ISIC/CIRA/ISAS '98 - (version of accepted paper),
September 14-17, 1998, Gaithersburg, Maryland

A Framework for Plan Execution in Behaviour-Based Robots

Joachim Hertzberg[†], *Herbert Jaeger*[†], *Philippe Morignot*[‡], *Uwe R. Zimmer*[†]

[†]German National Research Institute
for Information Technology (GMD)
Schloß Birlinghoven – 53754 Sankt Augustin
Germany

[‡]ILOG
9, rue de Verdun
92253 Gentilly
France

This paper presents a conceptual architecture for autonomous robots that integrates behaviour-based and goal-directed action as by following a traditional action plan. Dual Dynamics is the formalism for describing behaviour-based action. Partial-order propositional plans, which get generated by GRAPHPLAN, are used as a basis for acting goal-directedly; the concept is suitable for using other planning methods and plan representations, though. The paper presents the corresponding action and plan representations at the plan side and at the behaviour side. Moreover, it describes how behaviour-based action is biased towards executing a plan and how information from the behaviour side is fed back to the plan side to determine progress in the plan execution.

Keywords: **Plan execution, agent architectures, perception, reactive systems**

1. Introduction

An agent that cannot pursue long-term goals or that does not act in accordance with these goals whenever possible, is a bunch of reflexes, but not worth being called an agent. An agent that endlessly ponders its goals and is unable to react if and when circumstances so dictate, is a wise guy, but not worth being called an agent either. Much effort is currently put into the direction of building embedded agents that can do both [16], and the idea itself has a long tradition in AI, as the classical STRIPS/SHAKEY work [12] shows.

Taking an autonomous robot for an agent, agent architectures typically specify different layers (often, three of them), with a layer for symbolic reasoning and strategic planning on top, and at the bottom sits a control layer of elementary low-level actions (often, they are called reflexes or behaviours) that can map sensor signals directly into effector operation. Plans from the symbolic layer are somehow appropriately translated into 'programs' in terms of these low-level actions, whose execution may be flexibly modified to cope with unforeseen events or facts in the environment (often, this exibility is demonstrated in obstacle avoidance in navigation missions). [10], [17] are examples proving that this architecture can be made work – there are many more of them.

There seems to be a problem, though. The flow of control from the strategic planning layer down to the control layer can be handled, as exemplified by the research just mentioned. However, the information flow up to the planning layer is often impoverished. Ideally, the status of plan execution should be reported, including possible problems and failures, which should lead to, first, updating the planner's symbolic world model based on the available sensor and control information, and, second, plan repair or replanning if necessary. Such a schema does work for navigation, where the world model consists essentially in a robot position and orientation on a map; it has not been demonstrated to work generally in domains that are more typical for action planning, which require complex facts to be represented, tracked, derived, and sensed.

This paper describes another instance of coupling strategic planning and behaviour-based action. Its ingredients are not new, but their combination is, and so is its result. As a first, technical ingredient, we are using *dual dynamics* (DD) as a framework for formulating behaviours. DD is special in that it allows the *target dynamics* and the *activation dynamics* of a behaviour to be expressed separately, thereby adding some exibility to describing the robot's low-level actions.

Second, more on a conceptual level, we do not treat an agent's plans like a program that, once embarked on, should or does literally program its behaving like

a C program determines how a computer operates. Rather, we treat a plan as a resource for acting that is used as one among several information sources that together shape the combination of currently active behaviours. This *plans-as-resources* rather than *plans-as-programs* metaphor has been advocated by several researchers, e.g., [15].

Third, we do not attempt to sense the environment separate from acting. Instead, we draw the information relevant for action from the respective enabledness or disabledness of behaviours, where it is worth noting that not each physically enabled behaviour will get executed – plans help decide which among possibly many executable behaviours contributes most to the current overall behaving.

The ingredients just mentioned and their interplay are technically described in this paper. Presently, our work is in the state of a concept, with no implementation on one of our concrete robots available yet. Work in this direction is underway.

The paper, then, is structured as follows. The following section 2 gives a brief introduction into the basic concepts of DD. It also introduces en passant the demo domain of this paper, namely, an errands task, which is stripped of to a didactic micro version. The next section 3 gives the representation of the demo domain as it is employed in the planner. We are using GRAPHPLAN [2] as a planner and, accordingly, a propositional representation language on the planning level. It should be understood from the outset, however, that GRAPHPLAN was chosen to demonstrate that no specially designed planner is required for our purposes. The framework that we are presenting is neither restricted to using classical planning nor to a propositional representation language. In fact, having a richer language, such as ADL [13], might be helpful, but that is no issue in this paper.

Section 4 contains the heart of the paper, describing first, how the behaviours that correspond to operators in the plan mix into the overall behaviour of the agent, and, second, how information from the execution can be fed back into the planning representation. Section 5 concludes by discussing our approach in the light of previous work.

