MINDER1: An implementation of a protoemotional agent architecture

Ian Wright
Aaron Sloman

School of Computer Science,
The University of Birmingham
Edgbaston, Birmingham,
B15 2TT, UK
I.P.Wright@cs.bham.ac.uk,
A.Sloman@cs.bham.ac.uk

December 9, 1996

Abstract

An implementation of an autonomous resource-bound agent able to operate in a simulated dynamic and complex domain is described. The agent, called MINDER1, is a partial realisation of an architecture for motive processing and attention. It is shown that a global processing state, called *perturbance*, can emerge from interactions of subcomponents of the architecture. Perturbant states are characteristic features of many states that are commonly called emotional. The agent is compared to other computer simulations of emotional phenomena.

Contents

1 Introduction

There are many approaches to the study of emotions. We favour a *design-based* approach (Sloman, 1994) that takes the stance of an engineer attempting to build, and understand the design options for, a system that exhibits the phenomena to be explained. We move from abstract requirements for functioning agents to possible designs that satisfy those requirements. This is an exploration of *niche-space* and *design-space* and the mappings between them (Sloman, 1995a). Our interest focuses on *complete* or 'broad but shallow' agents (Bates, Loyall & Reilly, 1991) that combine many capabilities (such as perception, planning, goal management and action) but where each capability is initially implemented in a simplified fashion. The interaction of sub-components of an architecture can give rise to emergent phenomena, some of which we believe are important in an understanding of the emotions.

In this paper we describe an implemented architecture that partially supports a processing state called *perturbance*, which is a partial or total loss of control of attention. The state is related to existing information processing theories of emotion and existing computational models. We claim that perturbances will and do exist in resource-bound architectures that attempt to meet a requirement for managing multiple motives in complex and dynamic domains. The implementation demonstrates this for our particular agent design and simulated world, points to futher extensions of the work, and illuminates one of the characteristic features of human emotional states.

2 A design for motive processing

The requirements and high level design for the agent architecture that informs all our work has been described elsewhere, notably in (Beaudoin, 1994), summarised in (Beaudoin & Sloman, 1993; Sloman, Beaudoin & Wright, 1994; Wright, Sloman & Beaudoin, 1996), and first elaborated in (Sloman, 1978; Sloman, 1985). It is partly inspired by Georgeff's Procedural Reasoning System (Georgeff & Ingrand, 1989; Georgeff & Lansky, 1989; Rao & Georgeff, 1991b; Rao & Georgeff, 1991a).

The discussion of the design is not repeated here. However, the main design points to keep in mind are: (i) the architecture is multi-layered allowing coarse-grained asynchronous parallelism, (ii) there is a reactive layer that includes 'automatic' processes all implemented in highly parallel dedicated, but trainable, 'hardware'. A major function of the reactive layer is to generate motives (although motives may be generated by non-reactive processing). (iii) A variable threshold attention filter only allows motives of sufficient *insistence* (Sloman, 1987) to surface and be candidates for higher level, management processing. Insistence is a 'quick and dirty' heuristic measure of the urgency and importance of a motive, and is a dispositional ability of a motive to gain management resources and attract attention. The mechanisms assigning insistence values must work using simple 'heuristic' measures because computing accurate measures of urgency and importance could be too slow and computationally expensive, possibly diverting the very higher level processes the filter mechanism is designed to protect. However, insistence measures based on fallible heuristics can sometimes cause 'bad' decisions about what should and should not enter management processing. (iv) A management layer involves processes that, among other things, decide whether new motives should be adopted or not. Management processes, but not the low level processes, can create, consider and evaluate explicit representations of options before selecting between them, for deliberation, planning and problem-solving. (v) A metamanagement layer involves goal-directed processes that refer either to management processes or metamanagement processes, such as deciding whether to decide to adopt a goal, or raising the motive filter threshold level if management processes are too 'busy' and so forth. More sophisticated metamanagement can include evaluation of management activities and the development of new deliberative strategies.

