In Proc. ISIC/CIRA/ISAS'98, pp. 8-13

A Framework for Plan Execution in Behavior-Based Robots

J. Hertzberg, H. Jaeger, U. Zimmer
GMD, Schloss Birlinghoven,
53754 Sankt Augustin, Germany

P. Morignot
ILOG, 9, rue de Verdun,
92253 Gentilly, France

ABSTRACT

We present a conceptual architecture for autonomous robots that integrates behavior-based and goal-directed action as by following a traditional action plan. Dual Dynamics is the formalism for describing behavior-based action. Partial-order propositional plans are used as a basis for acting goal-directedly; the concept is suitable for using other planning methods and plan formats, though. We describe the corresponding action and plan representations at the plan side and at the behavior side. Moreover, we demonstrate how behavior-based action is biased towards executing a plan and how information from the behavior 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, and the idea itself has a long tradition in AI, e.g., [7].

Taking an autonomous robot for an agent, robot 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 low-level actions (often called reflexes or behaviors) that can map sensor signals directly into effector operation. Plans from the symbolic layer are translated into "programs" in terms of these low-level actions, whose execution may be flexibly modified to cope with unforseen events in the environment (often, this is demonstrated in obstacle avoidance in navigation). [11, 2] are examples proving that this architecture can work.

There is a problem, though. The flow of control from the strategic planning layer down to the control layer can be handled, as shown by the research just mentioned. But the information flow up to the planning layer is often impoverished. Ideally, the status of plan execution should be reported, including problems and failures, which should lead to, first, updating the planner's symbolic world model based on 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 a new instance of coupling strategic planning and behavior-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 behaviors. DD is special in that it allows the *target dynamics* and the *activation dynamics* of a behavior to be expressed separately, thereby adding some flexibility to describing the robot's low-level actions. Second, we do not treat an agent's plans like a program that, once embarked on, does literally program its behaving. 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 behaviors. This *plans-as-resources* rather than *plans-as-programs* metaphor has been advocated by several researchers, e.g., [10]. Third, we present a way of deriving high-level world features from the enabledness or disabledness of behaviors, thereby filtering from the possibly huge amount of sensor data those parts that are relevant for acting by definition of the behaviors.

This paper describes these ingredients and their interplay. Presently, our work is in the state of a concept, with no complete implementation on a robot available yet. Work in this direction is underway. The paper is structured as follows. Section 2 introduces basic DD concepts as well as the demo domain of this paper. Section 3 gives the representation of the demo domain as used in the planner. We are using a standard planner, GRAPHPLAN [1], to demonstrate that our framework does not require a specially designed planning method; accordingly, we use a propositional representation language for planning. Section 4, the heart of the paper, describes how behaviors that correspond to plan operators influence the robot's action, and, second, how information from the execution is fed back into the planning representation. Section 5 concludes.

2 DUAL DYNAMICS

This section sketches some basic features of DD. For details, see, e.g., [4, 6]. It also introduces the demo domain

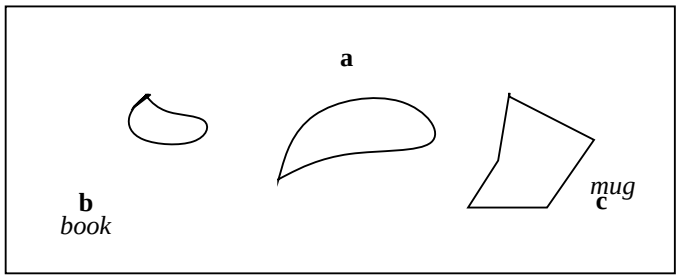

Figure 1: A scene in the errands domain.

of this paper, as seen from the DD side.