2. Dual Dynamics: A Brief Introduction

This section sketches some basic features of DD. For details, see, e.g., [9], [8]. It also introduces the demo domain of this paper, as seen from the DD side.

Dual dynamics is an approach to put behaviour-based robotics [3], [14] on a formal basis in terms of self-organizing dynamical systems. DD models an agent's complete behaviour control system as a continuous dynamical system, which is specified by ordinary differential equations (ODE's). The various

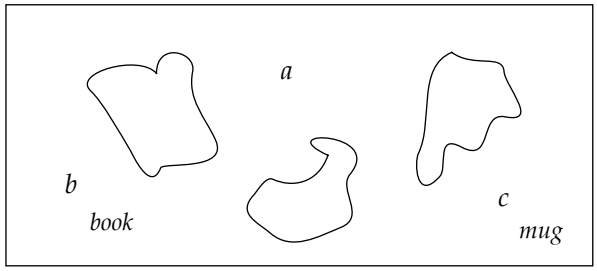

figure 1 : The environment in the errands domain

behaviours are modelled as subsystems. The theoretical contribution of DD is to explain how these subsystems are coupled, by sharing certain variables and by inducing bifurcations on each other. DD spells out several restrictions on which couplings are admissible. This results in an overall system behaviour which is transparent although subsystems undergo bifurcations. From a practical point of view, this transparency allows to design and debug complex behaviour systems. The approach is being used with mobile robots in the VUB AI Lab [7] and at the University of Bielefeld. Implementations of controllers for behaviour-based robots in terms of DD exist, using PDL [19], [18] as a particularly handy programming language.

In general, DD behaviour systems are hierarchic, with comprehensive, long-term behaviours (like **work** or **replenish-energy**) at higher levels and elementary, short-term behaviours (like **turn-left**) at lower levels. Much of DD theory is concerned with interactions between levels. For the purpose of the present paper however, an elementary single-level system is sufficient. Therefore, we here omit all that DD has to say about level organization.

Before we proceed with explaining DD, we shall introduce a demo arena including a simple robot. The arena is an enclosed area with obstacles and three distinct locations *a*, *b*, *c*. These locations are marked by acoustic beacons which emit different sounds. Some *books* and coffee *mugs* are distributed in the arena, not necessarily at the distinct locations. A variety of errand tasks can be formulated in this setup. Figure 1 sketches that instance of the scenario, which will get used for planning later.

The robot has two propulsion wheels which are independently controlled by motor signals m_r, m_l, where $m_r = 1$ means full speed forward of the right motor, and $m_r = -1$ means full retract (analogously for m_l). Thus, the robot can move forward and backward and turn with different speeds (even on the spot). Furthermore, the robot has a gripper, which it can use to gather and dump a book and a mug. For simplicity, we assume that the gripping system is in itself quite sophisticated. It can be triggered to gather the book and put it in its payload space, simply by setting a 0-1-valued control input g_{book} to 1 for some seconds.

Analogically, the gripper is triggered to gather the mug, dump the book, and dump the mug by raising g_{mug}, d_{book}, and d_{mug} to 1.

The robot can sense obstacles with two active range sensors mounted to its left and right front, whose areas of perception overlap in front of the robot. They return continuous values $o_l, o_r \in [0, 1]$. It reads $o_l = 1$ or $o_r = 1$ if an obstacle is very close to the left resp. right, and $o_l = 0$ resp. $o_r = 0$ if no obstacle is seen on the left or the right.

The beacons in places *a*, *b*, *c* can be perceived by an acoustic sensor which returns three values $\varphi_a, \varphi_b, \varphi_c \in [-1, 1]$ where $\varphi_a = -1$ means that *a*'s sound comes at 90° from the right, a reading of $\varphi_a = 0$ means that *a* is straight ahead, and $\varphi_a = 1$ implies *a* lies exactly left. Distance to *a*, *b*, *c* is measured by binary variables $\delta_a, \delta_b, \delta_c$ which usually read 0 but jump to 1 when the corresponding beacon is reached.

Finally, we assume the robot is equipped with a special book-and-mug detector (presumably relying on books being bright red, mugs being green, and the rest of the world coming in shades of grey). This detector yields two binary variables γ_{book}, γ_{mug} which jump to 1 when a book or a mug are perceived within a distance which is near enough for the object to be gripped.

Now we return to explaining DD, by discussing a simplistic DD behaviour system for this robot. This system consists of the behaviours (1) **approach**$_a$, (2) **approach**$_b$ (3) **approach**$_c$, (4) **roam**, (5) **gather**$_{book}$, (6) **gather**$_{mug}$, (7) **dump**$_{book}$, and (8) **dump**$_{mug}$. (We will sometimes refer to the behaviours by the numbers just given.)

