UNIFYING CONTROL IN A LAYERED AGENT ARCHITECTURE*

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

This paper is also available as Technical Report TM-94-05 from the DFKI GmbH

Abstract

In this paper, we set up a unifying perspective of the individual control layers of the architecture INTERRAP for autonomous interacting agents. INTERRAP is a pragmatic approach to designing complex dynamic agent societies, e.g. for robotics [Müller & Pischel 94a] and cooperative scheduling applications [Fischer et al. 94]. It is based on three general functions describing how the actions an agent commits to are derived from its perception and from its mental model: *belief revision and abstraction*, *situation recognition and goal activation*, and *planning and scheduling*.

It is argued that each INTERRAP control layer — the behaviour-based layer, the local planning layer, and the cooperative planning layer — can be described by a combination of different instantiations of these control functions. The basic structure of a control layer is defined. The individual functions and their implementation in the different layers are outlined.

We demonstrate various options for the design of interacting agents within this framework by means of an interacting robots application. The performance of different agent types in a multiagent environment is empirically evaluated by a series of experiments.

*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

1 INTRODUCTION

The design of intelligent agents is an important research direction within multiagent systems (MAS) [Bond & Gasser 88, Durfee & Rosenschein 94], where the behaviour of a society of agents is described by modelling 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 major reasons for dealing with agent architectures: One is to explain and to predict agent behaviour; this means to describe how the decisions made by an agent are derived from its internal (mental) state and how this mental state is affected by the agent's perception. The second reason which goes beyond the first one 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[1]-style architectures [Bratman et al. 87, Rao & Georgeff 91], describing the internal state of an agent by several mental attitudes, namely *beliefs*, *goals*, and *intentions*. BDI theories provide a clear conceptual model representing the knowledge, the goals, and the commitments of an agent. However, they offer only little guidance to determine how the agent actually makes decisions based on its mental state, and have to be extended to actually support the design of resource-bounded and goal-directed agents for practical applications.

In [Rao & Georgeff 92], Rao and Georgeff have provided an abstract agent interpreter operationalising the BDI theory by describing an agent by one processing cycle. This cycle consists of the basic phases of option generation, deliberation, state update, execution, and update of the event queue. The system's reaction time is bounded from below by the time taken to perform a cycle. Moreover, since the individual processes within the cycle are monolithic, the architecture itself does not optimally support reactivity in a sense that it does not provide mechanisms e.g. allowing to recognise emergency situations in time. Rather, mechanisms for doing that (for example priority-based situation checking) have to be defined within the individual functions.

One way to overcome this problem is the use of layered agent architectures, that have become an important direction in intelligent agent design over the past few years (see e.g. [Brooks 86, Kaelbling 90, Ferguson 92, Firby 92, Lyons & Hendriks 92, Dabija 93, Steiner et al. 93, Müller & Pischel 94a, Müller & Pischel 94c]). Layering is a powerful concept for the design of resource-bounded agents. It combines a modular structure with a clear control methodology, and supports a natural modelling 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

[1]BDI ≡ Belief, Desire, Intention

[Wooldridge & Jennings 95].

The agent architecture INTERRAP which is described in this paper aims at combining the advantages of BDI-style architectures with those of layered ones. By this combination, our goal is to provide an architecture that serves both to explain agent behaviour and to support system design. INTERRAP adopts the mental categories used in BDI theory to describe an agent's knowledge, its goals, and its state of processing. It extends the work of [Rao & Georgeff 91, Rao & Georgeff 92] by organising an agent's state and control within a layered architecture. The problem-solving capabilities of an agent are described hierarchically by a behaviour-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. The need to develop an architecture which is suitable to build real applications has enforced a more pragmatic perspective[2].

Previous work [Müller & Pischel 94a, Müller & Pischel 94b] 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 analyse: the main part of the paper deals with the definition of a uniform structure for the different control layers. This uniformity is based on certain similarities of the processes running at the different layers: on the one hand, local planning and cooperative planning certainly utilise different levels of knowledge, but require rather similar techniques and algorithms; on the other hand, reactivity and deliberation are rather two extremes in a broad spectrum of agent behaviours than two really different paradigms.

Section 2 provides an overview of the architecture; the new uniform structure of an INTERRAP control layer is presented. Issues of knowledge representation and belief revision are discussed in Section 5. Section 6 describes a model for situation recognition and goal activation. The implementation of planning and scheduling in INTERRAP is outlined in Section 7. Section 8 provides an example for how INTERRAP is used to design an application system. The performance of different agent types is analysed in Section 9.

2 THE INTERRAP AGENT ARCHITECTURE

INTERRAP is an approach to modelling resource-bounded, interacting agents by combining reactivity with goal-directed and cooperative behaviour. In this Section, we present the basic concepts of the architecture.

[2]The abstract agent interpreter defined in [Rao & Georgeff 92] also uses BDI-theory in a conceptual sense.

2.1 Overview

Figure 1 illustrates the overall structure of the architecture. InteRRaP describes an agent as consisting of a world interface, a control unit, and a knowledge base (KB). The control unit consists of three layers: the behaviour-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

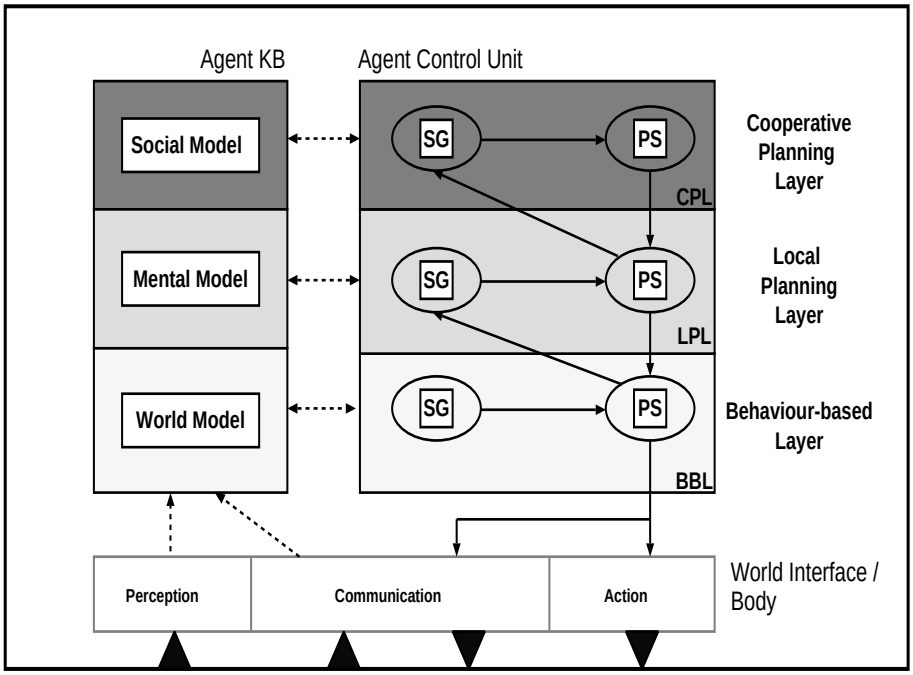

Figure 1: The InteRRaP Agent Architecture

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 behaviour* (PoB)), and to deal with routine situations (using *procedure PoB*). Reactors are triggered by events recognised 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 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 and/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 will be explained in more detail in the following

sections of this paper.