DD is an approach to put behavior-based robotics [3, 9] on a formal basis in terms of self-organizing dynamical systems. It models an agent's complete behavior control system as a continuous dynamical system, which is specified by ordinary differential equations (ODE's). The behaviors are modeled 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 restricts these couplings, resulting in an overall system performance that is transparent although subsystems undergo bifurcations. This transparency allows to design and debug complex behavior systems. Ample experience exists in using DD for robot control, e.g., [5], and a DD programming language called PDL [14, 13] as well as teaching material for DD/PDL [12] are available.

In general, DD behavior systems are hierarchic, with comprehensive, long-term behaviors (such as **work** or **replenish-energy**) at higher levels and elementary, short-term behaviors (like **roam**) at lower levels. Much of DD theory is concerned with interactions between levels, but this sketch omits all that DD has to say about level organization, as a single-level system is sufficient for the purpose of the paper.

For explaining DD, we shall use the *errands domain:* an enclosed area with obstacles and three distinct locations *a, b, c*, over which some *book*s and coffee *mug*s are distributed (not necessarily at *a, b, c*). A variety of errand tasks can be formulated in this setup. Fig. 1 sketches that instance of the scenario which will get used for planning later.

The robot's two wheels are controlled by independent 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. It has a gripper, which it can use to gather and dump a book and a mug. For ease of presentation, we assume that the gripping system is quite sophisticated. It can be triggered to gather the book and put it in its loading space, simply by setting a 0-1-valued control input g_{book} to 1 for some time. 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 senses obstacles ahead with two range sensors on its left and right front, which return continuous values $o_l, o_r \in [0, 1]$. $o_l = 1$ (resp. $o_r = 1$) if an obstacle is immediately ahead left (right), and $o_l = 0$ ($o_r = 0$) if space is open to the left (right). Acoustic beacons are installed in *a, b, c* and can be perceived by sensors that return values $\varphi_a, \varphi_b, \varphi_c \in [-1, 1]$, where $\varphi_a = -1$ means that *a*'s sound comes at 90 degrees from the right, $\varphi_a = 0$ means *a* is straight ahead, and $\varphi_a = 1$ implies *a* lies exactly left. Proximity to *a, b, c* is signalled 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 has a special book-and-mug detector, yielding binary variables $\gamma_{book}, \gamma_{mug}$ which jump to 1 when a book (mug) is perceived close enough for gripping.

A simplistic behavior system for this robot consists of the behaviors (1) **approach**$_a$, (2) **approach**$_b$ (3) **approach**$_c$, (4) **roam**, (5) **gather**$_{book}$, (6) **gather**$_{mug}$, (7) **dump**$_{book}$, and (8) **dump**$_{mug}$. (In some formulas presented below, we will refer to the behaviors by their numbers.) When no plan is present, these behaviors interact and produce the following default, global behavior pattern. The robot randomly alternates between approaching the beacons, and roaming about the arena, avoiding obstacles in the process. Whenever a book (mug, resp.) is passed close enough for gathering, the robot first dumps a book (mug) if carrying one, and then loads the book (mug) just found.

The first thing to note about DD is that an elementary behavior is specified as a compound system consisting of a *target dynamics* and an *activation dynamics*—hence the name "dual dynamics". The target dynamics subsystem of a behavior specifies the target trajectories of all actuators relevant for the behavior. Taking **approach**$_a$ (fig. 2) as an example, the relevant target variables are m_l and m_r, since only the drive motors are relevant for approaching *a*. Thus, the target dynamics of **approach**$_a$ consists of two ODE's for m_l and m_r.

The $\dot{m}_l$ equation 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 changed according to φ_a. The second component results in a slowdown near obstacles.[1] The third component breaks when *a* is reached.

The k_i are time constants that have to be suitably chosen for the dynamics to work as desired. Much of the domain modeler's knowhow concerns the suitable selection of such time constants. In this article, we shall not further

[1] In this equation and others to follow, factors $(0 - v)$ 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.

