves as an interface to the neighbour modules and as an interpreter for the execution of the patterns of behaviour. The pattern maintenance unit describes a mechanism for determining the situated priority of the patterns (see [MP93] for a detailed description of pattern priority). A pattern agenda is maintained and recomputed in each BBC cycle; the pattern with highest priority in this agenda is chosen for execution by the *bbc-control.* In the following, we will explain our concept of patterns of behaviour in more detail.

3.2.1 Patterns of Behaviour (PoB)

According to their activation/effect functionality, we distinguish between four basic types of patterns: *reactors*, *control modifiers*, *knowledge modifiers*, and *procedures.* Table 1 overviews this classification. *Reactor patterns* are triggered by external

Effect / Activation	modify world	modify knowledge	shift control to PBC
external (world)	reactor	knowledge modifier	control modifier
internal (PBC)	procedure	?	?

Table 1: Classification of Patterns of Behaviour

events and cause the agent to perform some sort of action. For example, stopping when facing an obstacle in front of it should be implemented as a reactor pattern. *Knowledge modifiers* are patterns that change the internal state of the agent (e.g. its knowledge). They are activated by changes in the world perceived by the agent (i.e. by changes in the agent's world model). In our approach, they are used to implement the recognition and classification of situations. Similar to knowledge sources in a blackboard system, there are patterns that recognize and abstract specific situations (e.g. *another agent ahead, standing in a narrow corridor*). Other patterns recognize more complex pattern based on the results of the lower-level knowledge modifiers. *Control modifiers* are patterns that expand control to the planner by calling the PBC. For example, a pattern `treat_order_beh` will activate the PBC with the goal of planning an order as soon as the agent has received a transportation order. Finally, *procedure* patterns implement what is viewed as abstract actions by the planner[3]. For example, moving straight ahead to a landmark is likely to be implemented as a procedure in a robot application, i.e. is atomic from the perspective of the PBC. Since we assume that the planner basically plans *actions*, our classification does not take into account patterns that are triggered internally and yield only a modification of the agent's world model or an activation of the planner.

[3]A different view of procedures is that of *routine tasks* whose performance does not require explicit planning.

Based on this classification, patterns of behaviour are abstractly defined with the following attributes:

```
( PoB
      :name              /* Name of pattern */
      :type              /* reactor, modifier, procedure */
      :args              /* arguments */
      :activation        /* activation condition */
      :monitor           /* conditions monitoring execution */
           :failure      /* failure condition:  stop execution */
           :success      /* condition for successful termination */
           :during       /* conditions that must hold during execution */
           :post         /* condition that must hold after execution */
           :exceptions   /* user-definable exceptions */
      :exec_body         /* actual executable body; e.g. control program */ )
```

An exception is a tuple *(Cond, Act)*, where `Cond` is a *state formula*, and `Act` is an action description. The operational semantics is that if `Cond` becomes satisfied by the current state of the world, `Act` is executed. Activation, failure, success, and post conditions can be viewed as predefined exceptions with a fixed semantics of their action part. Apart from these, user-defined exceptions can be specified.

Thus, patterns of behaviour have self-monitoring facilities. The conditions of an active pattern are monitored by so-called *guards*, which are basically knowledge modifiers, i.e. patterns of behaviour which become active when the respective condition (e.g. the termination condition of the parent pattern) becomes true. For example, the termination condition for a PoB for moving to a landmark is satisfied when the agent has reached the landmark. This again is monitored by a guard pattern.

An important problem is the interference between concurrent patterns of behaviour. For example, approaching a shelf to grasp a box may conflict with an obstacle avoidance pattern. In the literature, several mechanisms for dealing with this problem are known. For example, DASEDIS [BS92] supports a simple static ordering of possible scripts describing agent behaviour based on a total ordering of intentions. Brooks [Bro86] has proposed a more sophisticated suppression/inhibition mechanism for the coordination between patterns of behaviour of different layers. This, however, also leads to a static, hard-wired connection between patterns. Moreover, it is not trivial for a pattern of behaviour to determine which other pattern could possibly lead to a forbidden world state. Our approach is to specify in the `:during` condition a formula describing a set of forbidden world states for each active pattern. This state description is made public to all other patterns of behaviour. Thus, each time a pattern of behaviour tries to execute a primitive action, it has to check locally whether performing this action would lead to a forbidden world state. In order keep this procedure manageable for practical applications, the algorithm for determining the truth of the monitoring conditions as well as the representation of the monitoring conditions have to be restricted. In our system, representing the world model as a set of ground propositions, and implementing the truth predicate via database matching (as provided e.g. by OPS-5) turned out to be a viable tradeoff balancing expressiveness and efficiency.