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

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

Table 1 illustrates how the functions defined above are distributed over the individual modules. In the following sections, the implementation of the functions is presented

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

in more detail.

2.2 The Control Layers

Viewed from a certain level of abstraction, 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 $i \in \{B, L, C\}$[3] consists of two processes implementing the functions SG and PS; these interact with each other and with processes from neighbour layers:

[3]Throughout this paper, we use the subscripts B for BBL, L for LPL, and C for CPL.

- The **situation recognition and goal activation** process SG_i recognises 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, and selects goals to be pursued as new intentions, taking into account the current intention structure. Moreover, it does the actual planning, i.e. the computation necessary to achieve these goals.

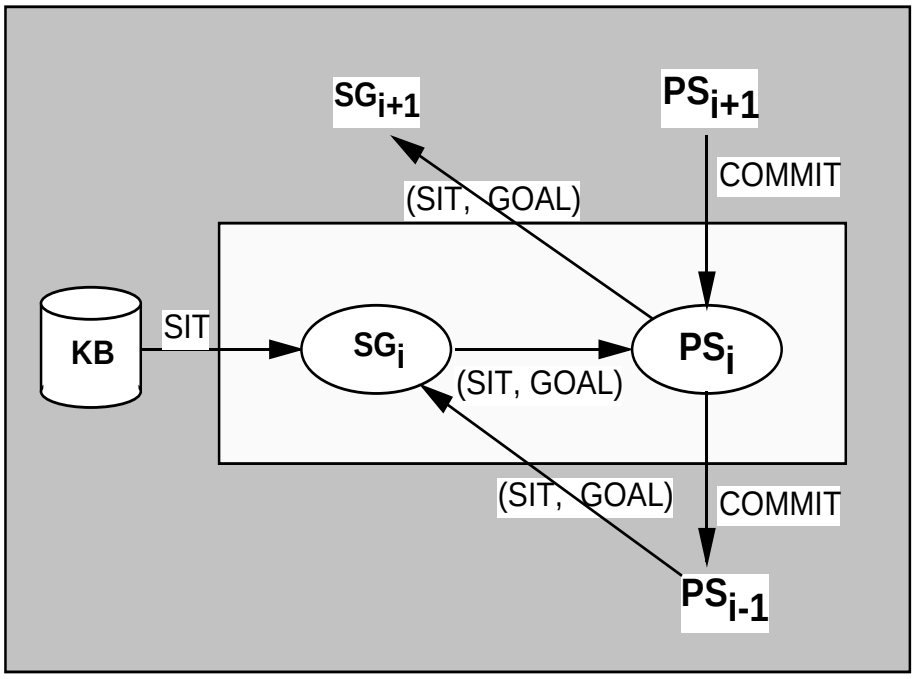

Figure 2: Structure of an InteRRaP Control Layer

The implementation of the two functions in InteRRaP is explained in more detail in Sections 6 and 7.

2.3 The Flow of Control

The control flow and thus the behaviour 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[4].

[4]Further more specialised protocols cannot be discussed here due to space restrictions.

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. It has been extended to allow more flexible interaction between the local and cooperative planning layers.

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 behaviours. These requests are regarded as commitments made by the PS_L component as a consequence of the intentions derived in this process.

Based on these protocols, the possible problem-solving behaviour of an INTERRAP agent can be classified by three generic *control paths*: the *reactive path*, the *local planning path*, and the *cooperative planning path*. Following the reactive path, a class of emergency situations is recognised in SG_B and directly dealt with using reactor patterns (example: stop to avoid a collision). In the local planning path, the LPL is activated to deal with more complex situations (example: planning a transportation order), a plan is devised and executed by activating procedure patterns. Finally, the cooperative planning path is triggered by the CPL; it involves communication and coordination among agents (example: negotiate a joint plan for resolving a blocking conflict).

3 THE LOADING DOCK APPLICATION

In this Section, we present the FORKS application, a MAS developed according to the INTERRAP architecture. The FORKS system describes a MAS for an interacting robots application, i.e. automated forklifts that have to carry out transportation tasks in a loading dock. The implementation of FORKS as a computer simulation running on UNIX workstations is based on the multiagent development platform AGENDA [Fischer et al. 95]; in order to evaluate the concepts in a real robot scenario, the FORKS+ system has been designed and implemented; it constitutes an implementation of FORKS using real KHEPERA miniature robots [Mondada et al. 93].

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.