When no plan is present, these behaviours interact and produce the following default, global behaviour pattern. The robot randomly alternates between approaching the beacons, and roaming about the arena. While manoeuvring, obstacles are avoided. Whenever a book is passed near enough for gathering, the robot first dumps a book (if it had none, it will perform a void dump motion), then loads the book freshly encountered. The same occurs when a mug is encountered. We will now describe the DD specification of the **approach**$_a$ behaviour in some detail.

The first thing to note about DD is that an elementary behaviour is specified as a compound system consisting of a *target dynamics* and an *activation dynamics* subsystem – hence the naming, 'dual dynamics'.

The target dynamics subsystem of a behaviour specifies the target trajectories of all actuators relevant for the behaviour. In the case of the **approach**$_a$ behaviour, the relevant target variables are m_l and m_r (see figure 2), since for approaching *a*, only the drive motors are relevant. Thus, the target dynamics of **approach**$_a$ consists of two ODE's for m_l and m_r.

Behaviour 1: ***approach***$_a$

Target dynamics

$$\dot{m}_l = k_1\Big(\frac{1}{2} + \frac{1}{2}o_l - \frac{1}{2}o_r - \frac{1}{2}\varphi_a - m_l\Big) + k_2(o_l + o_r)(0 - m_l) + k_3\delta_a(0 - m_l) \quad (1)$$

$$\dot{m}_r = k_1\Big(\frac{1}{2} + \frac{1}{2}o_l - \frac{1}{2}o_r - \frac{1}{2}\varphi_a - m_r\Big) + k_2(o_l + o_r)(0 - m_r) + k_3\delta_a(0 - m_r) \quad (2)$$

Activation dynamics

$$\dot{\alpha}_1 = k_4\Big(1 - \sigma\Big(\sum_{i \neq 1}\alpha_i\Big) - \alpha_1\Big) + k_5\delta_a(0 - \alpha_1) \quad (3)$$

+ noise + operator-coupling terms

figure 2 : Definition of the **approach**$_a$ behaviour

Let us consider the equation for m_l. We assume that the reader is familiar with the basics of ODE's. Equation (1) has three additive components. The first component tells the left motor to take on a default forward speed of 1/2, which is accelerated when an obstacle is sensed to the left, decelerated when an obstacle is sensed to the right, and accelerated/decelerated according to the *a*-direction sensor reading. The second component pulls a break on the motor proportionally to overall distance to obstacles, resulting in a slowdown near obstacles.[1] The third component breaks when a is reached. The equation for the right motor target m_r is analogous.

If k_3 is considerably greater than k_1 (which it should be), equations (1) and (2) should make the robot generally drive toward *a*, avoiding obstacles (and walls) on the way, and slow down almost to a standstill in the direct vicinity of *a*.

The k_i are time constants that have to be suitably chosen for the dynamics to work as desired. Much of the DD designer's know-how concerns the suitable selection of such time constants. In this article we shall not further discuss this hairy issue.

Now we explain the activation dynamics subsystem. This subsystem regulates a single variable α_1, the behaviour's *activation*. Generally, every DD behaviour owns an activation variable, whose dynamics should

[1] *In this equation and others to follow, factors* $(o - v)$ *occur in definitions of* $\dot{v}$ *for variables v. We use them instead of writing -v to make clear that the respective term pulls v towards 0 in the same way that a factor* $(1 - v)$ *pulls v towards 1.*

Behaviour 4: ***roam***

Target dynamics

$$\dot{m}_l = k_6\Big(\frac{1}{2} + \frac{1}{2}o_l - \frac{1}{2}o_r - m_l\Big) + k_7(o_l + o_r)(0 - m_l) \quad (4)$$

$$\dot{m}_r = k_6\Big(\frac{1}{2} + \frac{1}{2}o_l - \frac{1}{2}o_r - m_r\Big) + k_7(o_l + o_r)(0 - m_r) \quad (5)$$

Activation dynamics

$$\dot{\alpha}_4 = k_8\Big(1 - \sigma\Big(\sum_{i \neq 4}\alpha_i\Big) - \alpha_4\Big) \quad (6)$$

+ noise + operator-coupling terms

figure 3 : Definition of the **roam** behaviour

result in a variation range between 0 and 1. An activation of 1 means that the behaviour is 'enabled', i.e., the target values produced by the target dynamics subsystem are passed on to the actuators. An activation value of 0 means that target values are not passed on, but are 'inhibited'. Thus, the activation variable of a behaviour can be viewed as a gatekeeper, which decides *when* the behaviour influences the actuators.