3.2.2 Implementation

In the following, we will outline the instantiation of the abstract model of the BBC that we implemented for the FORKS system. The control part of the BBC is realized based on the rule selection and interpretation strategy of OPS-5 implemented in the MAGSY basic agent language [Fis93]. A pattern of behaviour is represented as a rule set using a context maintenance algorithm based on the MEA strategy. Specific rules of this rule set monitor the activation and monitoring conditions. The execution body is formulated using OPS-5 based on the world interface primitives. Reactive, data-driven, blackboard-like systems can be implemented very conveniently based on OPS-5. Currently, we are working on a higher-level pattern specification language similar to the one shown above, that can be compiled down to OPS-5 or C.

3.3 The Plan-Based Component

The plan-based component of the INTERRAP model incorporates the agent's local planning facilities. It has two main functionalities according to its two neighbour modules, the BBC and the CC. Firstly, it has to devise a plan for a goal upon request from the BBC (call `do(Goal)`) and to control the correct execution of this plan. Secondly, it has to interpret the agent's part in a joint plan conceived by the CC. A complete specification of the PBC interface is provided by [MP93]. Note that, at this stage, INTERRAP does not tie the system designer down to a specific planning formalism. Rather, any mechanism can be used that can be adapted to this interface specification. Since one of our major concerns when modeling the FORKS domain has been to keep the planning process tractable by incorporating domain knowledge, we chose an approach to planning from second principles: the actual planning task is reduced to selecting from a plan library and instantiating a suitable plan indexed by the agent's current goal. The assumption behind this procedure is that of *universal plans* [Sch89], i.e. plans that will achieve their goals irrespective of the initial state of the world.

In the rest of this section, we will define an instantiation of the PBC for the loading dock domain.We describe the internal structure of the PBC, the plan representation chosen, and we discuss plan execution.

3.3.1 Internal Structure of the PBC

It consists of a planning control module *pbc-control*, of a *plan generator* module, and a *plan evaluator* module. The *pbc-control* contains the PBC interface and the plan interpreter. The interface receives messages from and sends messages to the CC and the BBC. The plan interpreter controls the processing and the decomposition of a plan. 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. In each cycle, the interpreter chooses one of the goal stacks and processes the top goal of this stack. Processing a goal means either (1) to expand the goal into subgoals, or (2) to activate a pattern of behaviour.

Once activated by a call `do(Goal)` from the BBC, the *pbc-control* passes a request to the *plan generator*. In the FORKS application, the task of the plan generator is merely to select and to instantiate a set of plans for `Goal` from a plan library. The FORKS local planner is hierarchical for it performs a one-step expansion every time it is called, i.e. the current goal is expanded into its direct subgoals only. Using a decision-theoretic evaluation model, the best of these plans is selected by the *plan evaluator*.

3.3.2 Plan Representation

In [McD91, LH92], requirements on a robot planning language were described. Based on this, we defined the plan language $\mathcal{P}_0$ which allows to build plan expressions starting from primitive plan steps (abstract actions) $p, p_1, p_2, \ldots$ and conditions (state formulae) $c, c_1, c_2, \ldots$ using sequential ($[p_1, p_2]$) and disjunctive ($[p_1; p_2]$) composition of plan steps, tests (**if** c **then** p_1 **else** p_2) and iteration (**while** c **do** p) commands.