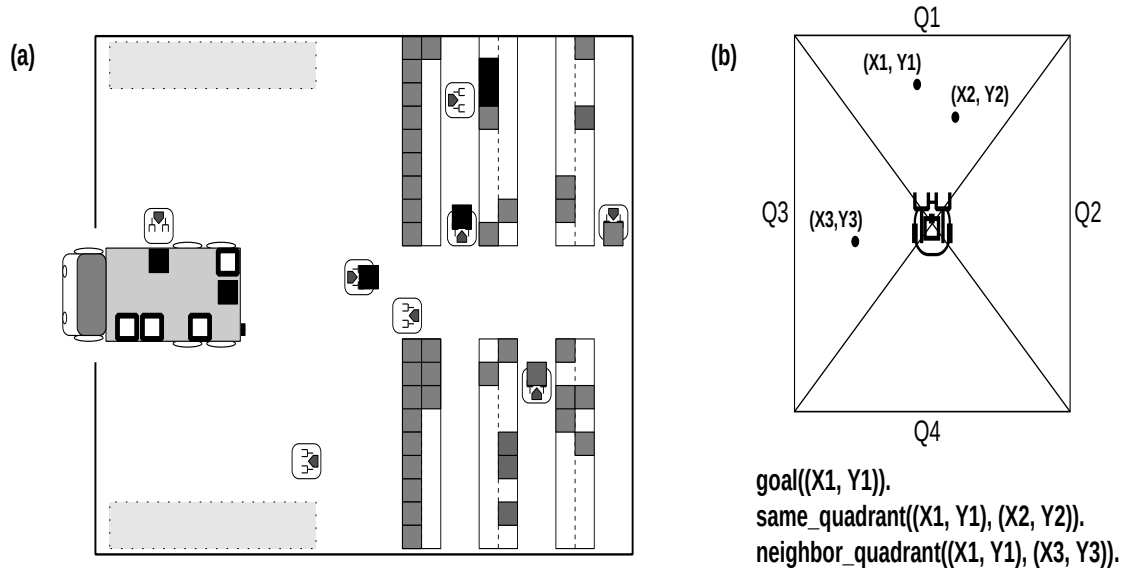

Figure 3: (a) The Loading Dock (b) Quadrants

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

$$a \in \mathcal{A} \equiv \{moveto(Dir), turnto(Dir), grasp\_box, put\_box\}, Dir \in \{n, e, s, w\}.$$

Performing actions changes the state of the world:

- *moveto(Dir)* moves the agent to the next square in the direction denoted by *Dir*. The action fails if the square in front of the agent is occupied. In the FORKS+ system, the implementation of *moveto* is rather complex: an agent may recognise that another agent is approaching only after it has already started to perform the action. For this purpose, in FORKS+ *moveto* provides means allowing the robot to move back by turning around to the previous location and reporting failure of the action in order to guarantee its logical atomicity.

  In order to simplify the problem of computing the current position while driving, the robot orients itself by following induction lines using infrared floor sensors and a simple control algorithm. For a more detailed discussion of aspects of behaviour-based control in the FORKS+ system, we refer to [Müller et al. 95].

- *turnto(Dir)* has the agent turn around to a direction specified by *Dir*; this action is needed e.g. to turn to a shelf in order to search through it, when the agent is located at a square neighbouring the shelf. *Turnto* always succeeds;

however, even this action is non-trivial in the real robot application, since it is prone to the accumulation of deviations in direction; thus, from time to time, the robot has to calibrate itself in order to avoid losing orientation.

- *grasp_box* is an action which succeeds if the agent is not holding a box, and stands in front of a field of type $t \in \{truck, shelf\}$ which is occupied by a box. In this case, result of the action is that the box is no longer on the shelf/box, but is held by the agent. In all other cases, the action fails. In FORKS+, the robot is able to additionally check whether it is really holding the box after having performed the *grasp_box* command by means of a light barrier that is integrated into the gripper.

- *put_box* is the inverse action to *grasp_box*.

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.

4 KNOWLEDGE REPRESENTATION

In this section, we will outline the basic knowledge representation (KR) mechanism for InteRRaP agents which is provided by the AgendA development environment for multiagent system applications [Fischer et al. 95]; the system development layer of AgendA defines a set of basic reasoning mechanisms and the knowledge representation model AKB[5]. Most parts of this Section are adopted from [Weiser 95].

4.1 AKB Representation Schema

The basic elements of the knowledge representation schema are the following:

- Concepts C, C_1, C_2, ...
- Types $T, T_1, T_2, \ldots$
- Attributes $A : C \mapsto T$
- Features $F : C \mapsto T$
- Relations $R \subseteq C_1 \times C_2 \ldots \times C_n$

Attributes A may have default values $default(A) = k$; features are attributes of a concept that cannot be changed; $init(F) = k$ denotes the initial value of a feature. Apart from standard types such as *integer*, *string*, *real*, ..., new types can be defined by Oz [Henz et al. 93] procedures.

An AKB-schema declaration thus looks as follows:

[5] Assertional Knowledge Base

```
[ ...
 concept(   name:    ConceptName )
 relation(  name:    RelationName
            domain:  ConceptName_1 # ... # Conceptname_n )
 attribute( name:    AttributeName
            concept: ConceptName
            type:    Type )
 default(   name:    AttributeName
            value:   DefaultValue )
 feature(   name:    FeatureName
            concept: ConceptName
            type:    Type
            init:    Init )
 ...
]
```

4.2 AKB Interface Specification

The first class of functions which are offered by AKB are assertional functions which are needed to modify the agent's knowledge base (KB). In the following, X, Y denote input variables, $?X, ?Y$ denote output variables, i.e. values returned by the function call:

Asserting Beliefs

- *createObject(?Id)*: returns a unique identification of a newly created object. AKB is object-oriented in a sense that concept instances are represented as Oz objects.
- *enterConcept(Id Concept)*: creates an instance of a concept denoted by *Concept* and binds it to the object identified by *Id*.
- *enterRelation(IdList Rel)*: Defines an instance of a new relation denoted by relation *Rel* among the concept instances denoted by the object identifiers in *IdList*. The ordering of the members of *IdList* is meaningful; it corresponds to their ordering in the relation.
- *setValue(Id Attr NewVal)*, *setValue(Id Attr NewVal ?OldValue)*: assigns the value denoted by *NewVal* to the attribute *Attr* of the concept instance denoted by *Id*. *SetValue* with four arguments additionally returns the old attribute value.

Retracting Beliefs

- *deleteObject(Id)*: delete an object that has been created before. Deleting an object that is bound to a concept instance deletes the concept instance and all instances of relations where this concept instance is a member.
- *deleteConcept(Id Concept)*: deletes the instance of *Concept* denoted by *Id*.
- *deleteRelation(IdList Rel)*: deletes the instance of relation *Rel* denoted by *IdList*.
- *retractValue(Id Attr)*: removes the value for the attribute *Attr* of the concept instance denoted by *Id*.

