A Pragmatic BDI Architecture*

Klaus Fischer, Jörg P. Müller**, Markus Pischel

DFKI GmbH, Stuhlsatzenhausweg 3, D-66123 Saarbrücken

Abstract. We present a unifying perspective of the individual control layers of the agent architecture INTERRAP. INTERRAP aims at modeling autonomous resource-bounded agents that interact with each other in dynamic multiagent environments. INTERRAP implements a pragmatic Belief-Desire-Intention (BDI) architecture, where the agent's mental state is distributed over a set of layers. Based on the processes of *situation recognition* and *planning and scheduling*, a uniform description for each control layer – the behavior-based layer, the local planning layer, and the cooperative planning layer – is provided. We demonstrate various options for the design of interacting agents within this framework in an interacting robots application. The performance of different agent types in a multiagent environment is experimentally evaluated.

1 Introduction

The design of intelligent agents is an important research direction within multiagent systems (MAS) [6], where the behavior of a society of agents is described by modeling the individuals and their interactions from a local, agent-based perspective. Thus, finding appropriate architectures for these individuals is one of the fundamental research issues within agent design.

There are at least two reasons for dealing with agent architectures: One is to explain and to predict agent behavior; this means to describe how an agent's decisions are derived from its internal (mental) state and how this mental state is affected by the agent's perception. The other reason is to actually support the design of MAS. It deals with providing tools and methodologies for designing computational agents and their interactions in an implemented system.

A prominent example for architectures that are primarily driven by the former reason are BDI architectures [3, 15], describing the internal state of an agent by the mental attitudes of *beliefs*, *goals*, and *intentions*. BDI theories provide a clear conceptual model of the knowledge, the goals, and the commitments of an agent. However, they offer little guidance to the modeling of motivation and intention formation; thus, they have to be extended to actually support the design of resource-bounded and goal-directed agents for practical applications.

Another important direction in intelligent agent design are layered architectures (see Section 7). Layering is a powerful concept for the design of resource-bounded agents.

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

** email: jpm@dfki.uni-sb.de, phone: ++49 681 302 5331, fax: ++49 681 3025341

It supports a natural modeling of different levels of abstraction, responsiveness, and complexity of knowledge representation and reasoning. However, a recent criticism of layered architectures has been that they are mainly motivated by intuition, and that they are too complex to allow the formal investigation of properties of agents and multiagent systems [19].

The agent architecture INTERRAP which is described in this paper aims at combining the advantages of BDI-style architectures with those of layered ones. Thus, our goal is to provide an architecture that serves both to explain agent behavior and to support system design. INTERRAP adopts the mental categories used in BDI theory to describe an agent's knowledge, goals, and state of processing. It extends the work of [15, 16] by organizing an agent's state and control within a layered architecture. The problem-solving capabilities of an agent are described hierarchically by a behavior-based layer, a local planning layer, and a cooperative planning layer. INTERRAP adopts the BDI-model rather in a conceptual than in a strictly theoretical sense. Thus, this paper does not provide a new theory for beliefs, desires, and intentions, but takes a pragmatic perspective.

Previous work [12, 13] has described the basic layered structure of the INTERRAP architecture and a first simple concept and implementation of the individual control layers. In this paper, we present a redesign of INTERRAP aimed to make the architecture easier to describe and to make agents easier to analyze by providing a clear control methodology. Using the FORKS application describing an automated loading dock as an example, we then show how different agent types can be described using the control framework and we provide empirical results comparing their behavior in the loading dock.

2 The INTERRAP Agent Architecture

INTERRAP is an approach to modeling resource-bounded, interacting agents by combining reactivity with deliberation and cooperation capabilities. This section illustrates the basic concepts of the architecture. Due to space limitations, the discussion is kept somewhat superficial. We refer to [12] for more details.

2.1 Overview

Figure 1 illustrates the overall structure of the architecture. INTERRAP describes an agent by a world interface, a control unit, and a knowledge base (KB). The control unit consists of three layers: the behavior-based layer (BBL), the local planning layer (LPL), and the cooperative planning layer (CPL). The agent knowledge base is structured correspondingly in a world model, a mental model, and a social model. The different layers correspond to different functional levels of the agent. The purpose of the BBL is to allow the agent to react to certain critical situations (by so-called *reactor patterns of behavior* (PoBs)), and to deal with routine situations (using *procedure PoBs*). Reactors are triggered by events recognized from the world model that incorporates the agent's object-level knowledge about its environment. The LPL gives the agent the ability of longer-term deliberation. It builds on world model information, but additionally uses