Although the activation dynamics subsystem has to rule only a single variable, this subsystem can become surprisingly complex. In the case of $\mathbf{approach}_a$, the activation dynamics consists of 4 kinds of additive terms. The function σ appearing in the first is a suitable thresholding function (e.g., a sigmoid), which rises to 1 when its argument surpasses some threshold, and is about 0 otherwise. The first term thus states that α_1 rises to 1 unless some other behaviour's activations are noticeable, in which case α_1 is pulled to 0. The second term pulls α_1 to zero when a is reached. Some small amount of noise is added to avoid deadlocks. Finally, a number of 'operator-coupling terms' further shape the activation dynamics to enable plan execution. They will be explained in section 4.

The remaining behaviours shall be described more briefly. $\mathbf{approach}_b$ and $\mathbf{approach}_c$ are analogous to $\mathbf{approach}_a$. The equations for behaviour (4), **roam** (figure 3), resemble those of $\mathbf{approach}_a$. While this behaviour is active (i.e., while α_4 is big), the robot simply should drive forward with default speed 1/2, slow down before and turn away from obstacles, and resume standard forward motion in a new direction.

Behaviour 5: $\mathbf{gather}_{book}$

Target dynamics

$$g_{book} = 1\,,\quad g_{mug} = d_{book} = d_{mug} = m_l = m_r = 0 \quad (7)$$

Auxiliary variables (for behaviours 5 and 7)

$$\dot{\beta}_1 = k_9(0 - \beta_1) + k_{10}\gamma_{book}(1 - \beta_1) \quad (8)$$

$$\dot{\beta}_2 = k_{11}\sigma\dot{\beta}_1(1 - \beta_2) + k_{12}(1 - \beta_2) \quad (9)$$

Activation dynamics

$$\dot{\alpha}_5 = k_{13}((1 - \sigma_1(-\dot{\beta}_2))\sigma_2(-\dot{\beta}_2) - \alpha_5) \quad (10)$$

+ operator-coupling terms

figure 4 : Definition of the $\mathbf{gather}_{book}$ behaviour

Behaviour 7: $\mathbf{dump}_{book}$

Target dynamics

$$d_{book} = 1\,,\quad g_{book} = g_{mug} = d_{mug} = m_l = m_r = 0 \quad (11)$$

Activation dynamics

$$\dot{\alpha}_7 = k_{14}(\sigma_1\dot{\beta}_2 - \alpha_7) \quad (12)$$

+ operator-coupling terms

figure 5 : Definition of the $\mathbf{dump}_{book}$ behaviour

Since the activation dynamics basically only says, 'unless any other behaviour is active, **roam**', roaming is the fallback behaviour which takes over whenever nothing else is going on.

The target 'dynamics' of the gather and dump behaviours (figure 4 and 5) are simple constants and thus need not be specified by ODE's. The activation dynamics are a bit more involved. The rationale is that after spotting a book (i.e., after γ_{book} jumps to 1), for some fixed time interval $\mathbf{dump}_{book}$ becomes activated, after which in another time interval $\mathbf{gather}_{book}$ gets active. This involves a timing mechanism which is triggered by γ_{book}. There are many ways of implementing such a timing. We used auxiliary variables β_1, β_2 to this end. We include the equations without going into details. (The two time windows are determined by the thresholds for σ_1 and σ_2, where the latter is smaller than the former.)

3. Plans in the Errands Domain

We are now very briefly looking at the errands domain from the planning side, starting with its representation for the planner. This planner happens to be GRAPHPLAN. Note, however, that this choice was exclusively guided by the desire to keep simple the planning representation and the planning process for investigation and explanation. Other planners with their respective representations should be usable as well, but have not yet been studied.

The domain signature, start situation and goal conditions as appropriate for GRAPHPLAN are given in figure 6. The original robot location is *a*; the goal is to have both the *book* and the *mug* at location *a*. We have three operators GOTO, LOAD, and UNLOAD, all with the intuitive interpretation. Figure 7 defines them in GRAPHPLAN syntax. Finding a plan for solving the resulting planning problem is straightforward. This plan is given in figure 8a and 8b.

Planning operators may, but need not correspond directly to behaviours in the DD representation. In the errands domain, there are the obvious correspondences between LOAD and **gather**, UNLOAD and **dump**, and GOTO and **approach**, where the behaviours need not mention the current or past locations. In more complex domains, the planner may work with operators that correspond to higher-level behaviours, or it may work with macros that get expanded before plan execution into elementary operators corresponding to behaviours of some level. We have not investigated that yet, but we assume it is practical. The point is: In any case, the designer of the complete plans-plus-DD domain representation has to make sure that the operators make connection to the behaviours in the technical sense explained next.

```
(mug OBJECT) (book OBJECT)
(a LOCATION) (b LOCATION) (c LOCATION)

(preconds (at mug c) (at book b) (at robot a)

(effects (at mug a) (at book a))
```