Information Retrieval

The second important class of interface functions are *retrieval functions*. They are provide an access to the knowledge actually stored in the knowledge base. AKB offers the following retrieval functions:

- *getConceptMembers(Concept ?IdList)*: Returns a list of all instances of *Concept*.
- *isConceptMemberP(Id Concept ?Bool)*: Returns true if the concept instance denoted by *Id* is a member of *Concept*.
- *getRelationMembers(Rel ?ListOfIdLists)*: Returns a list of list of concept instances denoting all tuples that define relation *Rel*.
- *isRelationMemberP(IdList Rel ?Bool)*: *?Bool* returns true if the tuple denoted by *IdList* is a member of the relation *Rel*.
- *getRelationFiller(Rel k* $IdList_1$, $?IdList_2$*)*: for an n-ary relation *Rel*, for $1 \leq k \leq n$, and for a list $IdList_1 = \{o_1, \ldots, o_{n-1}\}$ of concept instances with $|IdList_1| = n - 1$, *getRelationFiller* instantiates $IdList_2$ to

$$IdList_2 = \{o | (o_1, \ldots, o_{k-1}, o, o_k, \ldots, o_{n-1}) \in Rel\}.$$

- *getValue(Id AttrOrFeat ?Val)*: returns the value of an attribute or of a feature of the concept instance denoted by *Id*.

4.3 Planned Extensions

AKB as presented in this Section provides a general and simple knowledge representation formalism; future work will extend AKB in different directions.

- Adding a deduction rule mechanism which allows e.g. to express background theories and integrity constraints

- Extending the KB specification to a full object-oriented knowledge base providing inheritance, specialisation and generalisation (is_a relation).
- Defining a transaction concept for AKB which allows the atomic execution of a sequence of operations. This is especially important to synchronise the knowledge base access by different control layers of the INTERRAP architecture.

5 BELIEF REVISION AND KNOWLEDGE ABSTRACTION

This section describes a simple mechanism how perception can be transformed into belief; due to space limitations, we will not discuss belief abstraction, i.e. the derivation of more abstract or complex beliefs from simpler ones; for the belief revision process, we will focus on the world model part of the agent knowledge base, since this is most closely related to perception; mechanisms for revising the agent's mental and social model are beyond the scope of this paper.

In this paper, we assume that an agent perceives symbolic information; i.e. its perception is specified in the same language as its beliefs. At this point, the processing needed to obtain this level of representation is not considered. Furthermore, note that the agent's world model and its perception are represented as ground atomic first-order formulae. Thus, the problem is reduced to maintaining consistency of the world model, i.e. of the object-level beliefs an agent has about its environment. We assume that perception is time-stamped and that $time(p)$ denotes the time stamp of a proposition p.

In general, we distinguish between two kinds of consistency, namely *logical consistency* and *semantic consistency*:

Logical Consistency: We adopt an incomplete notion of logical consistency for ground atomic formulae: for a proposition p and a set of atomic propositions Σ we define $WLC(p, \Sigma)$ iff $not(\neg p \in \Sigma)$ (WLC = weakly logical consistent).

Semantic Consistency: A simple notion of semantic consistency is defined by describing a finite set of domain-specific axioms specifying that in a certain domain two facts in the world model are semantically inconsistent. A consistency axiom is of the form

$$SI(p_1, p_2) \leftarrow cond_1 \wedge cond_2 \wedge \ldots \wedge cond_k$$

where p_1 and p_2 are atomic first-order formulae. The conditions $\texttt{cond}_i$, $1 \leq i \leq k$ are inductively defined by first-order atomic formulae connected by the junctors $\neg$ and $\vee$. However, we require the variables V used in $\texttt{cond}_i$ to be a subset of the variables used in $\texttt{p}_1$ and $\texttt{p}_2$: $V \subseteq V_{p_1} \cup V_{p_2}$. We do not allow recursion within cond_i. Furthermore, we require that SI is only instantiated with ground atomic

formulae, i.e. with formulae q_1, q_2 with $\exists\theta_1,\theta_2.\theta_1 p_1 \equiv q_1 \wedge \theta_2 p_2 \equiv q_2$ for ground matchings θ_1, θ_2. These restrictions allow us to interpret the conditions $\texttt{cond}_i$ over the Herbrand Universe.

The intuitive semantics of SI is that $SI(p_1, p_2)$ is true if believing p_1 is not semantically consistent with believing p_2. For example, a consistency axiom denoting that it is not consistent to believe another robot to have two different locations at the same time is:

$$SI(location(A,(X_1,Y_1),O_1), location(B,(X_2,Y_2),O_2)) \leftarrow$$
$$A = B \wedge (X_1 \neq X_2 \vee Y_1 \neq Y_2 \vee O_1 \neq O_2) \wedge$$
$$time(location(A,(X_1,Y_1),O_1)) = time(location(B,(X_2,Y_2),O_2)$$

Based on the set of consistency axioms with the above properties, a decision method for the predicate SI can be defined. SI terminates since we do not allow recursion and instantiation of the axiom with formulae containing variables. This is important for the proof of proposition 1 (see below).

A Belief Revision Algorithm: Let agent a's world model at time t be $WM_t = \{q_1, \ldots, q_n\}$. Let $P_{t+1} = \{p_1, \ldots, p_k\}$ be the set of formulae perceived by a at time $t+1$. Let WLC and SI be meta-predicates for checking weak logical and semantical consistency as defined above. Then, a's new beliefs WM_{t+1} are computed by the following function:

func $BR(P_{t+1}, WM_t)$

$WM_{t+1} = WM_t$ /* initialise */
foreach $p \in P_{t+1}$ /* process each new perceived fact */
{
  if $\neg WLC(p, WM_{t+1})$ then /* logically inconsistent */
    $WM_{t+1} = WM_{t+1} \cup \{p\} - \{\neg p\}$
  else
  if $\exists q \in WM_{t+1}.SI(p,q)$ then /* semantically inconsistent */
    $WM_{t+1} = WM_{t+1} \cup \{p\} - \{q\}$
  else /* no inconsistency detected */
  $WM_{t+1} = WM_{t+1} \cup \{p\}$
}
return WM_{t+1}

The following properties of BR hold:

Proposition 1 *Let SI be a terminating decision predicate for semantic consistency as defined above. Then, function BR terminates for each finite input sets P and WM.*

Proof: Since P is finite, the *foreach* loop is performed only finitely often. To show the termination of the body of the loop, we have to show that the predicates $WLC(p_i, WM_{new})$ and $SI(p_i, q_j)$ evaluate to true or false after a finite time. The termination of WLC is trivial since it only involves checking membership in set WM_{new} which is finite by our assumption. The termination of SI is true by our assumption. □

Proposition 2 *BR is correct in a sense that it returns a set WM' of new beliefs that are weakly logically consistent and semantically consistent provided that the input set WM of beliefs is weakly logically and semantically consistent.*

Proof: To show weak logical consistency, we have to prove that there is no proposition p such that $\{p, \neg p\} \subseteq WM_{new}$. By our assumption, the input set WM does not contain such formulae. Assume that $p \in WM$ and $\neg p \in P$. In this case, due to line 7, WM_{new} will contain $\neg p$. Analogously, for $\neg p \in WM$ and $p \in P$, WM_{new} will contain p. This allows us to conclude that WM_{new} is weakly logical consistent in the case that P is weakly logical consistent.