Target dynamics

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

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

Activation dynamics

$$\dot{\alpha}_1 = k_4(1 - \sigma(\textstyle\sum_{i\neq 1}\alpha_i) - \alpha_1) + k_5\delta_a(0 - \alpha_1) + \text{noise} + \text{operator-coupling terms}$$

Figure 2: Definition of behavior #1: **approach**$_a$ for approaching location a. See text for explanations.

discuss this hairy issue. If k_3 is considerably greater than k_1, then **approach**$_a$'s target dynamics should make the robot generally drive toward a, avoiding obstacles on the way, and slow down almost to a standstill at a.

Now turn to the activation dynamics. This subsystem regulates a single variable α_1, the behavior's *activation*. Generally, every DD behavior owns an activation variable, whose range is [0,1]. An activation of 1 means that the behavior is "enabled", i.e., the target values produced by the target dynamics subsystem are passed on to the actuators. Activation 0 means target values are not passed on, but are "inhibited". Thus, the activation variable of a behavior can be viewed as a gatekeeper, which decides *when* the behavior influences the actuators.

Although the activation dynamics rules just a single variable, this subsystem can become quite complex. **approach**$_a$'s activation dynamics consists of 4 kinds of additive terms. 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 behavior's activations are noticeable, in which case α_1 is pulled to 0. The second term pulls α_1 to 0 when a is reached. Some noise is added to avoid deadlocks. Finally, "operator-coupling terms" influence the activation dynamics to enable plan execution. They will be explained in section 4.

The remaining behaviors shall be described more briefly. **approach**$_b$ and **approach**$_c$ are analogous to **approach**$_a$. The target and activation dynamics for behavior #4, **roam** are similar to those of **approach**$_a$—in fact, the equations are those of fig. 2 with fresh time constants and φ_a, δ_a set to 0. As a result, while **roam** is active (i.e., while α_4 is big), the robot would drive forward with default speed 1/2, slow down before and turn away from obstacles, and resume standard forward motion in a new direction. **roam**'s activation dynamics basically says, "as long as no other behavior is active, **roam**", making it the fallback behavior.

```
(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 3: Types, initial situation, and final situation for the errand domain.

```
(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 4: GRAPHPLAN errand domain operators.

The target "dynamics" of **gather** and **dump** are constants, setting their respective g or d variables to 1 and all other d and g variables to 0. The activation dynamics are a bit more involved. The rationale is that after, e.g., spotting a book (i.e., after γ_{book} jumps to 1), **dump**$_{book}$ becomes active for some fixed time interval, after which **gather**$_{book}$ gets active in another time interval. Implementing such a timing mechanism as triggered by γ_{book} is mostly technical, and we skip it here for brevity.

3 ERRANDS DOMAIN PLANS

We now very briefly look at the errands domain from the planning side, starting with its representation for the planner. We chose GRAPHPLAN[1] to be that planner, the rationale being: We wanted to use a standard planner and to keep simple the planning representation and the planning process for investigation and explanation at the present state of work. Having a richer representation language, such as ADL [8], and a planner that is able to handle it might be helpful in the long run, and it is not excluded by the framework that we are presenting. However, this issue has not yet been studied in depth.