We have acknowledged that the design sketch is speculative, vague and subject to revision in the light of implementation problems or empirical evidence (Wright, Sloman & Beaudoin, 1996). It is speculative in that empirical checking has not yet been attempted and may be very difficult. Nevertheless we think the whole architecture is in principle implementable using current artificial intelligence (AI) techniques, including neural networks and other 'sub-symbolic' mechanisms where appropriate, and that iterative 'broad but shallow' design based work of this kind is needed to understand the full complexity of emotional states and other 'high level' mental states and processes in human-like systems, including moods, attitudes, personality and so forth. The following section reports a partial implementation of the processing of motives based on our design.

3 An implementation of motive processing

This section describes (i) a toolkit that allows rapid prototyping of agent designs, (ii) a simulated microworld domain that serves as a demanding testbed for our agent architecture, and (iii) the implementation itself, including the capabilities and behaviour of the agent, and its three processing layers (reactive, management and metamanagement).

3.1 The SIM_AGENT toolkit

For our work exploring architectural design requirements for intelligent human-like agents, and other kinds, we need a facility for rapidly implementing and testing out different agent architectures. The SIM_AGENT toolkit (Sloman & Poli, 1995; Sloman, 1995e; Sloman, 1995d), freely available on the internet, is designed to make it relatively easy to implement a collection of interacting objects and agents, where each object (or agent) has internal complexity represented as sets of concurrent interacting condition-action rules that can invoke additional mechanisms of any kind, including ordinary procedures, neural nets, theorem provers, databases, genetic algorithms, and so forth.

The toolkit has two main components: Poprulebase (Sloman, 1995b) and the SIM_AGENT library (Sloman, 1995d; Sloman, 1995e). Poprulebase is a sophisticated and very general interpreter for condition-action rules, written in Pop-11. It is a forward-chaining production system interpreter, but provides a collection of unusual facilities, including a smooth interface to neural net or other 'sub-symbolic' mechanisms, mechanisms for control to be transferred between collections of rules as the context changes (allowing SOAR-like (Laird, Newell & Rosenbloom, 1987) pushing and popping of contexts), and facilities for altering the allocation of processing resources to different rulesets (Sloman, 1995c).

The SIM_AGENT library provides a set of base classes and scheduling mechanisms for running simulations involving a number of objects and agents whose internal mechanisms are implemented using Poprulebase. The scheduler simulates parallelism between agents and between subcomponents of agents. SIM_AGENT makes use of the object oriented facilities provided in the Pop-11 Objectclass package, a CLOS-like extension to Pop-11 providing multiple-inheritance, generic functions and so forth. Objects allow the production of re-usable, extendable software modules and shareable libraries. The agent and domain we are about to describe have been implemented using the toolkit[1].

3.2 The minder domain

We have chosen a domain that imposes many tasks on the agent requiring the management of multiple motives and the control of attention. The domain is analogous to a nursemaid or minder in charge of a collection of babies (see (Beaudoin & Sloman, 1993))[2], but highly simplified in order to avoid the need to solve all the problems of AI, including 3-D vision, motor control, and naive physics. Simplification allows us to address motive processing while avoiding problems best left to others. The 'nursery' can take various forms. In the form used here it is a two-dimensional room that contains a collection of simple reactive agents, called 'minibots' or dependents, which wander around getting into various difficulties, such as running out of 'charge' or falling into ditches. The reactive agents are dependents as they require the assistance of a minder agent whose task is to 'care' for them. For example, the minder agent can pick up dependents and carry them to a safe distance from a ditch. The wandering of the dependents ensures that new minder motivators are continually generated. Hence, the minder needs to arbitrate between many motives while maintaining reactivity to new, possibly urgent and important events. The domain could easily be extended in several directions. Figure 1 is a screenshot of a simple graphical representation of the nursery that is used for viewing the agent's external behaviour and debugging purposes.

3.3 Behaviour and capabilities

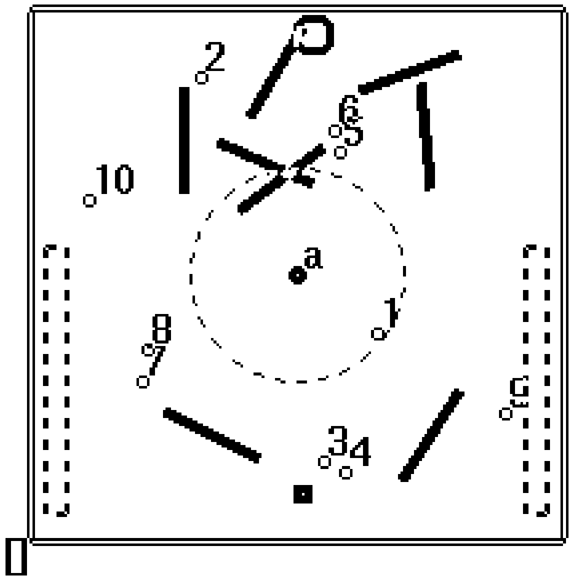
For ease of reference this particular implementation of the design is named MINDER1. The name is not an acronym.