Assume now that there is a proposition q with $\{q, \neg q\} \subseteq P$. In this case, either q will be selected by the *foreach* branch before $\neg q$, or vice versa. In the former case, $\neg q$ will overwrite q, whereas q will overwrite $\neg q$, in the latter. Therefore, WM_{new} is weakly logical consistent even if P is not.

Semantic consistency is ensured by the application of the predicate SI in line 9 of the function. If a proposition $p \in P$ is semantically inconsistent with a formula $q \in WM$ with respect to the set of axioms $\mathcal{C}$, q will be replaced by p. Since WM is assumed to be semantically consistent, so is WM_{new}. Semantic inconsistencies within P are resolved as described in the case of logical inconsistencies, namely by simple overwriting within the *foreach* loop. □

Note that BR is incomplete because the definition of WLC does not include full logical deduction. For instance, if $\{p, q\} \subseteq P_{t+1}, q \not\equiv \neg p$ but $\{p\} \models \neg q$, this type of logical inconsistency cannot be recognised by BR. Inconsistencies in P_{t+1} itself are resolved by BR depending on the order in which the $p \in P_{t+1}$ are processed.

The reason for the simple knowledge representation and belief revision formalism defined at the world model layer is efficiency. The world model represents the dynamic environment of the agent; based on its world model, the agent has to recognise critical situations such as threatening collisions very quickly and has to react to it. Inconsistencies are resolved by the simple strategy of preferring beliefs based on more recent information to older ones.

6 SITUATION RECOGNITION

Situations are described from the perspective 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 $S_C \subseteq SM$. Thus, it describes a portion of the agent KB containing parts from its world model,

its mental model, and its 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. a set of goals and intentions; the social model part (*social context*) describes belief about other agents characterising a specific situation and parts of the agent's joint intention structure.

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 recognised by this layer. Let $\mathcal{T}$ denote a set of time points. The semantics of the function SG_i is defined by a function $OCC_i : 2^{\mathcal{L}} \times \mathcal{L} \times \mathcal{T} \mapsto 2^{\mathcal{L}}$. $OCC_i(\Sigma_i^t, D_i, t) = \mathcal{S}'$ returns the subset $\mathcal{S}'$ of instantiations of a situation description $D \in \mathcal{D}_i$ which occur at time t, i.e. which can be derived from the set of beliefs Σ_i^t at time t. At layer i, situations are mapped to goals $G \in \mathcal{G}_i$: $\beta_i : \mathcal{S}_i \mapsto \mathcal{G}_i$. $SG_i : 2^{\mathcal{L}} \times \mathcal{T} \times 2^{\mathcal{L}} \times 2^{\mathcal{L}} \mapsto 2^{2^{\mathcal{L}} \times 2^{\mathcal{L}}}$ is defined as

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

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

OCC_B is defined by $OCC_B(\Sigma_B^t, \mathcal{D}_B, t) = S$ iff $\exists d \in \mathcal{D}_B : S = d\theta$ 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 behaviour-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 recognise other agents' goals, either through communication or through explicit goal recognition techniques.

Situation recognition is an incremental process, i.e. partial situations may be recognised at lower layers and complemented at higher layers. The SG_i process outputs pairs (S, G). A goal G is associated to each situation S recognised by SG_i. This pair characterises a new option to be pursued by the agent. It serves as an input to the planning and scheduling process described in the sequel.

For a detailed example of the situation recognition process, we refer to Section 8.1 and to [Müller 94a].

7 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 element $G' \in \mathcal{G}_i$ for being pursued next and devise a partial plan P' for achieving 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.

7.1 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 upward 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 table lookup with matching; thus, it is possible to make decisions quickly at the reactive layer. However, trying to build 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 $u : PLANS \mapsto I\!R$ (see [Haddawy & Hanks 90, Müller 94b]). 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.

7.2 Deciding What to Do

After a layer has decided to be competent for a situation, the actual planning process starts resulting in a commitment, e.g. a decision to perform a certain action. Again, this planning process differs throughout the InteRRaP layers: At the BBL, patterns of behaviour provide direct hard-wired links from situations to compiled plans that are executed; 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 in order to achieve the goal. For example, the implementation of the forklift robots in the loading dock application (see Section 8) is based on 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 ([Müller & Pischel 94b] describe such a mechanism for the loading-dock).

7.3 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 planning and scheduling 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 intention structure (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 basically 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 [Müller et al. 95]. Each execution step is a commitment to the execution of a primitive action in the world interface.

8 EXAMPLE: DESIGNING MULTIAGENT SYSTEMS USING InteRRaP

In this Section, we describe how the FORKS application presented in Section 3 has been modelled using the theoretical framework presented so far. The models for situation recognition and planning and scheduling defined above are instantiated by the example of recognising and handling conflict situations.

8.1 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 recognised in the SG_B module is a threatening collision. It can be modelled by a situation description sd_1:

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

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 */`
$\quad opposed((X_s, Y_s, O_s), (X_a, Y_a, O_a))\} \cup$
$\{INTEND(self, goto\_landmark(X_a, Y_a))\} \cup$ `/* mental context */`
$\{BEL(a, INTEND(A, goto\_landmark(X_s, Y_s))\}$ `/* social context */`

8.2 Planning and Scheduling

Once recognised, 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: behaviour-based, local planning, and cooperative planning mechanisms.