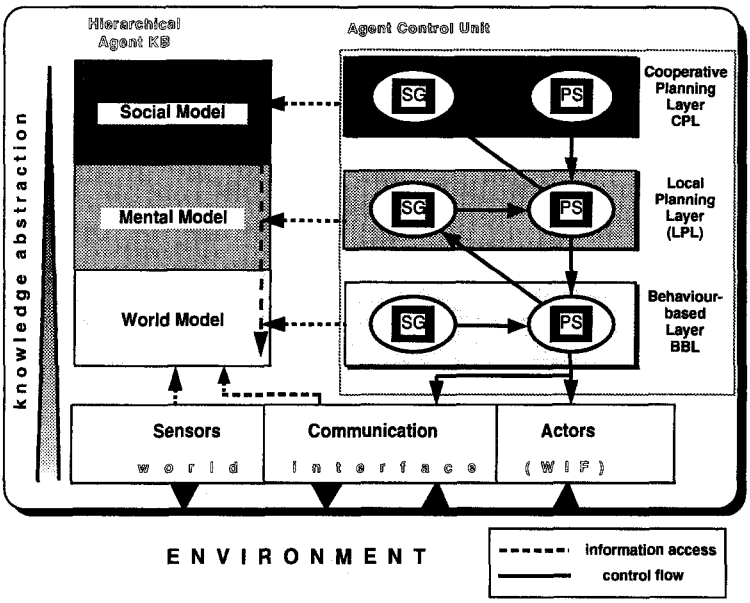

Fig. 1. The INTERRAP agent architecture

the agent's current goals and local intentions maintained in the mental model part of the knowledge base, as well as domain-dependent planning mechanisms available. The CPL finally extends the planning functionality of an agent to *joint plans*, i.e., plans by or for multiple agents that allow to resolve conflicts and to cooperate. Apart from world model and mental model knowledge, the CPL uses information about other agents' goals, skills, and commitments stored in the social model of the knowledge base. The internal structure of the control components is explained in more detail in the following sections of this paper.

In the following, let $\mathcal{B}$, $\mathcal{G}$, $\mathcal{I}$ denote the beliefs, goals, and intentions of an agent, respectively, and let $\mathcal{P}$ denote a set of perceived propositions. The INTERRAP agent architecture implements three basic functions:

- $BR(\mathcal{P}, \mathcal{B}) = \mathcal{B}$ is a belief revision and knowledge abstraction function, mapping an agent's current perception $\mathcal{P}$ and its old beliefs $\mathcal{B}$ into a set of new beliefs $\mathcal{B}'$.
- $SG(\mathcal{B}, \mathcal{G}) = \mathcal{G}'$ is a situation recognition and goal activation function, deriving new goals $\mathcal{G}'$ from the agent's beliefs $\mathcal{B}$ and its current goals $\mathcal{G}$.
- $PS(\mathcal{B}, \mathcal{G}, \mathcal{I}) = \mathcal{I}'$ is a planning and scheduling function, deriving a set $\mathcal{I}'$ of new intentions (commitments to courses of action) based on the beliefs $\mathcal{B}$, the goals $\mathcal{G}$ selected by SG, and the current intentional structure $\mathcal{I}$ of the agent.

Table 1 shows how the functions defined above are distributed over the individual layers. In this paper, we focus on the functions SG and PS. For issues of knowledge representation and belief revision, we refer to [9].

Layer / Function	BBL	LPL	CPL
BR	generation and revision of beliefs (world model)	abstraction of local beliefs (mental model)	maintaining models of other agents (social model)
SG	activation of reactor patterns	recognition of situations requiring local planning	recognition of situations requiring cooperative planning
PS	reactor PoB: direct link from situations to action sequences	modifying local intentions; local planning	modifying joint intentions; cooperative planning

Table 1. The basic functions in the INTERRAP control hierarchy

2.2 The control layers

The processes implemented at the different layers of the INTERRAP architecture have many similarities in that they describe different instantiations of the basic functions SG and PS. Based on this observation, we present a uniform structure shared by each layer. Figure 2 shows the internal structure of an INTERRAP control layer. Each layer $i \in \{BBL, LPL, CPL\}$ consists of two processes implementing the functions SG and PS; these interact with each other and with processes from neighbor layers:

- The **situation recognition and goal activation** process SG_i recognizes situations that are of interest for the respective layer; it results in the activation of a goal.
- The **planning and scheduling** process PS_i implements the mapping from goals to intentions and thus, to actions. It receives as input goal–situation pairs created by the SG component of the layer; it determines the plans to achieve the goals, schedules them into the current intention structure of the agent, and monitors the execution of plan steps.

The implementation of the two functions in INTERRAP is explained in more detail in Sections 3 and 4.

2.3 The flow of control

The control flow and thus the behavior of an INTERRAP agent emerges from the interaction among the individual modules as illustrated in Figure 2. The model provides two basic protocols specifying the global flow of control[3].

Upward Activation Requests: If PS_i is not competent for a situation S, it sends an activation request containing the corresponding situation-goal pair to SG_{i+1}; there, the situation description is enhanced by additional knowledge available to this component in order to produce a suitable goal description. The result of processing S is reported back to PS_i. This mechanism implements a *competence-based* control mechanism.

[3] Additional, more specific protocols cannot be discussed here due to space restrictions.