Fig. 3 gives the domain signature, start situation and goal conditions as appropriate for GRAPHPLAN. The original robot location is a; the goal is to have both the *book* and the *mug* at a. We have three operators GOTO, LOAD, and UNLOAD, all with the intuitive interpretation. Fig. 4 defines them in GRAPHPLAN syntax. A plan for solving the resulting planning problem, which GRAPHPLAN finds in milliseconds, is given in fig. 5.

```
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 5: A plan for solving the errands problem in GRAPHPLAN's output format. Numbers before operators specify execution order; operators with equal numbers may be executed in either order.

Planning operators may, but need not correspond directly to behaviors 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 behaviors need not mention the current or past locations. In more complex domains, the planner may work with operators that correspond to higher-level behaviors, or it may work with macros that get expanded before plan execution into elementary operators corresponding to behaviors 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 behaviors in the technical sense explained next.

4 COUPLING SYMBOLS AND DYNAMICS

4.1 From Symbols to Dynamics

Operators, which are symbolic entities, have to be executed by the DD system, which is dynamic. We assume a plan execution monitor on the symbolic side picks a single operator for execution at every point of time. Usually, this is one of the operators that are next in the current plan according to its execution order. Alternatively, a user may give the robot single operators to execute.

As will become apparent, "execute" is a misleading term, as the DD system is not strictly commanded to carry out an action. Rather, it is biased more or less strongly in its ongoing activity towards achieving the operator's effects. Assume, e.g., that GOTO(L, a) for some location L is next for execution. That does not mean the robot should rush toward a regardless of circumstances. If, e.g., its battery is low, it should recharge first. Calling GOTO(L, a) should result in a general, persistent tendency for the robot to proceed toward a, yet leave some freedom to do other things if circumstances so require. If a is reached, the plan execution monitor will notice it (by the mechanism to be described in sec. 4.2) and pick the next operator. As will be clear from the description given next, the monitor may abandon execution of an operator prior to termination; this possibility is not further discussed here.

This "operator-oriented biasing" of the DD system is effected by the *operator-coupling terms* (OCTs). In every behavior's activation dynamics, there is an OCT for the operator GOTO(L, a). These OCTs should result in the desired persistent tendency to move toward a. Fig. 6 presents suitable OCTs for GOTO(L, a). As visible from these examples, OCTs have a simple common format:

approach$_a$	$\dot{\alpha_1} = \ldots + s_{\mathrm{GOTO}(L,\mathrm{a})} c^1_{\mathrm{GOTO}(L,\mathrm{a})}(1-\alpha_1)$
roam	$\dot{\alpha_4} = \ldots + s_{\mathrm{GOTO}(L,\mathrm{a})} c^4_{\mathrm{GOTO}(L,\mathrm{a})}(1-\alpha_4)$
all others	$\dot{\alpha_i} = \ldots + s_{\mathrm{GOTO}(L,\mathrm{a})} c^i_{\mathrm{GOTO}(L,\mathrm{a})}(0-\alpha_i)$

Figure 6: OCTs for GOTO(L, a).

$$\dot{\alpha_j} = \ldots + s_{\mathrm{OP}} c^j_{\mathrm{OP}}(Z - \alpha_j),$$

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

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

Returning to fig. 6, we now see that when GOTO(L,a) is called, **approach**$_a$ and **roam** are encouraged with strengths $c^1_{\mathrm{GOTO}(L,\mathrm{a})}$ and $c^4_{\mathrm{GOTO}(L,\mathrm{a})}$, respectively. Since **approach**$_a$ directly does what the operator is intended for, the encouragment should be strong. It seems reasonable to also encourage **roam** a bit, just in case **approach**$_a$ does not get active for some reason. The other behaviors do not apparently contribute to (or even contradict) the operator's intention. Therefore, the corresponding OCTs feature $Z = 0$.

Using this metaphor of dis- and encouragment, and the mechanism of OCTs, plausible OCTs for the other operators used in our errand scenario are straightforward. E.g., an operator LOAD($book$,L) should encourage the behavior **gather**$_{book}$ rather strongly, discourage **dump**$_{book}$ strongly, and discourage mildly all other behaviors.