Figure 2 shows MINDER1 constructing an enclosure used to keep minibots at the north end of the nursery to prevent them from wandering into ditches. MINDER1's behaviour can be observed in real-time but with occasional brief halts as pop-11 collects accumulated 'garbage' (old data that can be destroyed to free up memory). A separate window contains the textual trace output of the simulation. During a normal run the trace monitors the state transitions of all motives. However, a collection of debug flags can be altered so that MINDER1's current beliefs, sense data, motives, other knowledge stores, and internal processing may be examined at runtime.

MINDER1 'scurries' around the nursery pursuing its various motives. For example, it may sense that a dependent needs recharging and move towards it to pick it up. Also, MINDER1 often replaces its current motive with a new motive. For example, MINDER1 may be taking a dependent to the recharge point but 'notices' that another dependent is about to fall into a ditch. MINDER1 drops the minibot it is holding, saves the other minibot from falling into the ditch, and then returns to the recharge task. If the original minibot has moved MINDER1 may then search

[1] Other computational experiments with the toolkit are described in (Davis, Sloman & Poli, 1995; Davis, 1996). See http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html for mpeg movies of some of the experiments, including the minder domain.

[2] The idea for movable and rotatable bars is borrowed from Nilsson's botworld domain (Nilsson, 1994; Benson & Nilsson, 1995).



Object	Capabilities	Graphic
Minder agent	See section 3.3	Dark circle labelled "a"
Dependent agents	Erratic roaming	Numbered light circles
Fences	Movable	Dark bars
Recharge point	Refuels dependents	Small square
Dismissal point	Removes dependents	Large rounded square
Ditches	Damages dependents	Dashed rectangles
Walls	Encloses 2-d surface	Bounding box

Figure 1: **The microworld domain**

for it in the immediate vicinity. Minibots frequently fall into ditches and become damaged because MINDER1's perceptual field and speed of movement is limited. If there are no other pressing motives then MINDER1 may take the damaged minibot to the dismissal point. There are many other kinds of motive and corresponding behaviours that could be described (see appendix 8.1 for a full list).

Informally, MINDER1 seems to prioritise its tasks in an reasonably intelligent manner: it successfully acts in the nursery domain to achieve its motives. However, as the number of minibots increases performance deteriorates. MINDER1 can manage multiple motives, dynamically rescheduling them where necessary. Its resource limited management processes are protected from too high a processing load and interruption by a filter threshold that can vary according to the number and insistence level of motives. However, there are limitations, discussed in section 3.7.2. The architecture is evaluated in section 3.7.1.

The following three subsections describe how this functionality is achieved.

3.4 The reactive layer

Purely reactive processes cannot explicitly construct representations of alternative possible options and select between them: they have reflex-like or 'ballistic' causality. If conflicting actions are generated simple weight combinations or winner-take-