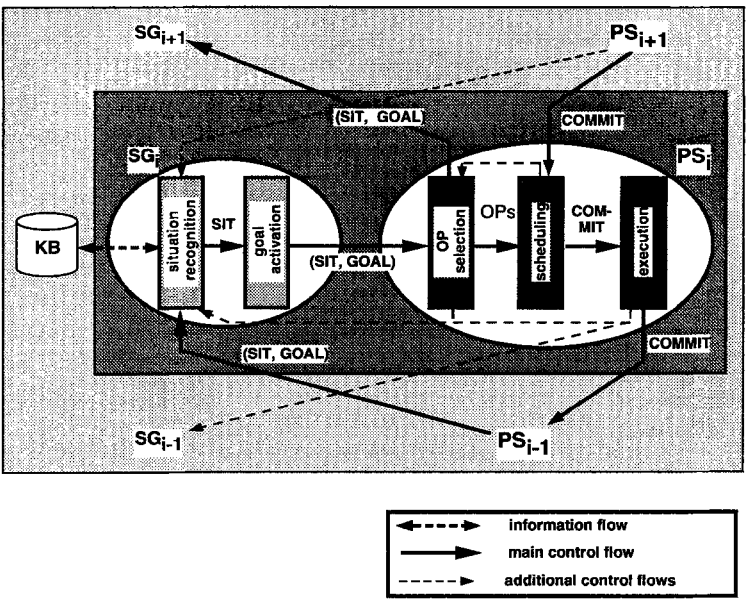

Fig. 2. Structure of an INTERRAP control layer

Downward Commitment Posting: Planning and scheduling processes at different layers coordinate their activities by communicating commitments. For example, this allows the local planning component both to integrate partial plans devised by the CPL layer in the course of a joint plan negotiation and to take into account certain commitments made by the upper layer (integrity constraints). Also the interface between the LPL and BBL component is designed by the higher layer posting activation requests for patterns of behaviors.

2.4 Coherence

The coherence problem results from the concurrent access to actions, perception, and knowledge by a set of layers, possibly leading to different results of situation recognition, to inconsistent decisions, and thus, to an incoherent behavior of the agent. Thus, the question is how coherent agent behavior can be achieved, i.e., how to coordinate situation recognition and the authority to perform actions.

The hierarchical control regime of INTERRAP allows to simplify some of these problems by restricting concurrency in activation and by restricting the concurrent access to the actuators. It allows us to deal without global control rules as in [7]. The main idea for coordination between reactive and plan-based layers is to give priority to the actions proposed by the BBL, and to allow a posteriori correction by the LPL [5]. In order to avoid foreseeable harmful interactions between the LPL and the BBL, there is the possibility of explicit suppression [4] which allows the LPL to enable or disable

PoB in certain situations by sending appropriate messages to the BBL. Moreover, PoB that are no longer useful from the planner's point of view can be cancelled.

3 Situation Recognition

Situations are described from the view of an individual agent. A situation S is a set of formulae $S \equiv S_B \cup S_L \cup S_C$ with $S_B \subseteq WM$, $S_L \subseteq MM$, and $S_C \subseteq SM$[4]. It describes a portion of the agent KB containing parts of its world model, mental model, and social model. The world model part (*external context*) of a situation is a set of ground atomic formulae; the mental model part (*mental context*) describes parts of the local intention structure of the agent, i.e., goals and intentions; the social model part (*social context*) describes belief about other agents characterizing a specific situation.

Classes of situations are denoted by formulae in a first-order language $\mathcal{L}$, so-called *situation descriptions*. Situation descriptions provide patterns that can be instantiated to situations. For each layer i within the INTERRAP hierarchy, a set $\mathcal{D}_i \subseteq 2^{\mathcal{L}}$ of situation descriptions is defined that are recognized by this layer. Let $\mathcal{T}$ denote a set of time points. The semantics of the function SG_i is defined by a function $OCC_i : 2^{\mathcal{L}} \times 2^{\mathcal{L}} \times \mathcal{T} \mapsto 2^{\mathcal{L}}$. $OCC_i(B_i^t, D_i, t) = \mathcal{S}'$ returns the subset $\mathcal{S}'$ of instantiations of a situation description $D_i \in \mathcal{D}_i$ which occur at time t, i.e., which can be derived from the set of beliefs B_i^t at time t. At layer i, situations are mapped to goals by a function $\beta_i : \mathcal{S}_i \mapsto \mathcal{G}_i$, where the function $SG_i : 2^{\mathcal{L}} \times \mathcal{T} \times 2^{2^{\mathcal{L}}} \times 2^{\mathcal{L}} \mapsto 2^{2^{\mathcal{L}} \times 2^{\mathcal{L}}}$ is defined as

$$SG_i(B_i^t, t, \mathcal{D}_i, \mathcal{G}_i) \stackrel{\text{def}}{=} \{(S, G) | \exists D \in \mathcal{D}_i \exists G \in \mathcal{G}_i . S \in OCC_i(B_i^t, D, t) \wedge G = \beta_i(S)\}.$$

Thus, given the beliefs, the situation descriptions to be monitored, and the potential goals the agent may adopt at time t, the output of function SG is a set of situation–goal pairs, namely the pairs (S, G) where situation S instantiates one of the input situation descriptions, and where situation S is mapped to goal G by the goal activation function.

Differences between the control layers result from restrictions on the admissible form of the set B_i^t and from the implementation of OCC_i. For the BBL, we have $B_B^t \subseteq WM$. For the LPL, we have $B_L^t \subseteq WM \cup MM$. Situation recognition in the CPL may access the whole knowledge base: $B_C^t \subseteq WM \cup MM \cup SM$.

OCC_B is defined by $OCC_B(B_B^t, D_B, t) = S$ iff $S = D_B\theta$ for a ground substitution θ. This many-pattern, many-objects matching problem can be solved e.g., by the RETE algorithm, allowing fast recognition of situations that have to dealt with quickly at the behavior-based layer. On the other hand, OCC_L and OCC_C include checking whether the agent itself has a specific goal or an intention, or even if other agents have certain goals or intentions. For OCC_L we assume that local goals are also represented as ground formulae; moreover, we require that an agent explicitly knows all its goals and intentions. In the case of OCC_C, however, more complex, time-consuming deduction may be necessary e.g., in order to recognize other agents' goals, either through communication, or through explicit goal recognition techniques.