figure 6 : Types, initial, and final situation

```
(operator GOTO
    (params (<l1> LOCATION) (<l2> LOCATION))
    (preconds (at robot <l1>))
    (effects (del at robot <l1>) (at robot <l2>)))

(operator LOAD
    (params (<o> OBJECT) (<l> LOCATION))
    (preconds (at <o> <l>) (at robot <l>))
    (effects (del at <o> <l>) (has-robot <o>)))

(operator UNLOAD
    (params (<o> OBJECT) (<l> LOCATION))
    (preconds (has-robot <o>) (at robot <l>)
    (effects (at <o> <l>) (del has-robot <o>)))
```

figure 7 : Planning operators, in GRAPHPLAN syntax

4. Coupling Symbols and Dynamics

4-1. From symbols to dynamics

This subsection explains how operators, which are symbolic entities, are executed by the DD system, which is dynamic. As will become apparent soon, 'executed' is a somewhat misleading term. The DD system is not strictly commanded to carry out an action. Rather, it is biased more or less strongly in its natural ongoing activity such that the operator's effects are likely to be achieved.

We assume that a single operator is picked for execution at every point of time. Usually, this is one of the next executable operators of the current plan. Alternatively, the robot can be given from a user single operators to execute.

We shall demonstrate the biasing mechanism using an example. Assume that plan execution calls on the operator GOTO(*L*, *b*) for some location *L*. From our

```
1  GOTO_a_c
2  LOAD_mug_c
3  GOTO_c_b
4  LOAD_book_b
5  GOTO_b_a
6  UNLOAD_mug_a
6  UNLOAD_book_a
```

figure 8a: A four-action plan in GRAPHPLAN's format

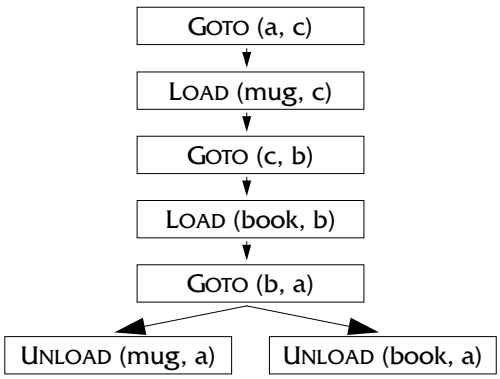

figure 8b: A four-action plan in the standard graph representation of partial orders

Behaviour 2: ***approach***$_b$

$$\dot{\alpha}_2 = \ldots + s_{Goto(L,b)} c^2_{Goto(L,b)} (1 - \alpha_2) \quad (13)$$

Behaviour 4: ***roam***

$$\dot{\alpha}_4 = \ldots + s_{Goto(L,b)} c^4_{Goto(L,b)} (1 - \alpha_4) \quad (14)$$

Behaviour i: all others

$$\dot{\alpha}_i = \ldots + s_{Goto(L,b)} c^i_{Goto(L,b)} (0 - \alpha_i) \quad (15)$$

figure 9 : OCT's for GOTO (*L*, *b*)

observer's perspective, this intuitively means that the robot should go to *b*. But as it is intelligent and the world is hardly predictable, the robot should not just quit doing whatever it was doing before, and push its way toward *b* regardless of circumstances. If, for instance, the robot's battery is going down, it should take a break in going to *b*, and recharge for a while. Or, if the robot was just UNLOADing a mug when the operator GOTO(*L*, *b*) was called, the unloading should first be finished. For reasons like this, calling the operator GOTO(*L*, *b*) should result in a general, persistent tendency for the robot to proceed toward *b*, yet leave some freedom to do other things if circumstances so require.

This kind of 'operator-oriented biasing' of the DD system is effected via the operator-coupling terms (OCTs). In every behaviour's activation dynamics, there is an OCT for the operator GOTO(*L*, *b*). These OCTs should result in the desired persistent tendency to move toward *b*.

Figure 9 presents suitable OCTs for GOTO(*L*, *b*). As can be seen from these examples, OCTs have a simple common format (16):

$$\dot{\alpha}_j = \ldots + s_{OP} c^j_{OP} (Z - \alpha_j) \quad (16)$$

where s_{OP} is a switch variable which jumps to 1 when the operator *OP* is called and is otherwise 0, c^j_{OP} is a non-negative constant, and $Z \in \{0, 1\}$.

These OCTs work by superimposing an influence on the dynamics of the dynamics of α_j. This influence is a pull toward 0 if $Z = 0$, i.e., a 'discouragement' of the corresponding behaviour. It is an encouraging pull toward 1 if $Z = 1$. The strength of this dis- or encouragement is determined by the time constant c^j_{OP}. If it is low, the behaviour's activation dynamics is only mildly modified by the OCT. If it is much higher than other time constants in the rest of the behaviour's activation dynamics equation, then the behaviour is mandatorily activated ($Z = 1$) or deactivated ($Z = 0$) by the OCT. The influence is switched on or off by s_{OP}. This switch variable yields the causal connection between the symbolic plan execution mechanism and the DD system: whenever the plan execution calls an operator *OP*, the switches s_{OP} jump to 1 in all behaviour's activation dynamics.