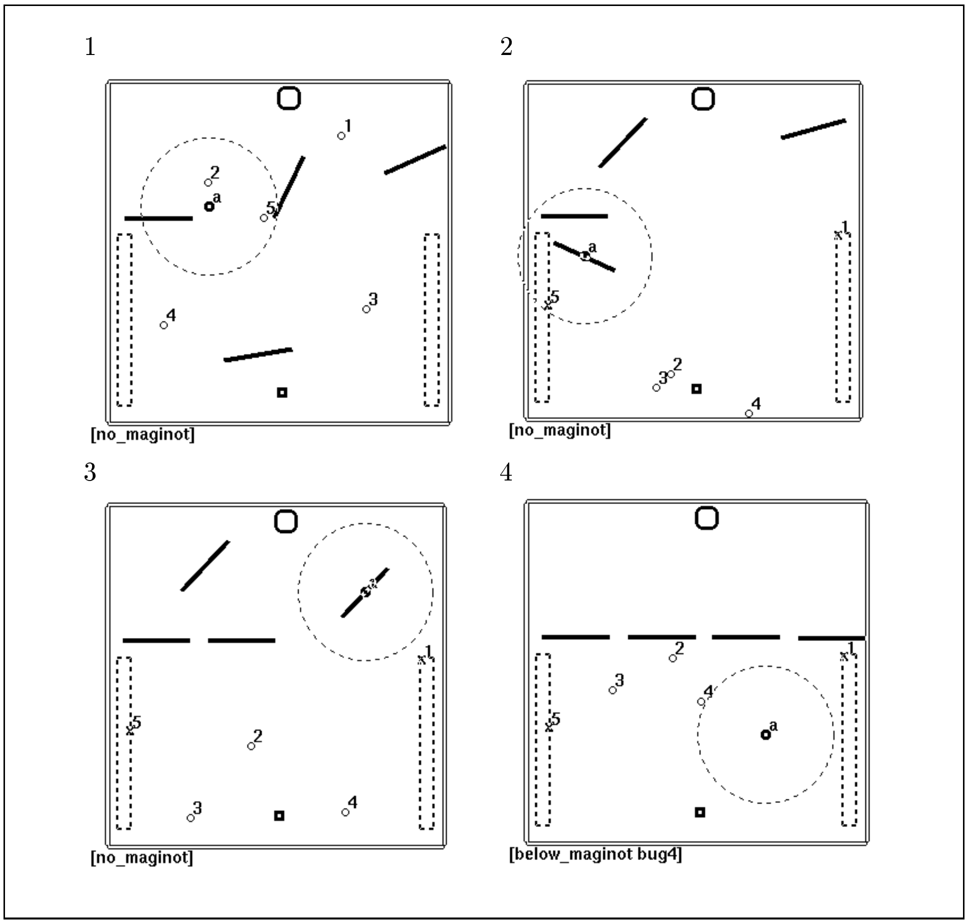


Figure 2: **Stages in the construction of an enclosure**. Note: text under window is the descriptor of the currently active motive.

all mechanisms determine the outcome. The reactive subsystems in MINDER1 are (a) perception, (b) belief maintenance, (c) reactive plan execution, and (d) preattentive motive generation. Each is described in turn.

3.4.1 Shallow perception

MINDER1's perceptual subsystem is implemented in a very shallow manner: in each time slice the required, externally visible features of domain objects are converted into a representation that is stored in an internal database, which, at any time, will have only partial and possibly incorrect information about the environment. The sampling range is limited to a fixed radius (see dashed circle in figure 1). Any object within range can be sensed but, unlike vision, the occlusion of objects is not supported. The perception rulesets are provided with sufficient processing cycles to perform all necessary domain sampling within each time slice. An example *new_sense_datum* is the following:

```
[new_sense_datum
    time 64 name minibot4 type minibot status alive distance 5.2 x 7.43782
    y 12.4632 id 4 charge 73 held false]
```

The data structure describes a dependent *minibot4* that was sensed at time 64, which is alive, has charge 73, is not held by another agent, and is at distance 5.2 units from MINDER1. The sensing of other objects is essentially similar, except the sensing of fences, which includes information about orientation and size.

3.4.2 Shallow belief maintenance

Beliefs, compared to sense data in this system, are more complex structures. They represent states-of-affairs that, due to the dynamism of the domain and the limits of perception, may or may not hold. There are two kinds of belief: sensory-based beliefs, which contain information about objects in the environment and are constructed from *new_sense_datum* items, and agent-based beliefs, which are generated by agent actions, such as beliefs about fences serving as components of an enclosure (see figure 2).