Situation recognition is an incremental process, i.e., partial situations may be recognized at lower layers and complemented at higher layers. The SG_i process outputs

[4] We use the subscripts B for BBL, L for LPL, and C for CPL.

pairs (S, G). A goal G is associated to each situation S recognized by SG_i. This pair characterizes a new option to be pursued by the agent. It serves as an input to the planning and scheduling process described in the sequel.

4 Planning and Scheduling

According to Figure 2, at any point in time, the planning and scheduling process PS_i of layer i may receive input from two possible sources: situation-goal pairs from the SG_i process and commitment messages from the planning and scheduling process PS_{i+1} at the next higher layer. The output of PS_i are situation-goal pairs which are sent to SG_{i+1} and commitments to PS_{i-1}. PS_i maintains an intention structure which informally can be looked upon as the agent's runtime stack, holding the agent's current goals $\mathcal{G}_i$ and its intentions $\mathcal{I}_i$, denoting its state of planning and plan execution. Each situation–goal pair (S, G) received from SG_i at time t is processed according to the following steps:

1. If layer i is competent for (S, G), continue with step 2; otherwise send an upward activation request `request(do(S, G))` to SG_{i+1}; RETURN
2. Add G to the set $\mathcal{G}_i$.
3. Select an subset $\mathcal{G}' \in \mathcal{G}_i$ for being pursued next and devise a partial plan P' for achieving the goals[5] in $\mathcal{G}'$ given the current intention structure $\mathcal{I}_i$.
4. Compute the modified intention structure $\mathcal{I}_i'$ and thus, the next commitment.

This procedure is basically the same for the planning and scheduling modules at any layer; however, as is outlined in the sequel, the individual steps are implemented in a different manner.

4.1 Competence

The competence-based control flow is a central feature of INTERRAP. Each layer can deal with a set of situations, and is able to achieve a set of goals. The competence of layer i for a situation–goal pair (S, G) is decided by a predicate $\chi_B : \mathcal{S} \times \mathcal{G} \mapsto \{0, 1\}$. The competence predicates for the individual layers are defined as follows:

$\chi_B(S, G) = 1$ iff ex. a reactor PoB whose activation condition matches G.
$\chi_L(S, G) = 1$ iff ex. a single-agent plan p_s that achieves G given start situation S.
$\chi_C(S, \{G_1, \ldots, G_n\}) = 1$ iff ex. a joint plan p_j that achieves $\bigcup_{i=1}^{n} G_i$ given S.

If $\chi_i(S, G) = 0$ for a situation S and goal G, the layer is not competent for this situation/goal; then, an activation request containing (S, G) is sent to SG_{i+1}, notifying this layer of the new situation. χ_B can be computed by a simple matching; thus, it is possible to make decisions quickly at the reactive layer. However, constructing a plan may be necessary in order to determine χ_L and χ_C. These functions can be augmented by not only requiring the existence of a plan, but also requiring a minimal quality of the plan based on a utility function $u : PLANS \mapsto \mathbb{R}$. This is useful for an agent in order to decide whether to start a cooperation in a certain situation because there is only a poor local solution.

[5] Here, we assume that the goals in $\mathcal{G}'$ can be achieved independently of each other.

4.2 Deciding what to do

After a layer has decided to be competent for a situation, the planning process starts resulting in a commitment, e.g., a decision to perform a certain action. This planning process differs throughout the INTERRAP layers: At the BBL, patterns of behavior provide direct hard-wired links from situations to compiled executable procedures; thus, they ensure high responsiveness of the system to emergency situations. At the LPL, a single-agent planner is used to determine a sequence of actions to achieve the goal. For example, the forklift robots in the loading dock application (see Section 5) use a library with domain plans. Multiagent planning situations at the CPL are described by an initial situation and by the goals of the agents involved in the planning process. Cooperative planning therefore involves agreeing on a joint plan that satisfies the goals of the agents ([13] describe such a mechanism for the loading-dock).

4.3 Execution

The execution of an action a by the PS_i process of a layer i is done by posting a commitment `request(commit(a))` down to the process PS_{i-1}. Commitments made by PS_C to PS_L are partial single-agent plans which are local projections of the joint plan negotiated among the agents. This partial plan is scheduled into the current local plan of the agent. Commitments made at the LPL, i.e., from PS_L to PS_B, are activations of procedure PoB determined to be executed. Finally, at the BBL, commitments result from the actual execution of procedures. Procedures describe sequences of activations of primitive actions (or the sending of messages) which are available in the agent's world interface. Procedures are processed by a stepwise execution mechanism. Each execution step is a commitment to the execution of a primitive action in the world interface.

5 Designing Multiagent Systems with INTERRAP

In this section, we present the FORKS application, a MAS developed according to the INTERRAP architecture. After describing the domain, the models for situation recognition and planning and scheduling defined above are instantiated by the example of recognizing and handling conflict situations.

5.1 The domain

The FORKS simulation system describes a MAS of interacting robots, automated forklifts that have to carry out transportation tasks in a loading dock. Figure 3 illustrates the structure of the loading dock. It is represented as a grid of size $m \times n$; each square $((i,j),t,r)$ can be of type $t \in \{ground, truck, shelf\}$ and can be within region $r \in \{parking\_zone,\ hallway,\ truck\_region,\ shelf\_region\}$. Squares of type $truck$ and $shelf$ can additionally contain at most one box.