Behaviour-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}. Let $\mathcal{A}$ be a set of alternatives, $\mathcal{G}$ be a set of goals, $g \in \mathcal{G}$. Let WM_i denote the agents world model at time i. A behaviour-based decision algorithm is defined as follows:

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

In the sequel, we define two classes of possible decision functions $\mathcal{F}$:

Definition 1 (Probabilistic Decision Function (PDF)) *Let $\mathcal{A}$ be a non-empty set of alternatives, $\mathcal{G}$ a set of goals; let $f : \mathcal{A} \times \mathcal{G} \mapsto [0,1]$ be a probability distribution on $\mathcal{A}$. Then a PDF is $\mathcal{F}_p^f(\mathcal{A}, g) = a_i$ with probability $f(a_i, g)$ for each $a_i \in \mathcal{A}$. We omit the superscript f for $\mathcal{F}$ in cases it is irrelevant.*

An important special case of PDF are *random decision functions*:

Definition 2 (Random Decision Function (RDF)) *A PDF $\mathcal{F}_r \equiv \mathcal{F}_r^f$ is an RDF iff $f(a, g) = \frac{1}{|\mathcal{A}|}$ for all $a \in \mathcal{A}$ and for all g in definition 1.*

The following proposition holds for the use of random decision functions in the loading dock domain defined in Section 3:

Proposition 3 *Let $\mathcal{F}_r$ be an RDF, let $\mathcal{A} \equiv$* {moveto(Dir), turnto(Dir), grasp_box, put_box} *be the set of alternatives as defined in Section 3 (non-deterministic case). Let L be a finite grid of size $n \times m$, let (X_i, Y_i) denote an arbitrary square in L. Then:*

1. *An agent using $\mathcal{F}_r$ as a decision function will reach each square (X, Y) that is reachable from (X_i, Y_i) infinitely often.*
2. *For each $(X, Y) \neq (X_i, Y_i)$, there is no finite upper bound on the maximal number of steps required to reach (X, Y) for the first time.*

Proof: **ad 1.** The first part of proposition 3 follows directly from the random walk theorem stated in [Chung 74].

ad 2. Let (X_i, Y_i), $1 \leq X_i \leq n$, $1 \leq Y_i \leq m$ be the initial position of the agent. Let $(X, Y) \neq (X_i, Y_i)$, $1 \leq X \leq n$, $1 \leq Y \leq m$ be an arbitrary square within grid L. Let $location(s)$ denote the access function to the agent's physical location (X_s, Y_s) in state s.

Assume that there ex. $n \in N$ which is an upper bound of steps required to reach (X_i, Y_i) from (X, Y). This means, for the length $|\alpha|$ of the biggest possible sequence of actions $\alpha = (a_1, a_2, \ldots)$, $a_i \in \mathcal{A}$ denoting a sequence of state transitions

$$s_0 \xrightarrow{a_1} s_1 \ldots \xrightarrow{a_n} s_n$$

with $location(s_0) = (X_i, Y_i)$, $location(s_n) = (X, Y)$, and $location(s_i) \neq (X, Y)$ for all $1 \leq i < n$, we have $|\alpha| \leq n$.

Now, we define a sequence β of actions $\{b_1, b_2, \ldots, b_m\}$, $m > n$ and $location(s_0) = (X_i, Y_i)$, $location(s_m) = (X, Y)$, and $location(s_i) \neq (X, Y)$ for all $1 \leq i < m$. We will show that β exists for the set $\mathcal{N} = \{(X_i, Y_i - 1), (X_i, Y_i + 1), (X_i - 1, Y_i), (X_i + 1, Y_i)\}$ of neighbour squares to (X_i, Y_i). This suffices to show that no finite lower bound exists for any other square (X', Y'), since $m' > m$ actions will be required to reach (X', Y'). We define $\beta = (turnto(n), turnto(south), \ldots, moveto(D))$, where the turning sequence is repeated $\lceil \frac{n}{2} \rceil$ times, and D denotes the direction corresponding to each $(\hat{X}, \hat{Y}) \in \mathcal{N}$. Obviously, $|\beta| \geq n + 1 > n$.

It remains to show that β is selected with a probability $p(\beta) > 0$. This holds true because $p(\beta) = (\frac{1}{|\mathcal{A}|})^{|\beta|} > 0$. From this, proposition 3 follows immediately. □

Note that proposition 3.1 does not hold for probabilistic decision functions in general since we do not require $f(a, \_) \neq 0$ for all $a \in \mathcal{A}$.

In the loading dock, the probability function f can be defined e.g. as:

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

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

Same_quadrant and *neighbor_quadrant* are predicates relating different squares with respect to their relative location from the perspective of an agent (see figure 3.b). Function f defines a slight 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 8.3 we show how behaviour-based agents can be modelled using PDF and RDF.

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 [Müller & Pischel 94a]). 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[6]. 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 9); 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 generated and negotiated among the agents (see Section 7 and [Müller 94b, Müller & Pischel 94b]).

8.3 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

[6]We use Dijkstra's algorithm with quadratic complexity.

how the design of individual agents determines the behaviour of the MAS. In the sequel, five exemplary agent types for the loading dock application are defined; they are analysed empirically in Section 9.

The random walker (RWK): RWK is an agent that chooses its actions randomly; i.e. it always uses the random decision function $\mathcal{F}_r$. 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).

Behaviour-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.

Behaviour-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 [Müller & Pischel 94a] 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 behaviour 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).

9 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 behaviour of different types of InteRRaP agents and how they depend on different internal and environmental parameters.

9.1 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 (for eight and four agents), respectively, with the five agent types RWK, BCR, BCH, LCH, and LCC. The focus of the experiment was to evaluate the system behaviour with respect to the following questions:

- Is one of the described agent types or conflict resolution strategies dominant for the FORKS application?
- How gracefully degrade the different types and strategies when the number of agents is increased? How robust are they?
- How well do communication-based strategies compared to local ones?

9.2 Results

The main results of the experiments are illustrated by the diagrams 4.a - 4.d.

Absolute performance: Diagram 4.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). RWK performs worst in all experiments. The plan-based types do somewhat better than the behaviour-based ones; especially LCC yields the best results in terms of actions; however, the value of explicit coordination depends on the cost of communication.

Conflict Efficiency: Diagram 4.b displays the the ratio of actions needed for conflict resolution to the total number of actions. Since RWK does not explicitly recognise conflicts, it is not included in this statistics. The main result to be noted here is that LCC performs well for small agent societies, whereas it actually does not increase conflict resolution efficiency for large agent societies, in comparison with local methods.