A skeletal plan library has been defined for the loading dock application. It consists of a set of entries `plan-lib ::= ( lpb-entry`$_1$`, ..., lpb-entry`$_k$ `)`. 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 is a $\mathcal{P}_0$ plan expression which species the expansion of the entry plan step. `Type = b` denotes that Body is an abstract action, implemented by an executable pattern of behaviour in the BBC. Figure 3 shows an excerpt from the FORKS 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,
        [do(random_search(B))]). ;; pattern of behaviour
        ...
lpb_entry(goto_landmark(L1), b,
        [do(goto_lm_beh(L1))])
lpb_entry(goto_landmark(L1), s,
        [rr(where_am_i(?L0)), do(gen_moves(L0, L1))])
        ...
```

Figure 3: Exemplary Plan Library

Thus, from the (top-down) point of view of the planner, the behaviour-based component is a library implementing the abstract actions specified in the local plans. A pattern of behaviour corresponds to a procedure which the PBC may call, and which terminates with either a success or a failure. Calling such an abstract or primitive action is done by an `activate(bbc, Name_of_Behaviour(Args))` message. The planner then waits for the termination report by the pattern of behaviour which has been activated. This waiting is asynchronous, i.e. the planner can work on other goals and accept new planning tasks by other patterns of behaviour in the meantime.

4 Integrating Interaction

Some situations occurring in a multi-agent world exceed the problem-solving capacities of the local planner in that they require interactive capabilities. These situations are described, recognized and handled from the local point of view of an agent by so-called *patterns of interactions* at three different layers which correspond to the three knowledge-based layers of the INTERRAP model: the *situational context* of a pattern of interaction describes the external situation by which the interaction is characterized; the *mental context* describes the current goals of the agent which are affected (endangered, blocked, or supported) by the interaction; In some cases, knowledge of the situational context is sufficient to recognize and to handle an interaction: for example, if a forklift approaches to another forklift f_a very quickly, f_a has to dodge - no matter what goals it has. For others, knowledge about the goals of the agent that perceives the interaction is required: the pure fact that another agent stands in front of forklift f_a is not sufficient to recognize a blocking conflict - unless f_a has the goal to move ahead. The recognition and treatment of other interactions requires additional knowledge about the goals of other agents. For example, a blocking conflict between two forklifts in a narrow shelf corridor may be resolved by cooperation when the goals of both agents are known. The INTERRAP model supports a hierarchical model of the recognition and handling of interactive situations: the situational context of a pattern of interaction is represented at the behaviour-based layer; knowledge about the agent's local goals is maintained at the local planning layer; knowledge about the *social context* (goals of other agents) is kept at the cooperation layer. Moreover, different mechanisms of interaction can be specified at different layers, from simple local mechanisms in the BBC (random moves, waiting behaviors for conflict resolution) to the construction and negotiation of joint plans in the CC. For a more detailed description of the model of interaction underlying the INTERRAP model we refer to [Mü94b].

In this section, we describe the functionality and the structure of the CC, the representation of joint plans as well as subjects of plan evaluation and execution which have to be dealt with when describing plans by and for multiple agents.

4.1 The Cooperation Component

The main functionality of the CC is that it has to devise a *joint plan* for a certain situation and upon request by the PBC, given a description of the situational context and of the mental context. The social context (goals of other agents involved) has to be provided and checked by the cooperation component by evaluating available information about other agents.This classification process results in a type of interaction, which is used as an index to an entry in a library of joint plans. The basic parts of the CC are the *cc-control*, the *joint plan generator*, the *joint plan evaluator*, and the *joint plan translator*. The *cc-control* is the interface between the CC and the PBC. Moreover, it coordinates the work of the other submodules of the component. The *joint plan generator* has to return a set of joint plans for a given situation specification which satisfies the goals of all agents involved. As mentioned earlier, our objective has been to keep planning tractable by utilizing domain-specific information, and to concentrate on the interplay among the modules. Therefore, for the FORKS domain,