Forklift agents occupy one square at a time; they have a range of perception (e.g.: one square in front), can communicate with other forklifts and perform actions $a \in$

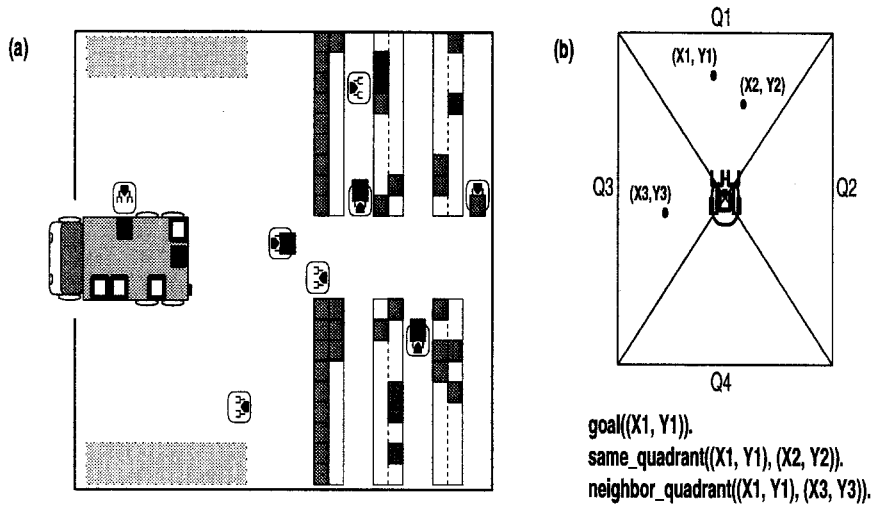

Fig. 3. (a) The loading dock (b) Quadrants

$\{moveto(dir), turnto(dir), grasp\_box, put\_box\}, dir \in \{n, e, s, w\}$. Agents receive orders to load or unload trucks; while performing their tasks, they may run into conflicts with other agents. E.g., agents may block each other, i.e., one agent may have the goal to move to a square occupied by another one, or two agents may try to move to one square by the same time.

5.2 Situation recognition and goal activation

The situation recognition capability of an agent is distributed over the three layers BBL, LPL, and CPL, allowing fast recognition of emergency situations, and a thorough classification of other situations, when more time is available.

An example for an emergency situation to be recognized in the SG_B module is a threatening collision. It can be modeled by a situation description sd_1:

$$sd_1 = \{location(self, (X_S, Y_S), O_S), status(self, moving), \\ perception(self, O_S, ((X, Y), T, R), \neg free((X, Y))\}$$

Note that sd_1 is defined merely by the external context, i.e., without taking into consideration knowledge about the agent's goals. A second type of conflict are blocking conflicts, which are defined by the fact that the agent is not moving, but intends to move to a square that is occupied by another agent. A situation description sd_2 for a mutual blocking conflict is:

$sd_2 =$

$\{location(self, (X_s, Y_s), O_s), location(A, ((X_a, Y_a), O_a)$ `/* external context */`

$opposed((X_s, Y_s, O_s), (X_a, Y_a, O_a))\}\cup$
$\{intends(self, goto\_landmark(X_a, Y_a))\}\cup$ /* mental context */
$\{bel(self, intends(A, goto\_landmark(X_s, Y_s))\}$ /* social context */

5.3 Planning and scheduling

Once recognized, there are several different possibilities to deal with a conflict situation. These possible reactions are implemented in the agents' PS processes. We draw a distinction between three basic classes of mechanisms which can be directly associated to the different INTERRAP control layers: behavior-based, local planning, and cooperative planning mechanisms.

Behavior-based mechanisms: This class of mechanisms has the Markov property: the decision of an agent at an instant t_i only depends on the state of the world at time t_{i-1}. One important class of decision functions having this property are *probabilistic decision functions* (PDFs). Let $\mathcal{A}$ be a non-empty set of actions, $\mathcal{G}$ a set of goals; let $f\colon \mathcal{S} \times \mathcal{A} \times \mathcal{G} \mapsto [0, 1]$ be a conditional probability distribution on $\mathcal{A}$ given $s \in \mathcal{S}, g \in \mathcal{G}$. Then a PDF is $\mathcal{F}^f(s, \mathcal{A}, g) = a_i$ with probability $f(s, a_i, g)$ for each $a_i \in \mathcal{A}$. We omit the superscript f for $\mathcal{F}$ in cases it is irrelevant.

An important class of PDFs are *uniform decision functions*, i.e., decision functions producing random behavior: A PDF $\mathcal{F}_r \equiv \mathcal{F}_r^f$ is an UDF iff $f(s, a, g) = \frac{1}{|\mathcal{A}|}$ for all $a \in \mathcal{A}$ and for all s, g.

```
proc PS_B
  i = 0;
  init([s_i , G_i ]);
  repeat
    i = i + 1;
    s_i = update_beliefs(s_{i-1}, Perc_i);   /* Perc_i = perception at time i */
    G_i = update_goals(G_{i-1}, s_i);        /* determine new goals */
    g = select_unsatisfied_goal(G_i);        /* select one goal */
    A = compute_alternatives(A, g, s_i);     /* compute alternatives */
                                               for the goal */
    next_action = F(s_i, A, g);              /* commit to next action */
                                               using decision function F */
    try_execute(next_action);
  forever
```