At any particular moment many of MINDER1's beliefs will be wrong. For example, it may believe that *minibot4* is in the absolute coordinate location (80,90). However, location (80,90) may be currently out of sensor range and *minibot4* may have moved. A design problem arises when MINDER1 returns to a location within sensor range of location (80,90). If *minibot4* is sensed the corresponding belief is updated; however, if *minibot4* is absent then no *new_sense_datum* will update the belief: it is not possible to sense an absence. Without rudimentary belief maintenance MINDER1 will continue to hold a false belief despite having sufficient information to infer its falsity. The problem is solved by storing defeaters with each belief. Defeaters are conditions that must remain false in order for the belief to remain true. If the defeating conditions for a particular belief evaluate to true then the belief is removed. An example *belief* is the following (note that double equals matches any sequence of items in a list):

```
[belief time 20 name minibot8 type minibot status alive distance 17.2196
    x 82.2426 y 61.2426 id 8 charge 88 held false
    [defeater
        [[belief == name minibot8 == x ?Xb y ?Yb ==]
        [WHERE distance(myself.location, Xb,Yb) < sensor_range]
        [NOT new_sense_datum == name minibot8 ==]]]]
```

The defeater is composed of poprulebase conditions, and says, 'IF I have a belief about minibot8 AND I have no new sense data about minibot 8 AND I am in a position that, according to my belief, I should have new sense data about minibot 8 THEN my belief is false'. The defeater mechanism allows arbitrary size hierarchies of defeats to be formed. For example, a 'second order' belief may be justified by a 'first order' belief. If the defeater of the first order belief evaluates to true then the second order belief is also removed. In this way beliefs are automatically maintained, although it should be noted that belief maintenance is a difficult problem to solve in general (Logan, 1996). Also, the strategy adopted here could become computationally expensive (consider deleting a long chain of n-order beliefs). If belief maintenance is to be 'deepened' a more efficient strategy is needed.

3.4.3 Reactive plan execution

MINDER1 needs to be able to act in the domain. As the nursery is changing continually it is fruitless for MINDER1 to attempt to construct and execute precise but inflexible plans that depend on beliefs that can rapidly become obsolete. Instead, MINDER1 needs a level of plan execution that is robust, that is can recover from unexpected failures, and reactive, that is can immediately react to new contingencies in the course of plan execution without the need to engage higher level, resource limited systems. For example, MINDER1 may have a plan to move to location (50,50). The execution of this plan will involve many steps, including moving forward, rotating, sensing the route ahead, planning routes around obstacles and so forth. It is possible that obstacles, such as fences, can be moved by other agents; therefore, a purely classical planning approach, in which a complete plan is constructed prior to action and then blindly followed in detail, is likely to fail. Plans that can alter themselves to achieve their goals via continuous feedback from the current situation avoid this difficulty. An approach that partially meets this requirement is Nilsson's *teleo-reactive (TR) program* formalism (Nilsson, 1994). We have adopted it here, and will briefly describe it.

A TR program is an ordered set of production rules that directs the agent toward a goal in a manner that takes into account changing environmental circumstances (Nilsson, 1994):

$$\begin{aligned} K_1 &\longrightarrow a_1 \\ K_2 &\longrightarrow a_2 \\ &\dots \\ K_n &\longrightarrow a_n \end{aligned}$$

K_i are conditions on agent knowledge (including information items such as *beliefs*, *new_sense_datum*, representations of current motives etc.) and a_i are actions on the world or on beliefs. On *every* cycle the conditions of all active TR programs are evaluated from top to bottom (simulating the presumably parallel implementation of a reactive subsystem). The a_i of the first K_i that evaluates to true is executed. If, on the next cycle, the same K_i evaluates to true then the same action is executed, and so on until the conditions change. A TR program must satisfy a regression property, which, informally, states that an action, a_i, will eventually achieve a condition, K_j, which is higher in the list ($j < i$). TR programs can call other TR programs or themselves, that is they can be hierarchic and recursive. Figure 3 provides an example TR program implemented in MINDER1.

K_1: ˜(held(obj)) ∧ charged(obj) ⟶ a_1: null
K_2: held(obj) ∧ charged(obj) ⟶ a_2: DROP(obj)]
K_3: held(obj) ∧ close_enough(obj, recharge_point) ⟶ a_3: CHARGE(obj)
K_4: T ⟶ a_4: TAKE_OBJECT(obj, recharge_point)

Predicate	**Semantics**
held(obj)	T if agent holds object obj
charged(obj)	T if agent believes obj has sufficient charge
close_enough(obj1, obj2)	T if agent believes obj1 and obj2 are adjacent
Imperative	**Semantics**
DROP(obj)	Agent attempts to drop obj, makes hold(obj) false
CHARGE(obj)	Agent charges obj at recharge_point, makes charged(obj) true
TAKE_OBJECT(obj1, obj2)	Agent moves obj1 to obj2 by invoking another TR program

Figure 3: **TR program charge_object**

The TR program is called by a a higher level executor (see section 3.5) that unifies a parameter value with *obj*, such as the value *minibot4*. In this example, the TAKE_OBJECT action is a call to further, more complex, TR programs that can search for objects, home in on their location, pick them up, move while avoiding obstacles, and so forth. Even this simple program can deal with unexpected failures: for example, if, for whatever reason, the obj is no longer held the TR program will reactively 'drop down' to condition K_4 and attempt to relocate and hold the obj.