Degradation: The factor of performance degradation δ shown in figure 4.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

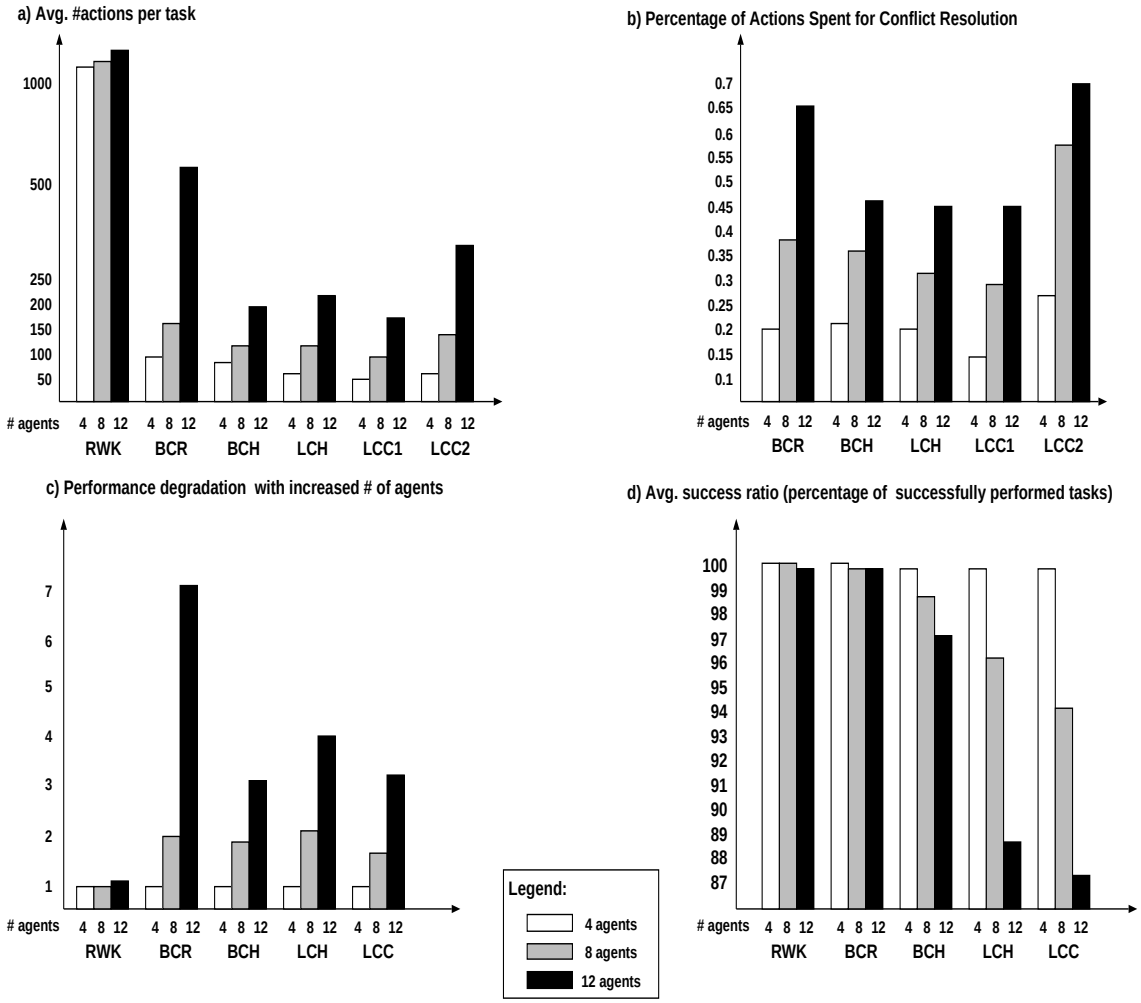

Figure 4: Experimental Results for the FORKS Application

that the behaviour-based agent types (except BCR[7]) tend to degrade more gracefully than the more complex ones (LCH and LCC).

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 recognised and handled by the agents. The main result concerning robustness is that behaviour-based strategies tend to be more robust than

[7]The poor performance of BCR in the twelve agent case is due to a cascade effect resulting from the fact that if there are many other agents around, while trying to resolve a conflict by performing n steps random walk, the agent is very likely to run into a new conflict aso.

plan-based, cooperative strategies. Randomness has been shown to be a powerful tool for avoiding and resolving deadlocks. Note that the robustness results are a little too optimistic, especially for LCC types, since the joint plan negotiation protocol used in the experiment cannot handle deadlocks caused by multiple conflicts; thus, if an agent runs into such a situation very early, it will is kept there for the rest of the experiment. Since tasks are allocated dynamically, other agents will perform its tasks; thus, the agent will report only one failed task. Currently, we are developing a negotiation protocol that can cope with multiple conflicts.

10 DISCUSSION

In this paper, we identified three basic functions explaining the transformation from what an agent perceives (its input) to what it does (its output): belief revision and abstraction, situation recognition and goal activation, and planning and scheduling. The individual control layers of the INTERRAP agent architecture were redefined according to a new uniform structure based upon these functions. 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 abstract architecture has provided a basis for the reimplementation of INTERRAP using the Oz programming language [Henz et al. 93]. The concepts have been evaluated by an interacting robots application, an automated loading dock [Müller & Pischel 94b] using KHEPERA miniature robots; empirical results were presented showing how different options to design agents according to the INTERRAP model affect the behaviour of the system these agents are in.

The focus of this paper has been on describing the structure of the individual layers rather than on describing how they interact. The problem of coherence in layered architectures, i.e. how the interaction between the different layers should be designed in order to achieve coherent behaviour of the agent, is beyond the scope of this paper. Some of its aspects have already been discussed in [Müller et al. 95]; it remains a subject for our future research.

ACKNOWLEDGEMENTS

Thomas Weiser has implemented the knowledge representation formalism. We thank Anand Rao and Michael Georgeff for helpful comments on BDI architectures. We are especially indebted to Michael Kolb and Donald Steiner for long and fertile discussions on the new structure of the agent architecture.

REFERENCES

[Bond & Gasser 88] A. **Bond** and L. **Gasser**. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, Los Angeles, CA, 1988.

[Bratman et al. 87] 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.

[Brooks 86] 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.

[Chung 74] K. L. **Chung**. *Elementary Probability Theory with Stochastic Processes.* Springer, New York, 1974.

[Dabija 93] V. G. **Dabija**. *Deciding Whether to Plan to React.* PhD thesis, Stanford University, Department of Computer Science, December 1993.

[Durfee & Rosenschein 94] E. H. **Durfee** and J. **Rosenschein**. *Distributed Problem Solving and Multiagent Systems: Comparisons and Examples.* In: M. Klein (ed.), Proc. of the 13th International Workshop on DAI, pp. 94–104, Lake Quinalt, WA, 1994.

[Ferguson 92] I. A. **Ferguson**. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents.* PhD thesis, Computer Laboratory, University of Cambridge, UK,, 1992.