the plan generator selects a set of joint skeletal plans indexed by the situation specification from a joint plan library rather than planning from first principles. The best of these plans is then selected by the *plan evaluator* (see also subsection (4.3)). After a negotiation phase, where the agents involved in the interaction agree on a joint plan to execute, the *joint plan translator* transforms the joint plan into single-agent plans which are interpreted and executed by the PBCs of the respective agents.

4.2 Representation of Joint Plans

In the sequel, we define a simple plan language for joint plans. Let $A = \{a_1, \ldots, a_n\}$ be a set of agents, $S = \{s_1, \ldots, s_k\}$ be a set of abstract actions, the primitives of our plan language. For each occurrence of an $s \in S$ in a plan P, $s = s_a$ is labelled by the agent a that executes s in the plan.

Definition 1 Let $s, s_1, s_2 \in S$. Then plan language $\mathcal{P}_1$ is defined as follows:

- $\perp \in \mathcal{P}_1$ (empty plan body).
- $s \in \mathcal{P}_1$ for a primitive plan step $s \in S$.
- Let $s_1, s_2 \in \mathcal{P}_1$. Then $[s_1, s_2] \in \mathcal{P}_1$ (sequential composition of plan steps).
- Let $s_1, s_2 \in \mathcal{P}_1$. Then $[s_1; s_2] \in \mathcal{P}_1$ (non-deterministic alternative composition of plan steps).
- Let $s_1, s_2 \in \mathcal{P}_1$. Then the simultaneous composition $[s_1 \| s_2] \in \mathcal{P}_1$.
- Let $s_1, s_2 \in \mathcal{P}_1$, let e be an arbitrary predicate. Then [**if** e **then** s_1 **else** s_2] $\in \mathcal{P}_1$ (conditional branch).
- Let $s \in \mathcal{P}_1$, let e be an arbitrary predicate. Then [**while** e **do** s] $\in \mathcal{P}_1$ (while-loop).

$\mathcal{P}_1$ is a linear plan language, since there is no means to express that two actions s_i, s_j performed by two agents i, j are independent, i.e. can be executed concurrently. Plan language $\mathcal{P}_2$ remedies this:

Definition 2 ($\mathcal{P}_2$) Let $A = \{a_1, \ldots, a_m\}$ be a set of agents. The plan language $\mathcal{P}_2$ is defined as follows:

- $[] \in \mathcal{P}_2$ (empty plan step).
- Let $l_1, \ldots, l_m \in \mathcal{P}_1$ be plan bodies, so that for each l_i, $i \leq m$
  - a single agent a_i performs plan body l_i (i.e. all role variables occurring in plan steps in l_i are instantiated with the singleton set $\{a_i\}$), or
  - l_i contains only plan steps labelled by agent a_i *and* concurrent 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 main idea is that a plan step does no longer denote one primitive plan step performed by one agent, but rather represents a set of partial plans (one for each agent participating in the joint plan) which can be executed concurrently.

4.3 Plan Evaluation

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 a reasonable plan is. It is the task of the joint plan evaluator to determine whether a plan is reasonable by computing its utility. The evaluator accepts as input a list $[P_1, \ldots, P_k]$ of joint plans proposed for achieving a goal, and returns a list of evaluated plans $[(P_1, e_1), \ldots, (P_k, e_k)]$ where e_i is the utility ascribed to P_i. Defining utilities for plans is a complicated matter. In general, the utility of a plan P can be computed as the difference between its worth W and its cost C [ZR91]. A cost function for a plan can often be computed in a straightforward manner: for example, in the current FORKS implementation, the utility of a conjunctive[4] plan P is defined by $C(P) = \sum_{p \in P} c(p)$, where p are primitive actions in P, where we assume that a cost function c on primitive plan steps is given. The worth of a plan, however, depends on the worth of the goals that are achieved by this plan and is often much more difficult to be obtained. We refer to [HH94] for the combination of utilities and goals and to [Mü94a] for a more complete picture of plan evaluation in the framework of INTERRAP.

4.4 Plan Transformation and Execution

It is the task of the translator module to transform a joint plan into a single-agent plan by projecting the agent's part of the joint plan and by adding synchronization actions which guarantee that the constraints contained in the original joint plan are satisfied during plan execution. The translation algorithm can be found in [Mü93]. In the following, we will show by means of an example how a blocking conflict between two forklift agents is resolved by a joint plan.

Let us assume that another agent, say j, blocks the way of agent i in a narrow corridor between two shelves. In this situation, the BBC recognizes a conflict and calls the PBC by a request `do(resolve_shelf_conflict((self, Agent)))`. The *pbc_control* recognizes that it does not have a plan for solving a blocking conflict in a shelf corridor. Therefore, it shifts control to the cooperation component which devises a joint plan, and initiates a negotiation with the other agent on the joint plan. In the simplest case, negotiation may consist of simple acceptance of 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, i.e. a plan with sufficiently high utility for all agents involved. The resulting joint plan is translated by the CC into a single-agent plan which is augmented by synchronization commands. For example, a simple joint plan for resolving the conflict situation shown in figure 4, allowing two agents two agents i and j to change places (represented in plan language $\mathcal{P}_2$) is