By transforming operators into OCTs, we do not "control" action by plans, nor do we oversoo all eventualities and account for them. We believe, though, that this is all right for the types of robots and worlds that we consider—control cannot be enforced in environments that are fundamentally unknowable and unpredictable.

4.2 From Dynamics to Symbols

We now turn to closing the gap between the information available on the DD side and the information required on the symbolic side. In this paper, we can only sketch the main features of our approach.

We require that every perception be active. There are three sources of information about the world. First, there is information from past action; this is described below in some detail. Second, information in symbolic form can be communicated from other agents, such as fellow robots or human users. And third, information in the required format may result from the activation of particular information-seeking behaviors; the complexity of these behaviors may vary depending on how "directly" the proposition in question can be mapped to the robot's sensor configuration. E.g., we have assumed above that proximity to *a, b, c* can be sensed directly (variables $\delta_a, \delta_b, \delta_c$), which means we have in fact assumed that the proposition $\mathsf{At}(robot, a)$ could be directly mapped to a sensor request; other propositions may be impossible to be sensed that directly. We do not go into detail with regard to the latter two information sources, but concentrate on drawing information from past action as a way of information gathering that is special for our DD context.

The history of the activation variables α_i contains the essence of what the robot did. For instance, $\mathsf{At}(mug, b)$ holds currently if $\textbf{approach}_b$ was recently active and uninterrupted, followed by $\textbf{dump}_{mug}$. There is a caveat, though. Since the activation dynamics of a behavior 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 source of information from which we permanently update the world model, we take the past history of all α_i and all s_{OP}. (The other two ways of acquiring information may be used in addition.)

Let us give an example. The errands domain involves just two types of propostions: For objects O and locations L, these are $\mathsf{Has\text{-}Robot}(O)$ and $\mathsf{At}(O, L)$; $\mathsf{At}(robot, L)$ is a special instance of At. To start with $\mathsf{Has\text{-}Robot}(book)$, the only way of having the book on board is having **gather**ed it in the past, and not **dump**ed it later. Writing $\alpha_i(t)$ for α_i in the past time point t, we have

$$\mathsf{Has\text{-}Robot}(book) \leftarrow \\ \exists t_1 . \alpha_5(t_1) \approx 1 \wedge \forall t_2 . \; [t_2 > t_1 \rightarrow \alpha_7(t_2) \not\approx 1]$$

$\mathsf{At}(robot, a)$ would be somewhat clumsier. We have noted above that it could be mapped to a simple sensing behavior that checks δ_a. But there is a way to derive it from past action, if so required. Let Δ be a constant denoting an upper limit of the duration that it takes to deactivate $\textbf{approach}_a$. Then

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

Let us explain this line by line: t_1 was less than Δ before t_2; $\textbf{approach}_a$ was active in t_1, and inactive in t_2; none of the behaviors that would have destroyed $\mathsf{At}(robot, a)$, i.e., none of behaviors 2, 3, and 4, was active after t_2; and the robot was determined to go to a within the critical interval $[t_1, t_2]$, i.e., $\textbf{approach}_a$ has not been switched off by chance, such as by spotting a book to be **gather**ed.

Before defining $\mathsf{At}(O, L)$, notice that we cannot—nor want to—guarantee that the robot knows the locations of all objects, not even of all those that it has dumped before and not touched later. If such a **dump** did not take place at one of the locations *a, b, c*, then the object is simply lost. (The robot may find it 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 $\mathsf{At}(robot, L)$, within a small, user-defined time window Δ' after deactivation of $\textbf{approach}_L$, 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 any 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 activation history all the time.

5 CONCLUSION