Returning to the example (fig. 8), we now see that when GOTO(*b*) is called, behaviours **approach**$_b$ and **roam** are encouraged with strengths $c^2_{GOTO(L,b)}$ and $c^4_{GOTO(L,b)}$, respectively. Since the behaviour **approach**$_b$ in this case rather directly does what the operator is intended for, the encouragement should be strong. It seems reasonable to also encourage **roam** a bit, just in case **approach**$_b$ does not get active for some reason. The other behaviours do not apparently contribute to the operator's intention. Therefore, the corresponding OCTs feature $Z = 0$.

Using this metaphor of dis- and encouragement, and the mechanism of OCTs, it is not difficult to spell out plausible OCTs for the other operators used in our errand scenario. For instance, an operator LOAD(*book*, *L*) should encourage the behaviour **gather**$_{book}$ rather strongly, discourage **dump**$_{book}$ strongly, and discourage all other behaviours mildly.

This kind of designing OCTs might seem a bit vague or unprincipled. We do not really 'control' the behaviour by plans this way. Nor do we oversee all eventualities, and account for them. We believe that this is all right for autonomous robots working in unpredictable environments – it is hard anyway to fully *control* action in the face of unknowable external circumstances, or in the face of a complexity of behavioural interactions which cannot be premeditated.

4-2. From Dynamics to Symbols

Both the planner and the plan execution monitor are working on the symbolic level, as specified in section 3. So the gap must be closed between the information that is available on the DD side and the information that is required on the symbolic side.

Obviously, making symbolic world descriptions from sensor readings – or 'turning pixels into predicates', as the buzz word goes – is no easy task. In our approach, we make a strong methodological commitment at this place: we do not use sensor readings, but only actions of the robot to obtain world descriptions. The motto is: 'Turn interactions into predicates!' Sensor readings help build the world descriptions only indirectly, due to the fact that actions are sensor-based. This is one of the central points of our proposal. We shall now proceed, first, by giving our epistemological reasons for this commitment, and second, by explaining how it can technically be realized in our DD-based robot.

From an epistemological perspective, we believe that agents cannot perceive the world per se. Pure, objective sensing does not exist, and observation yields no such thing as a 'true' world description. This negative statement is probably uncontroversial. The question is, by which positive statement can it be replaced? We propose to make upworld models from interaction experiences. Simply put, we should allow

in our world models *concepts* like tables and mugs only in terms of our interactions with these objects – if we can walk toward them, touch them, lift them up to feel their heaviness, move them around, drink from them, scratch their surface. Furthermore, we should allow in our world models *propositions* like 'the mug is on the table' only in terms of our doings – because we actively put the mug on the table, or told another person to put it there (where telling is an action). The only exception from this rule is that we may add/delete propositions to/from our world model if we are being told by an agent whom we believe. Again, however, this does not rely on sensing!

Technically, we propose to make the symbolic world model from the activation history of behaviours. The history of the activation variables α_i contains the essence of what the robot did. For instance, ***At*** (*mug*, *b*) holds in the current world model if the **approach**$_b$ behaviour was recently active and uninterrupted, followed by the **dump**$_{mug}$ behaviour. There is a caveat, however. Since the activation dynamics of a behaviour obeys different rules at different times, according to which s_{OP} are switched on, we must also consult the history of the switch parameters in order to interpret the activation variables properly.

Thus, as the experiential source of information from which we distil a world model, we take the past history of all α_i and all s_{OP}. Additionally, we allow the robot to be told propositions by other agents whom it believes (presumably, its human operators).

Deriving propositions from activations requires some care. We will illustrate this now with a concrete example. The errands domain involves just two types of propositions: For objects O and locations L, these are ***Has-Robot*** (O) and ***At*** (O, L); ***At*** (*robot*, L) is a special instance of ***At***. To start with ***Has-Robot*** (*book*), the only way of having the book physically on board is having **gather**ed it in the past, and not **dump**ed it later. If we write $\alpha_i(t)$ for α_i in the past time point t, we have

$$\begin{aligned} &\textbf{\textit{Has-Robot}}(book) \leftarrow \qquad (17)\\ &\exists t_1\{\alpha_5(t_1) \approx 1 \wedge \forall t_2\{t_2 > t_1 \rightarrow \neg(\alpha_7(t_2) \approx 1)\}\} \end{aligned}$$