```
JP_{i,j} = [[[move_aside(north)], []],
            [[walk_ahead], [walk_ahead]],
            [[move_aside(south)], []]].
```

[4] A probabilistic evaluation model is used to evaluate tests and iteration plan steps.

The plan JP has three plan steps. The separation in plan steps implements a precedence ordering $\sqsubset_t$ on plan steps. The transition between plan steps one and two describes the precedence constraint $walk\_aside(north)_i \sqsubset_t walk\_ahead_j$. Note that our representation allows concurrent actions of i and j in step two. The transition to step three corresponds to the constraint $walk\_ahead_j \sqsubset_t walk\_aside(south)_i$. This joint plan is translated into the following single-agent plan P for the agents by extracting their respective parts of the plan and by enhancing it with synchronization commands. These ensure that the precedence constraints expressed by the joint plan are respected in the execution. In the case of agent i, the translated plan looks as follows:

```
Pi = [move_aside(north)
      send_synch(j, ready),     ;; send message to agent j
      walk_ahead,
      wait_synch(j, ready),
      move_aside(south)].
```

The translation algorithm inserts a *wait* command in front of each plan step p for which there exists a plan step p' so that $p' \sqsubset_t p$. A *send* command has to be inserted after each plan step q for which there exists a plan step q' so that $q \sqsubset_t q'$. The translated plan is passed to the PBC which interprets it by again activating appropriate patterns of behaviour as described in section 3. Currently, a new joint plan language is under development which allows to explicitly express the constraints and which allows to specify arbitrary constraints on plan steps.

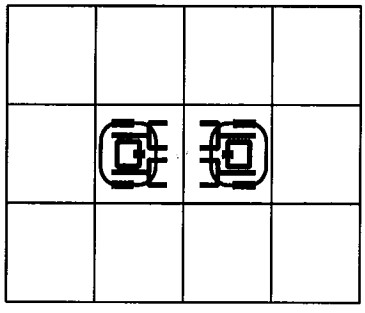

Figure 4: A Conflict Situation

Obviously, joint plans are not the only appropriate way of interacting. A great deal of the interaction in a robot environment will be described by local, behaviour-based mechanisms in the behaviour-based component of the INTERRAP architecture. However, we claim that the deliberative methods of interaction maintained in the CC become necessary for intelligent conflict resolution in highly constrained situations and for reasonable collaboration between autonomous agents.

5 Related Work

Our work is related with principles of architectures for modeling interacting autonomous systems with reactive and deliberative capabilities, and is this way related to work on cooperation and agent architectures done in DAI (see [IGY92, Jen92] to mention just a few of them) as well as to research on dynamic reactive planning. Our research has extended work on reactive systems (see e.g. [Bro86, Fer89]) by adding a deliberative component.

Classical AI planners usually consist of a plan generation module and a plan interpretation module. Plans are basically sequences of primitive actions. Since in many real-world domains, information comes in incrementally, other approaches have tried to interleave plan construction and execution (e.g. [DL86]). Georgeff and Lansky [GL86] proposed the use of *precompiled methods* in order to be able to cope with real-time constraints. Recently, architectures for reactive planning have been proposed which have shown new ways to integrate aspects of deliberation and reaction. This development has led to a general architecture for these kind of systems. A reactive planning system consists of a planner and a reactor module. 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. the SOAR system [Lai90]), or there is a planner with an associated mechanism for interrupt handling and replanning [PR90]. Other recent approaches seem to be closer to our model, since the planner and the behaviour-based component (reactor) run in parallel: e.g. in [LH92], planning is viewed as adapting the agent's reactions. Firby [Fir92] has presented a layered robot architecture which consists of a planner working with sketchy plans and an executor that activates robot control programs. He stresses the interface between symbolic (abstract actions) and continuous (control programs). Ferguson [Fer92] extends the reactor-planner architecture by a component called *modeling layer*. This layer is responsible for achieving adaptive behaviour and constitutes a first step towards the integration of learning into the system. Our main point is that there is no way to clearly model knowledge and strategies for coordination and cooperation in any of the above systems.

The idea of integrating a cooperation layer has been advocated for by Jennings (see e.g. [Jen92]). However, Jennings' work in the ARCHON system was primarily concerned with defining such a layer on top of a given system, whereas our research focusses on from-scratch system design. Our modeling of the cooperation component itself is closely related to work described by [Geo83] and by [KLR+92] as regards aspects of cooperative planning.

6 Evaluation and Outlook