At the present state of work (and within the limits of this paper), many issues remain unaddressed and many problems unsolved. The first is to implement on a real robot the concept that we have described, work being underway on a RWI B-14. Among the other issues are: choice of different planners and planning domain representations; a plan execution scheme that could make use of serendipituous events by encouraging "future" behaviors to facilitate jumping ahead opportunistically in plan execution; screening the world information available to prevent overloading the planner with irrelevant detail; and analyzing past activation sequences to find regularities that might be exploited for planning or in the domain representation.

The contribution of this work can be seen from two perspectives. From the behavior-based action point of view, it offers a way of sequencing behaviors to achieve longer-term goal-directedness, yet keeping intact the basic ideas and principles of behavior-basedness. The current plan is used as an important source of information—no

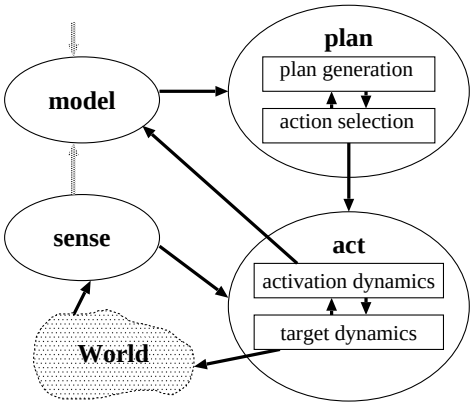

Figure 7: Sketch of the framework presented here. Ellipses are borrowed from the classical SMPA robot control framework [3]; boxes are new ingredients. Arrows denote influence between components; an arrow to an ellipse influences all sub-boxes. Light arrows depict model update by direct sensing and by symbolic information.

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 [10] to work.

From the planning point of view, our work contributes a principled way of coupling plans and dynamical action. It has been obvious from the beginnings of planning research that operators must be "implemented" in terms of executable procedures. Plans intended for helping humans organize their work 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. 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 that couple plans successfully with physical action (e.g., [7, 2]), it seems that feeding back information about the world into the planning representation is still a problem. 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.

As a new feature of our framework, we allow the world to be perceived through the history of activation values. This does respect the sensor data, but sees it indirectly as 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 behaviors. As a result, the ingredients of the naive *sense-model-plan-act* (SMPA) framework [3] for robot control are assembled in a different way, see fig. 7. Conceptually, two circles of activity concur, with the mutual influence as described above: sensorimotor coupling as by the DD behaviors alone (World-sense-act circle), and deliberation (model-plan-act circle). We are not aware of a like approach. It seems to have some potential for merging in a principled way the best of the behavior-based and the plan-based world for controlling autonomous robots.

ACKNOWLEDGEMENTS

While participating in this work, Morignot was an ERCIM postdoctoral research fellow at GMD. Hertzberg takes part in the European Commission's TMR network VIRGO, grant ERBFMRXCT960049.

REFERENCES

[1] A. Blum and M. Furst. Fast planning through plan graph analysis. *Artif. Intell.*, 90:281–300, 1997.

[2] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *J. Expt. Theor. Artif. Intell.*, 9:237–256, 1997.

[3] R. A. Brooks. Intelligence without reason. A.I. Memo 1293, MIT AI Lab, 1991.

[4] H. Jaeger. The dual dynamics design scheme for behavior-based robots: A tutorial. Arbeitspapier 966, GMD, Jan. 1996.

[5] H. Jaeger. Multifunctionality: A fundamental property of behavior mechanisms based on dynamical systems. In *Proc. SAB-98*, 1998. in press.

[6] H. Jaeger and T. Christaller. Dual dynamics: Designing behavior systems for autonomous robots. In S. Fujimura and M. Sugisaka, editors, *Proc. Int. Symposium on Artificial Life and Robotics (AROB'97)*, pages 76–79, 1997.

[7] N. Nilsson. Shakey the robot. Technical Report TN 323, SRI International, Apr. 1984.

[8] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. Int. Conf. on Principles of Knowledge Representation (KR-89)*, pages 324–332, 1989.

[9] R. Pfeifer and C. Scheier. *An Introduction to New Artificial Intelligence*. MIT Press, 1997.

[10] M. Pollack. The uses of plans. *Artif. Intell.*, 57(1):43–68, 1992.

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

[12] E. Schlottmann, D. Spenneberg, T. Höpfner, and T. Christaller. Die Programmierumgebung für den Kurs D2 auf der IK-97 – Navigation mobiler Roboter. Arbeitspapier 1081, GMD, 1997.

[13] D. Spenneberg, E. Schlottmann, T. Höpfner, and T. Christaller. PDL programming manual. Arbeitspapier 1082, GMD, 1997.

[14] L. Steels. The PDL reference manual. Memo 92-5, Free University, AI Lab, Brussels, 1992.