[Firby 92] R. James **Firby**. *Building Symbolic Primitives with Continuous Control Routines.* In: J. Hendler (ed.), Proc. of the First International Conference on AI Planning Systems. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[Fischer et al. 94] 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.

[Fischer et al. 95] K. **Fischer**, J. P. **Müller**, and M. **Pischel**. *AGenDA: A General Testbed for DAI Applications.* In: N. R. Jennings and G. M. P. O'Hare (eds.), Foundations of DAI. John Wiley & Sons, Inc., 1995.

[Haddawy & Hanks 90] P. **Haddawy** and S. **Hanks**. *Issues in Decision-Theoretic Planning: Symbolic Goals and Numeric Utilities.* In: Proc. of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control. Morgan Kaufmann, November 1990.

[Henz et al. 93] M. **Henz**, G. **Smolka**, and J. **Würtz**. *Oz - A Programming Language for Multi-Agent Systems.* In: Proceedings of the IJCAI, Chambery, 1993.

[Kaelbling 90] L. P. **Kaelbling**. *An Architecture for Intelligent Reactive Systems.* In: J. Allen, J. Hendler, and A. Tate (eds.), Readings in Planning, pp. 713–728. Morgan Kaufmann, 1990.

[Lyons & Hendriks 92] 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), pp. 153–162, San Mateo, CA, June 1992. Morgan Kaufmann.

[Mondada et al. 93] F. **Mondada**, E. **Franzi**, and P. **Ienne**. *Mobile Robot Miniaturization: A Tool For Investigation in Control Algorithms.* In: Proc. of the Third Int. Symposium on Experimental Robotics, Kyoto, Japan, October 1993.

[Müller & Pischel 94a] 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.

[Müller & Pischel 94b] J. P. **Müller** and M. **Pischel**. *Integrating Agent Interaction into a Planner-Reactor Architecture.* In: M. Klein (ed.), Proc. of the 13th International Workshop on Distributed Artificial Intelligence, Seattle, WA, USA, July 1994.

[Müller & Pischel 94c] J. P. **Müller** and M. **Pischel**. *Modeling Interacting Agents in Dynamic Environments.* In: Proc. of the European Conference on Artificial Intelligence (ECAI94), pp. 709–713. John Wiley and Sons, August 1994.

[Müller et al. 95] J. P. **Müller**, M. **Pischel**, and M. **Thiel**. *Modeling Reactive Behaviour in Vertically Layered Agent Architectures.* In: M. J. Wooldridge and N. R. Jennings (eds.), Intelligent Agents — Theories, Architectures, and Languages, volume 890: Lecture Notes in AI. Springer, January 1995.

[Müller 94a] J. P. **Müller**. *A Conceptual Model of Agent Interaction.* In: S. M. Deen (ed.), Proc. of CKBS'94 (Selected Papers), pp. 213–233. University of Keele, UK, 1994.

[Müller 94b] J. P. **Müller**. *Evaluation of Plans for Multiple Agents (Preliminary Report).* In: K. Fischer and G. M. P. O'Hare (eds.), Proc. of the ECAI Workshop on Decision Theory for DAI Applications, Amsterdam, NL, August 1994.

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

[Rao & Georgeff 92] 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), pp. 439–449. Morgan Kaufmann, October 1992.

[Steiner et al. 93] D. D. **Steiner**, A. **Burt**, M. **Kolb**, and Ch. **Lerin**. *The Conceptual Framework of MAI^{2}L.* In: Pre-Proceedings of MAAMAW'93, Neuchâtel, Switzerland, August 1993.

[Weiser 95] T. **Weiser**. *AKB: Assertional Knowledge Base*, 1995. Internal Report.

[Wooldridge & Jennings 95] M. J. **Wooldridge** and N. R. **Jennings**. *Intelligent Agents: Theory and Practice*, 1995. Submitted to Knowledge Engineering Review.

Agent Type	RWK			BCR			BCH		
# Agents	4	8	12	4	8	12	4	8	12
NO	50	100	150	50	100	150	50	100	150
SR in %	100	100	100	100	100	100	100	98.6	97.0
NA	53544	107839	166458	3932	15290	84004	3065	11259	27613
QDF	1	1.01	1.04	1	1.94	7.12	1	1.86	3.1
APT	1070.88	1078.39	1109.72	78.63	152.9	560.03	61.3	114.19	189.78
APC	-	-	-	747	5657	54603	644	3941	12702
CRE in %	-	-	-	0.19	0.37	0.65	0.21	0.35	0.46

Agent Type	LCH			LCC		
# Agents	4	8	12	4	8	12
NO	50	100	150	50	100	150
SR in %	100	96.0	88.7	100	94.0	87.3
NA	2697	10611	28948	2541	7528	22032
QDF	1	2.05	4.03	1	1.58	3.31
APT	53.94	110.54	217.57	50.82	80.08	168.25
APC	378	3289	13027	356	2108	9914
CRE in %	0.19	0.31	0.45	0.14	0.28	0.45
NM	-	-	-	356	4704	20052
MPT	-	-	-	7.12	50.04	153.13
NMA	-	-	-	2897	12232	42048
MAPT	-	-	-	57.94	130.13	321.1

Table 2: Results for Homogeneous Agent Societies of Types RWK, BCR, BCH, LCH, and LCC with 4, 8, and 12 Agents

A Table of Experimental Results

Table 2 displays the numerical results of the experiments with the five agent types in the loading dock reported above. The legend for table 2 is as follows:

RWK: random walker
BCR behaviour-based agent with random conflict resolution strategy
BCH: behaviour-based agent with heuristic conflict resolution strategy
LCH: local planner agent with heuristic conflict resolution strategy
LCC: local planner, cooperative conflict resolution strategy
NT: # of tasks
SR: success ratio
NA: # of performed actions
APT: # of actions per task
APC: # of actions per conflict resolution
CRE: % of actions spend for conflict resolution
QDF: quality degradation factor

NM: # of messages sent
MPT: # of messages per task
NMA: Total # of actions + # of messages
MAPT: Actions plus messages per task

APT and MPT have been computed by $\frac{NA}{NO \cdot SR}$ and $\frac{NM}{NO \cdot SR}$, respectively. That means that only successfully finished tasks have been taken into account for computing these values.

The quality degradation factor $QDF(x)$ for an x-agent experiment has been computed by $QDF(x) \stackrel{\text{def}}{=} \frac{NA(x) \cdot NO(4)}{NA(4) \cdot NO(x)} \cdot \frac{100}{SR}$.