MINDER1 currently has thirteen TR programs (see appendix, section 8.4) that serve as reactive behavioural building blocks. Each TR program is implemented as a set of SIM_AGENT production rules satisfying the regression property. As TR programs are fully evaluated each time step it is helpful to think of their semantics in terms of dedicated circuits that continuously evaluate feedback from actions (Nilsson, 1994)[3].

3.4.4 Generactivation of motivators

For MINDER1 to use its TR programs it requires goals to achieve. The source of motives in MINDER1 is a suite of generactivators that express the agent's concerns(Frijda, 1986). For example, a particular generactivator *G_low_charge* searches the internal database for beliefs about dependents with very low charge; if such a belief is found then the generactivator constructs a declarative representation of a motive and places it in a motive database[4]:

```
[MOTIVE motive [recharge minibot4] insistence 0.322 status sub]
```

The motive contains a motivational attitude towards a state of affairs in the domain ('make it true that minibot4 is recharged')[5], an insistence value, which, in MINDER1, is a heuristic, quantitative representation of the urgency and importance of the motive, and a current status, *sub*, which is a flag that states that the motive has not surfaced through the variable threshold attention filter (see appendix, section 8.1 for a full list of possible motives). Currently, the insistence heuristics have been built by hand. This is a simplification: we have avoided the need to design mechanisms that can construct insistence heuristics from domain interaction. The heuristics are functions that map conditions on agent knowledge to the reals in the interval [0,1]. For example, the generactivator that constructs motives when a dependent is too near a ditch uses the simple function *f(distance_from_ditch)* $\longrightarrow$ *insistence* to calculate insistence. The higher the number the higher the insistence and, consequently, the greater the motivator's dispositional powers to surface and grab management resources.

MINDER1 has eight generactivators expressing various concerns. A selection, described informally, are 'dependents must be fully charged', 'damaged dependents need to be removed from the nursery', 'the ditches need to be patrolled', 'an enclosure needs to be built to protect the dependents', 'ensure dependents are at safe

[3](Benson & Nilsson, 1995) describes further experiments with TR programs in a sophisticated agent architecture.

[4]Behaviour-based approaches to autonomous agency criticise the use of explicit representations of goals (e.g., see (Brooks, 1990; Brooks, 1991b; Brooks, 1991a; Agre & Chapman, 1987; Chapman, 1989; Chapman, 1990)). We believe that the behaviour-based approach in isolation is ultimately limited (Wright, 1994) and that declarative representations of goals are required for more sophisticated goal management, such as deciding whether to adopt goals, deciding when to schedule them, determining how important they are, and so forth. (Norman, 1994; Norman & Long, 1995) present arguments to show that agents with declarative representations of goals have the potential to be much more flexible than those that are purely reactive.

[5]Note, however, that the motivational attitude is not explicitly represented. The motivational attitude is implicit in the procedures that use the declarative representations of motives. In this case MINDER1 does not follow the specification for motives provided in (Beaudoin & Sloman, 1993; Beaudoin, 1994).

distances from ditches' and so forth. In addition to constructing motives, generactivators also remove motives when the original rationale for activation no longer holds, and dynamically alter insistence values, thereby reactivating motives to be candidates for surfacing.

The next section describes the resource limited management layer that takes explicit representations of multiple motives and translates them into intentions for action.

3.5 The management layer

The processes in the management layer are resource limited but can operate on explicit representations of alternative motives and select between them. Management processes pose design problems of great difficulty, and this is reflected in the shallowness of the management implementation. Discussion of the many limitations of the implementation is postponed to section 3.7.2 where the urgent need for a comprehensive theory of motive management is identified. The main management subsystems in this implementation are the motive filter mechanism, shallow motive management, and shallow plan execution. Each is described in turn. Figure 4 is a schematic representation of the architecture of and processes within MINDER1. The diagram is fully explained in this section.