Fig. 4. The BBL control cycle

The behavior-based layer of INTERRAP is defined by a control cycle which, in each loop, computes a set of alternative PoBs (in the following simply called *alternatives* that might be pursued; it then decides which one actually to pursue by means of a PDF. This

cycle is illustrated in Figure 4. In the loading dock, given a situation s, the probability function f can be defined e.g., as:

$$f(s, a, grasp\_box(B)) = \begin{cases} 1 & : \quad a = grasp\_box(B) \\ 0 & : \quad otherwise \end{cases}$$

$$f(s, moveto(Dir), goto\_landmark(L)) = \begin{cases} 0.5 & : \quad same\_quadrant(Dir, L) \\ 0.2 & : \quad neighbor\_quadrant(Dir, L) \\ 0.1 & : \quad otherwise. \end{cases}$$

$Same\_quadrant$ and $neighbor\_quadrant$ are predicates relating different squares wrt. their relative location from the perspective of an agent (see Figure 3.b). Function f defines a variation of a potential field method where the agent is attracted by its goal region (in the example box B and landmark L), and prefers options that let it proceed towards its goal. In Section 5.4 we show how behavior-based agents can be modeled using PDF and UDF. For a more detailed analysis of decision functions for behavior-based reasoning, we refer to [11] in this volume.

Local planning mechanisms: This class of mechanisms uses a planning formalism in order to determine the next action to be performed, taking into consideration the agent's current goals. For task planning, a hierarchical skeletal planner has been implemented in the FORKS system (see [12]). It decomposes goals into subgoals, until an executable procedure PoB is reached; in this case a commitment is posted to the BBL. In FORKS, a path planner $\mathcal{P}$ is used on a graph representation of the loading dock to determine the shortest paths between a given square and the goal square. If e.g., a blocking conflict is detected, $\mathcal{P}$ is run again to determine a new path to the agent's goal.

Cooperative mechanisms: Local planning mechanisms run into trouble in two cases: Firstly, if the number of agents increases, blocking conflicts occur very often (see Section 6); thus, the effort of replanning becomes too big. Secondly, given incomplete information, certain goal conflicts cannot be resolved by mere local replanning. Therefore, the PS_C process contains cooperative planning facilities. Joint plans for conflict resolution are negotiated among the agents and executed in a synchronized fashion (see Section 4 and [13]).

5.4 Agent design

The different mechanisms described in the above subsections can be combined by the system designer to build a variety of agents having different types and different properties. Thus, controlled experimentation is supported aimed at investigating how the design of individual agents determines the behavior of the MAS. In the sequel, five exemplary agent types for the loading dock application are defined; they are analyzed empirically in Section 6.

The random walker (RWK): RWK is an agent that chooses its actions randomly; i.e., it always uses an UDF $\mathcal{F}_r$. In the case of RWK, conflict resolution is done implicitly: if the agent selects an alternative that cannot be carried out, execution will fail and the agent will continue selecting alternatives randomly until it has found a solution (if one exists).

Behavior-based agent with random conflict resolution (BCR): BCR performs task planning using a PDF $\mathcal{F}_p$ as defined above. To resolve blocking conflicts, it shifts to random mode (using function $\mathcal{F}_r$) for n steps; after this, it uses function $\mathcal{F}_p$, again. The advantage of randomness is that it allows to get out of local optima; in practice, this has turned out useful to avoid livelocks.

Behavior-based agent with heuristic conflict resolution (BCH): Similar to BCR, BCH uses decision function $\mathcal{F}_p$ for task planning; however, to resolve blocking conflicts, it employs a different strategy: if possible, it tries to dodge the other agent instead of just moving randomly. Especially conflicts in the hallway region can be resolved efficiently by this strategy.

Local planner with heuristic conflict resolution (LCH): LCH uses the hierarchical skeletal planner described in [12] for local task planning; it employs the same heuristic conflict resolution strategy as BCH.

Local planner with cooperative conflict resolution (LCC): This agent type has the same local planning behavior as LCH; however, for resolving conflicts, it combines local heuristics (for conflicts in hallway and truck regions) with coordination via joint plans (for conflicts in shelf regions).

6 Experimental Results

In this section, the results of a series of experiments carried through for the loading dock application are reported. The goal of these experiments was to evaluate the behavior of different types of INTERRAP agents and how they depend on different internal and environmental parameters.

6.1 Description of the experiments

The test series reported in this paper contains tests with homogeneous agent societies. We ran experiments with four, eight, and twelve forklift agents. These agents had to carry out randomly generated tasks in a loading dock of size 15×20 squares, with six shelves and one truck. The topology of the loading dock (see Figure 3.a) ensures that any square of type *ground* is reachable from any other. The number of tasks were 50 for four agents, 100 for eight agents, and 150 in the twelve-agent case. Each experiment was repeated five times (for twelve agents) and ten times, respectively (for eight and four agents) with the five agent types RWK, BCR, BCH, LCH, and LCC. The focus of the experiment was to evaluate the system behavior wrt. the following questions: (i) Is one of the described agent types or conflict resolution strategies dominant for the FORKS application? (ii) How gracefully degrade the different types and strategies when the number of agents is increased? How robust are they? (iii) How well do communication-based strategies compared to local ones?

6.2 Results

The main results of the experiments are illustrated by the diagrams 5.a - 5.d.