At (*robot*, *a*) is somewhat clumsier. Let Δ be a constant denoting an upper limit of the duration that it takes to deactivate **approach**$_a$. Then we have

$$\begin{aligned} &\textbf{\textit{At}}(robot, a) \qquad (18)\\ &\quad \leftarrow \exists t_1, t_2\\ &\quad \{t_1 < t_2 \wedge t_2 - t_1 < \Delta\\ &\quad \wedge \alpha_1(t_1) \approx 1 \wedge \alpha_1(t_2) \approx 0\\ &\quad \wedge \forall t, i\{t_2 < t \wedge \alpha_i(t) \approx 1 \rightarrow i \notin \{2, 3, 4\}\}\\ &\quad \wedge \exists L\{\forall t \in [t_1, t_2]\{s_{GOTO(L,a)}(t) = 1\}\}\} \end{aligned}$$

Let us explain this line by line. t_1 was less than Δ before t_2; **approach**$_a$ was active in t_1, and it was inactive in t_2; none of the behaviours that would have destroyed ***At*** (*robot*, *a*), i.e., none of the behaviours 2, 3, and 4, was active after t_2; and the robot was determined to go to *a* within the critical time interval [t_1, t_2], i.e., **approach**$_a$ has not been switched off by chance, such as by gathering a book just spotted when executing an **approach**$_a$ behaviour.

Before defining ***At*** (O, L), notice first that we cannot – neither want to – guarantee that the robot knows the locations of all objects, not even of all those that it has dumped in the past and not touched later. If such a **dump** did not take place at one of the distinct locations that the robot explicitly knows about and knows how to approach, then the object is simply lost. (Note that it may find it again while roaming about the arena; in this case, it may gather the object and thus bring it back into focus.) O, then, is known to be at L if at some time ***At*** (*robot*, L) in the sense of the definition (16), within a small, user-defined time window Δ' after deactivation of **approach**$_a$, if O was then dumped, and O was not gathered later. The corresponding formalization is simply technical.

Using these predicate definitions, the robot can make available a situation description at every moment in time, which is formulated in terms of the planner's vocabulary. It may be practical to update the situation description incrementally rather than computing it from the complete available activation history all the time. Finally, there may be another way for the robot to learn information for this description, namely, by being told. We do not go into this issue here, but we assume that the ability of learning by acquiring symbolic terms and to respect this knowledge in future acting, is something that suits well an intelligent agent.

5. Concluding Discussion

The contribution of this work can be seen from two perspectives. From the behaviour-based action point of view, it offers a way of sequencing behaviours to achieve longer-term goal-directedness, yet keeping intact the basic ideas and principles of behaviour-basedness. The current plan is used as an important source of information – no more and no less. It modulates ongoing action rather than enslaving it. We feel that this is an adequate way of putting the *plans-as-resources* metaphor [15] to work.

From the planning point of view, our work contributes a principled way of coupling the plan level and the action level. It has been obvious from the beginnings of planning research that operators must be 'implemented' in terms of executable procedures, and this has been done in each and every practical planner let loose on a real domain. Plans intended for helping humans organize their work – like, say, in job-shop scheduling – have never suffered from this coupling problem, because the human in the loop maps plans to actions and action results back to the planning representations. Our work is obviously un-

necessary and not intended for this case of planning applications.

In the case of autonomous robots, no humans are in the loop, and no homunculi should be replacing them. While there are quite a number of examples for systems translating plans successfully into physical action – see, e.g., [12], [1], to name just one classical and one recent work – it seems that attempts to feed back information about the world into the planning representation were less successful. This does not seem to come by chance as turning sensor readings into symbolic descriptions poses all sorts of problems, from technical to epistemological ones.

We are tackling the problem by perceiving the world through the history of activation values. This is different from interpreting the sensor data directly in that it sees the world filtered through action, thus reducing information sucked out of the environment to that part, which is relevant for acting by the very definitions of the behaviours. We are not aware of a like approach in the context of planning for autonomous robots.

As another feature of our framework, the planning level and the behaviour level are to some degree decoupled in time, and they are meant to work in parallel. This idea is not new, the ATLANTIS framework [5] being another instance.

At the present state of work (and within the limits of this paper), many issues remain unaddressed and many problems unsolved. The first and foremost is to implement on a real robot the concept that we have described, the most plausible robot candidate being a RWI B-14, which we have available.

The choice of GRAPHPLAN as a planner and, accordingly, its domain language as a representation language was primarily guided by the desire for simplicity in the beginnings of our work. We do not want to suggest that classical, propositional partial order planning is the planning tool of choice for autonomous robots. As for the planning process, some *anytime* [4] flavour would suit a robot planner well – and it seems that this would fit nicely into our framework. As for the domain representation language, a somewhat higher expressivity, e.g., in the ADL direction, would be convenient; it remains to be seen, however, how such a language connects with the perception in terms of activation values – this necessity constrains our choice among the representation languages that we like best.