3.5.1 Shallow motive filter mechanism

MINDER1's filter threshold is a real number in the interval [0,1]. (Beaudoin, 1994) considers more complex filter mechanisms and (Norman, 1996) considers a similar, but more developed, filter-based selective attention mechanism implemented in a factory domain. Newly generated motives are initially of state *sub*. Call the set of motives of state *sub*, M_{sub}. All members of M_{sub} are candidates for surfacing: if their insistence values are equal to or higher than the current filter threshold they surface and their state changes to *surfacing*; otherwise, their state remains the same. Figure 5 lists the variety of motive states represented in MINDER1 and conditions for the surfacing and 'diving' (a surfaced motive returning to state *sub*) of motives. Motives that remain members of the set M_{sub} for more than one cycle are not sufficiently insistent to gain management resources. However, this can change by generactivators recomputing insistence levels or by the lowering of the filter threshold (see section 3.6 on metamanagement). The filter acts to limit the number of motives that management processes need to consider, that is it functions as an 'attention' filter.

3.5.2 Shallow motive management

Before describing motive management processes a short note explaining figure 5 is required. The set M represents all MINDER1's current existing motives. There are various subsets of M, including the sets unsurfaced, M_{sub}, and surfaced, $M_{surfaced}$. The subsets of $M_{surfaced}$ represent motive state transitions that are possible within management. These state transitions are now explained.

(a) Deciding. All $m_i \in M_{surfacing}$ are *decided* by management processes. Deciding involves determining whether the motive should remain surfaced, that is whether management resources should continue to be devoted to it. A full implementation of deciding would need to include processes of motive assessment, such as developing sophisticated qualitative measurements of urgency and importance, determining the risks and benefits of adopting the motivator, and comparing these measurements