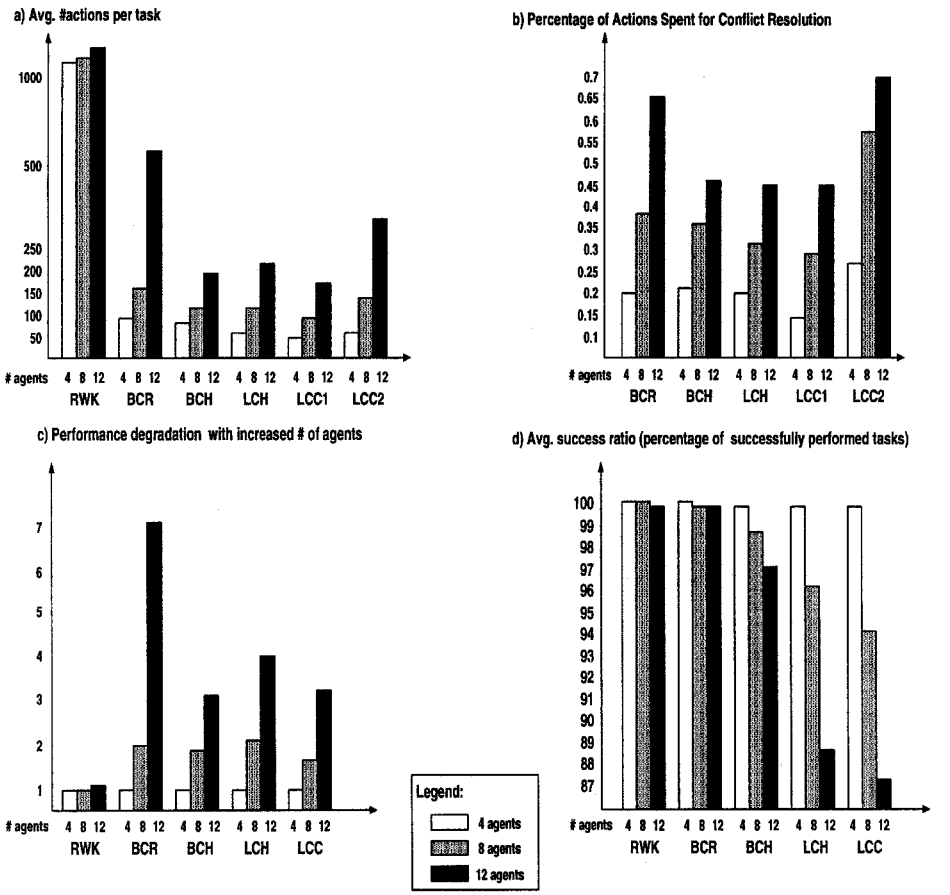

Fig. 5. Experimental results for the FORKS application

Absolute performance: Diagram 5.a shows the absolute performance for each agent type as the average number of actions needed per task. There are two entries for LCC: LCC1 only accounts for the number of physical actions (moves, turns, gripper actions), whereas LCC2 adds the number of messages sent (one message $\cong$ one action). As expected, RWK performs worst in all experiments. The best strategy is LCC; thus, resolving conflicts by communication pays off as regards the absolute performance. However, as LCC2 shows, the exact value depends on the cost of communication.

Conflict efficiency: Diagram 5.b displays the the ratio of actions needed for conflict resolution to the total number of actions. Since RWK does not explicitly recognize conflicts, it is not included in this statistics. The main result to be noted here is that LCC performs very well for smaller agent societies; for larger ones, it actually does not lead to a considerably higher conflict resolution efficiency, in comparison with local methods.

Degradation: The factor of performance degradation δ shown in Figure 5.c for x agents, $x \in \{4, 8, 12\}$ is computed as $\delta(x) \stackrel{\text{def}}{=} \frac{\#a(x) \cdot \#t(4)}{\#a(4) \cdot \#t(x)} \cdot \frac{1}{\rho}$, where ρ is the success ratio (see below), $\#a(x)$ denotes the total number of actions, and $\#t(x)$ denotes the total number of tasks in the x-agent experiment.

The performance of agent type RWK happens to be very insensitive to the size of the agent society, whereas the performance of all other agent types degrades considerably with a growing number of agents. A second interesting observation is that the agents employing simpler types of interaction show a more graceful degradation of performance than the more complex ones, especially the one based on communication. This is mainly due to the fact that the effort for communication and replanning outweighs the benefits of more elaborate strategies if the environment changes very rapidly.

Robustness: Robustness is measured by the success ratio ρ, which is the ratio of successfully finished tasks to the total number of tasks given to the agent. In our experiments, there are three sources of failures. Failures due to local maxima, deadlock situations caused by conflicts, and failures due to multiple conflicts that could not be adequately recognized and handled by the agents. The main result concerning robustness is that behavior-based strategies tend to be more robust than plan-based, cooperative strategies.

7 Related Work

Since the beginning of this decade, a considerable research effort has been devoted to the development of architectures for autonomous agents in dynamic environments. Only a few of these approaches can be discussed here; in particular, we focus on the relationship between our work and that of others contained in this volume. For an survey of developments in agent design, we refer to [19]. Layered approaches date back at least to Brooks' hardwired subsumption architecture [4]; the approach has been extended and refined by many other researchers, e.g., by adding planning capabilities to a reactive basis layer (e.g., [8], [7]). A recent approach incorporating the idea of layering to reconcile reaction and deliberation is 3T (see [2] in this book). The three layers in 3T are *reactive skills*, *sequencing*, and *deliberation*. In INTERRAP, the skill layer is implemented by reactor PoBs; sequencing tasks are running in all layers of INTERRAP, but particularly correspond to the handling of procedures in the behavior-based layer. Deliberation in INTERRAP is split up into local and cooperative deliberation; thus, while the focus in 3T still is on the case where we have a single agent acting in a dynamic world. INTERRAP extends the classical reactor-planner view by the cooperative layer to agents living in multiagent worlds, where dynamics is caused by the presence of other agents.