On the other hand, we have come to like the simplicity of the GRAPHPLAN approach for autonomous robots. The point is: The fact that the world is partially unknown, unknowable, and chaotic to the robot does not necessarily mean that this lack of knowledge must be reproduced and handled in the planner's domain representation. If the robot is able to change its current view of the world quickly and replan fast enough for the new situation, then it might be as well off with many, rapidly-changing, simple plans as with few, relatively stable, sophisticated plans that handle planning uncertainty. We are not sure about where a robot planner within our framework should best be positioned in the menagerie of AI planning methods, but it does not come by chance that we have not used anytime uncertainty planning methods available.

Turning to a technical problem, remember that a plan is coupled with the behaviours by using OCTs, where it is assumed that the planner picks the current operator for execution. While this is a clear scheme, it is not completely satisfying. It should be the whole plan that biases behaviour, thus allowing to take favour of serendipity, to fix exogenous goal destruction, and to jump ahead in the plan opportunistically [6] – all on the behaviour-based execution side. Designing plan-coupling terms instead of OCTs is a point of further research.

To end with a more general issue, the overall world information, as described, is obtained by the activation value history, the switch variables, and possibly symbolic information given right away. This is a whole lot of information. In particular, there will in general be more of it than a planner – be it GRAPHPLAN or something fancier – can handle fast. So a mechanism seems to be needed for filtering the information that looks relevant 'in the moment' out of the overall information, in order to tailor the planning problem that is given to the planner. While this idea is far from new (it is the background for the infamous qualification problem [11]), the problem is obviously unsolved. Our framework allows to tackle the question from a new perspective.

In sum, we feel that the approach just presented has some potential for merging in a principled way the best of the behaviour-based and the plan-based world for controlling autonomous robots.

References

[1] Ruth Aylett, Alex Coddington, David Barnes, and Rob Ghanea-Hercock
What does a planner need to know about execution?
In Preprints of Fourth European Conference on Planning, pages 28-40, 1997

[2] A.L. Blum and M.L. Furst
Fast planning through plan graph analysis
Artif. Intell., 90:281-300, 1997

[3] R. A. Brooks
Intelligence without reason.
A.I. Memo 1293, MIT AI Lab, 1991.
ftp'able at http://www.ai.mit.edu/

[4] T.L. Dean and M. Boddy
An analysis of time-dependent planning
In Proc. AAAI-88, pages 49-54, 1988

[5] Erann Gat
Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling a Real-world Mobile Robot
In Proc. AAAI-92, San Mateo, CA, 1992. Morgan Kaufmann

[6] B. Hayes-Roth and F. Hayes-Roth
A cognitive model of planning
Cogn. Sci., 3:275-310, 1979

[7] H. Jaeger
Brains on wheels: Mobile robots for brain research
Manuscript, available at http://www.gmd.de/People/Herbert.Jaeger/Publications.html, 1996

[8] H. Jaeger and Th. Christaller
Dual dynamics: Designing behavior systems for autonomous robots
In S. Fujimura and M. Sugisaka, editors, Proc. Int. Symposium on Articial Life and Robotics (AROB'97), pages 76-79, 1997

[9] Herbert Jaeger
The Dual Dynamics Design Scheme for Behavior-based Robots: A Tutorial
Arbeitspapiere der GMD 966, GMD, January 1996

[10] F. Kirchner and J. Hertzberg
A prototype study of an autonomous robot platform for sewerage system maintenance
J. Autonomous Robots, 4(4):319-331, 1997

[11] J. McCarthy and P. Hayes
Some philosophical problems from the standpoint of articial intelligence
Machine Intelligence, 4:463-507, 1969

[12] N.J. Nilsson
Shakey the robot
Technical Report Technical Note 323, SRI International, April 1984

[13] E.P.D. Pednault
ADL: Exploring the middle ground between strips and the situation calculus
In Proc. KR-89, pages 324-332, 1989

[14] R. Pfeifer and Ch. Scheier
An Introduction to New Articial Intelligence
To appear in MIT Press, 1997

[15] M.E. Pollack
The uses of plans
Artif. Intell., 57(1):43-68, 1992

[16] S. Russell and P. Norvig
Articial Intelligence: A Modern Approach
Prentice Hall, Englewood Cliffs, NJ, 1995

[17] A. Saffiotti, K. Konolige, and E.H. Ruspini
A multivalued logic approach to integrating planning and control
Artif. Intell., 76:481-526, 1995

[18] D. Spenneberg, E. Schlottmann, T. H□opfner, and Th. Christaller
PDL programming manual
Arbeitspapiere der GMD 1082, GMD, 1997

[19] L. Steels
Building agents out of autonomous behavior systems
In L. Steels and R.A. Brooks, editors, The 'Articial Life' Route to 'Articial Intelligence': Building Situated Embodied Agents. Lawrence Erlbaum, 1993