In this paper, the INTERRAP agent model was presented and evaluated by means of a robotics application, the loading-dock domain. We consider the main contribution of our work to be the following: Firstly, the scope of layered agent architectures has been extended by the integration of an additional cooperation layer: this allows a hierarchical recognition and deliberative treatment of a bulk of interactive situations such as conflicts and potential cooperations. Secondly, the architecture allows the

designer of an agent to provide agents with application-specific functionalities at the appropriate layers. It describes a first step towards a flexible system development shell.

Currently, the INTERRAP agent architecture does not provide a learning layer. One reason for this is self-restraint: the extension to cooperation seemed to guarantee sufficient interesting problems. On the other hand, it is not easy to see how learning will fit into the layered framework. Since learning should be possible at each layer of the agent, it is a currently open design decision whether the model should be extended by a *learner component* (LC) on top of the CC, or whether learning should be treated as an orthogonal concept which should be handled by an additional submodule in each of the INTERRAP modules. This extension is an interesting topic for future work. We are currently evaluating the simulation results in a real miniature robot environment. This requires a thorough adaptation of the world interface layer, whereas only little modification is necessary in the symbolic layers of the INTERRAP model. Finally, further extensions will deal with the integration of mechanisms for planning from first principles into the local as well as the cooperative planning framework.

References

[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.

[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.

[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.

[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.

[Fir92] R. James Firby. Building symbolic primitives with continuous control routines. In J. Hendler, editor, *Proc. of the First International Conference on AI Planning Systems.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[Fis93] K. Fischer. The Rule-based Multi-Agent System MAGSY. In *Proceedings of the CKBS'92 Workshop.* Keele University, 1993.

[FKM94] K. Fischer, N. Kuhn, and J. P. Müller. Distributed, knowledge-based, reactive scheduling in the transportation domain. In *Proc. of the Tenth IEEE Conference on Artificial Intelligence and Applications*, San Antonio, Texas, March 1994.

[Geo83] M. Georgeff. Communication and interaction in multi-agent plans. In *Proc. of IJCAI-83*, pages 125–129, 1983.

[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.

[HH94] P. Haddawy and S. Hanks. Utility models for goal-directed decision-theoretic planners, 1994. Submitted to *Artificial Intelligence* journal.

[IGY92] T. Ishida, L. Gasser, and M. Yokoo. Organization self-design of dstributed production systems. *IEEE Trans. Knowledge and Data Engineering*, 4(2):123–133, 1992.

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

[KLR+92] D. Kinny, M. Ljungberg, A. Rao, E. Sonenberg, G. Tidhar, and E. Werner. Planned team activity. In A. Cesta, R. Conte, and M. Miceli, editors, *Pre-Proceedings of MAAMAW'92*, July 1992.

[Lai90] J. Laird. Integrating planning and execution in SOAR. In J. Hendler, editor, *AAAI Spring Symposium on Planning in Uncertain and Changing Environments*, University of Maryland, Systems Research Center, Stanford CA, 1990.

[LH92] D. M. Lyons and A. J. Hendriks. A practical approach to integrating reaction and deliberation. In *Proc. of the 1st International Conference on AI Planning Systems (AIPS)*, pages 153–162, San Mateo, CA, June 1992. Morgan Kaufmann.

[McD91] D. McDermott. Robot planning. Technical Report 861, Yale University, Department of Computer Science, 1991.

[MP93] J. P. Müller and M. Pischel. The Agent Architecture INTERRAP: Concept and Application. Technical Report RR-93-26, German Artificial Intelligence Research Center (DFKI), Saarbrücken, June 1993.

[Mü93] J. P. Müller. Rational interaction via joint plans. In *Workshop on Communication, Coordination, and Cooperation in Multiagent Systems*, Berlin, September 1993. KI-93, 17. Fachtagung der Künstlichen Intelligenz.

[Mü94a] J. P. Müller. Evaluation of plans for multiple agents (preliminary report). In K. Fischer and G. M. P. O'Hare, editors, *Working Notes of the ECAI Workshop on Decision Theory for DAI Applications*, Amsterdam, NL, August 1994. Forthcoming.

[Mü94b] J. P. Müller. A model of interaction for dynamic multi-agent environments. In *Preproc. of the 2nd Intl. Working Conference on Cooperating Knowledge-based Systems (CKBS'94)*, Keele, 1994. Forthcoming.

[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.

[Sch89] M. Schoppers. Representation and automatic synthesis of reaction plans. Technical Report UIUCDCS-R-89-1546 (phd-thesis), Dept. of Comuter Science, University of Illinois at Urbana-Champaign, 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.