Another interesting approach to be read in this book is SIM_AGENT by Sloman and Poli [18]. The architectural concept is similar to 3T and to INTERRAP. An agent is a layered entity consisting of (1) a level of automatic processes, (2) a level of resource-bounded deliberation, and (3) a meta-level. Again, the main difference to INTERRAP is that cooperation is not modeled as a generic capability in the SIM_AGENT architecture. On the other hand, SIM_AGENT makes a conceptual distinction between meta- and

object-level process management, and it extends the current scope of INTERRAP by providing various toolkit functionalities.

Various other contributions inside this volume deal with *multiagent architectures* and testbeds (see e.g., [1], [14], [10]). The difference between this work and ours is that our focus is on the design of the individual agent and defines cooperation mechanisms viewed from this agent-centered perspective, whereas multiagent architectures rather provide tools for designing multiagent systems.

Finally, we should comment on the BDI aspect of our work. Looking at Rao and Georgeff's work, there is a development from a theoretical model [15] over an abstract agent interpreter [16] to a complex agent programming system supporting the development of real-world applications [17]. INTERRAP uses notion such as beliefs, goals, and intentions as useful abstractions of the mental state of an agent, but focuses on architectural issues, such as how reactivity and deliberation can be integrated within a layered BDI framework, and how cooperation mechanisms can be integrated, i.e., issues that have not yet been considered in depth by the researchers who have developed the BDI paradigm.

8 Discussion

In this paper, we identified two basic functions explaining the transformation from what an agent perceives to what it does: situation recognition and goal activation, and planning and scheduling. The individual control layers of the INTERRAP architecture were redefined according to a new uniform structure based upon these functions. The implementation of these concepts was shown by the example of an interacting robots application; empirical results were presented showing how different options to design agents according to the INTERRAP model affect the behavior of the system as a whole. The main contribution of the paper has been to provide a uniform control model allowing to express reactivity, deliberation, and cooperation by defining different instantiations of three general functions. The work reported in this paper has provided a basis for the reimplementation of INTERRAP using the Oz programming language developed at the DFKI. The FORKS system has been implemented both as a computer simulation and on KHEPERA miniature robots.

This paper has necessarily focussed on aspects of the individual agent. Aspects of cooperation and interaction have been treated very briefly. Future work will explore more complex planning and cooperation mechanisms and lead to a richer model of the LPL and the CPL than the one described in this paper.

References

1. M. Barbuceanu and M. S. Fox. The architecture of an agent building shell. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, LNAI series. Springer-Verlag, 1996. (In this volume).
2. R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors,

Intelligent Agents — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95), LNAI series. Springer-Verlag, 1996. (In this volume).

3. M. E. Bratman, D. J. Israel, and M. E. Pollack. Toward an architecture for resource-bounded agents. Technical Report CSLI-87-104, Center for the Study of Language and Information, SRI and Stanford University, August 1987.
4. Rodney A. Brooks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, volume RA-2 (1), pages 14–23, April 1986.
5. V. G. Dabija. *Deciding Whether to Plan to React.* PhD thesis, Stanford University, Department of Computer Science, December 1993.
6. E. H. Durfee and J. Rosenschein. Distributed problem solving and multiagent systems: Comparisons and examples. In M. Klein, editor, *Proceedings of the 13th International Workshop on DAI*, pages 94–104, Lake Quinalt, WA, 1994.
7. I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents.* PhD thesis, Computer Laboratory, University of Cambridge, UK,, 1992.
8. R. James Firby. *Adaptive Execution in Dynamic Domains.* PhD thesis, Yale University, Computer Science Department, 1989. Also published as Technical Report YALEU/CSD/RR#672.
9. K. Fischer, J. P. Müller, and M. Pischel. Unifying control in a layered agent architecture. Technical Memo TM-94-05, DFKI GmbH, Saarbrücken, January 1995.
10. C. A. Iglesias, J. C. González, and J. R. Velasco. MIX: A general purpose multiagent architecture. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, LNAI series. Springer-Verlag, 1996. (In this volume).
11. J. P. Müller. A markovian model for interaction among behavior-based agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, LNAI series. Springer-Verlag, 1996. (In this volume).
12. J. P. Müller and M. Pischel. An architecture for dynamically interacting agents. *International Journal of Intelligent and Cooperative Information Systems (IJICIS)*, 3(1):25–45, 1994.
13. J. P. Müller and M. Pischel. Integrating agent interaction into a planner-reactor architecture. In M. Klein, editor, *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence*, Seattle, WA, USA, July 1994.
14. S.-J. Pelletier and J.-F. Arcand. Cognitive based multiagent architecture. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, LNAI series. Springer-Verlag, 1996. (In this volume).
15. 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)*, pages 473–484, Cambridge, Mass., April 1991. Morgan Kaufmann.
16. A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In *Proc. of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 439–449. Morgan Kaufmann, October 1992.
17. A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.
18. A. Sloman and R. Poli. SIM_AGENT: A toolkit for exploring agent designs. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, LNAI series. Springer-Verlag, 1996. (In this volume).
19. M. J. Wooldridge and N. R. Jennings, editors. *Intelligent Agents – Theories, Architectures, and Languages*, volume 890 of *LNAI* series. Springer-Verlag, 1995.