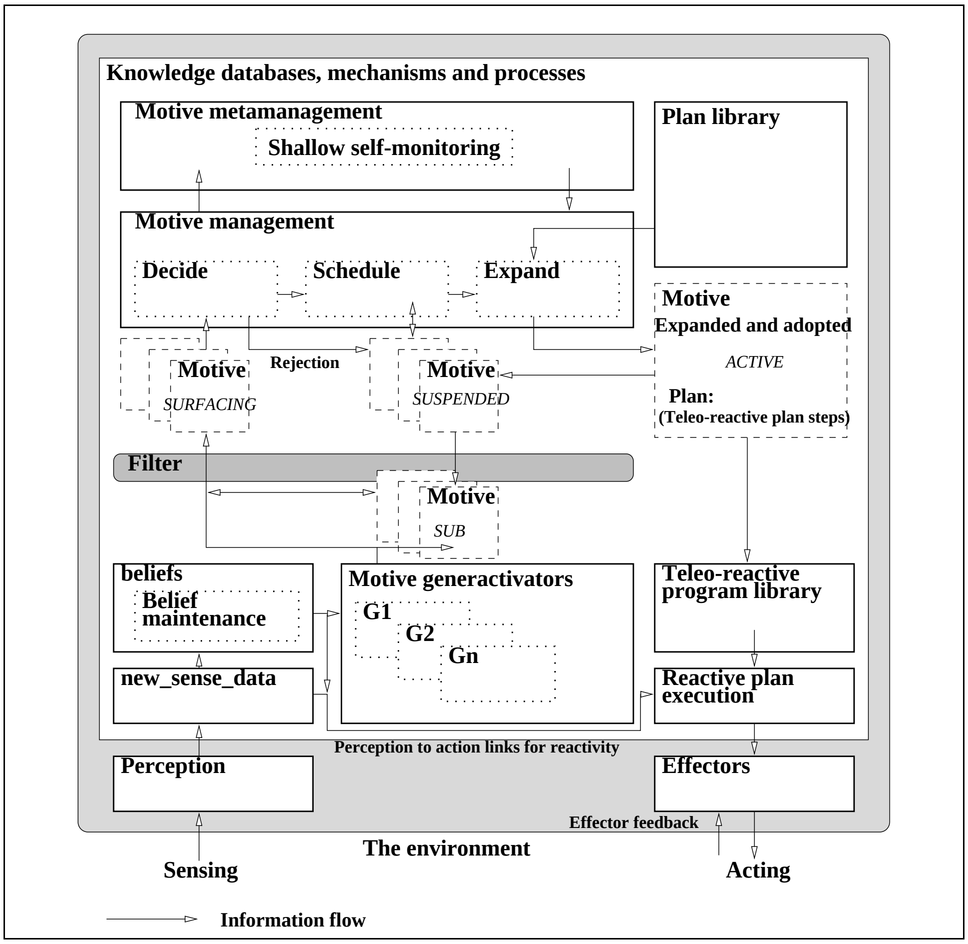


Figure 4: **The route from perception to action in MINDER1**. Solid lines represent mechanisms and databases, dotted lines represent processes that may occur within a mechanism, and dashed lines represent declarative data structures, such as motives.

with similar measurements of other motives. Often a motivator cannot be assessed until it has been partially expanded, or cannot be decided until it has been assessed, or cannot be assessed until partially executed, and so forth (Beaudoin, 1994). In other words, motive management systems that have purely linear motive state transitions are unlikely to meet the full requirements (compare (Sloman, 1978)).

MINDER1 has an extremely simple and shallow deciding process yet exhibits complicated interactions between deciding, scheduling, expanding and motivator states. The relations between these management processes are tangled but have been distinguished for the sake of exposition. For example, if a motive has already been scheduled, and is therefore active, it may be partially expanded in preparation for deciding. A motive such as:

```
[MOTIVE motive [save ditch1 minibot5] insistence 0.646361 status active]
```

is partially expanded to:

```
[MOTIVE motive [save ditch1 minibot5] insistence 0.646361 status active
```

Motive sets	Explanation of state
M	Current agent motives
M_{sub}	Unsurfaced motives
$M_{surfacing}$	Surfacing motives
$M_{surfaced}$	Surfaced motives
$M_{suspended}$	Surfaced but suspended motives
$M_{suspended,meta}$	Surfaced and suspended during meta-planning
$M_{suspended,execute}$	Surfaced and suspended during execution
M_{active}	Surfaced and adopted motives
$M_{active,meta}$	Surfaced and adopted for meta-planning
$M_{active,execute}$	Surfaced and adopted for execution

Where,

$M_{sub} \cup M_{surfaced} \equiv M$
$M_{surfacing} \cup M_{suspended} \cup M_{active} \equiv M_{surfaced}$
$M_{suspended,meta} \cup M_{suspended,execute} \equiv M_{suspended}$
$M_{active,meta} \cup M_{active,execute} \equiv M_{active}$
$length(M_{active}) = 1$

And,

if insistence($m_i \in M_{sub}$) $\geq$ filter_threshold then $m_i \in M_{surfacing}$;
if insistence($m_i \in M_{suspended}$) $\leq$ filter_threshold then $m_i \in M_{sub}$.

Figure 5: **Motive states and the transition between attentive and preattentive processing**

```
plan [[decide] [get_plan]]
trp [stop]
importance undef]
```

The partially expanded motivator, $m_i \in M_{active,meta}$, contains an initial metaplan with plan steps *decide* and *get_plan*. These plan steps are not external actions but calls to management processes. A metaplan is executed by the management system, whereas a normal plan is executed by the plan executor (see later). If the motive remains scheduled (i.e., another motivator has not displaced it as the active motive) the plan step *decide* is executed, invoking a decide routine stored in the plan library (see node 3 of figure 6, which is a graphical representation of management processes that occur on surfaced motives). Currently, MINDER1 has a single, shallow decide routine for all motivators, of whatever type. The decide routine determines the importance of the motivator, which for most motivators is the designator 'normal', meaning that the motivator's heuristic insistence level is held to be a good approximation of a developed measure of the importance of the motivator. However, no developed measure of importance ever occurs, and developed measures of urgency are not supported, which is a significant limitation of the implementation when compared to the complete design[6]. An important exception (node 4 in figure 6) to this is discussed in section 4. If the motivator still remains scheduled on subsequent cycles the next plan step, *get_plan*, is invoked. It has the effect of retrieving a stored plan from the plan library (see node 6). However, due to the parallelism of the MINDER1 architecture, new motives may surface at any time, be scheduled for immediate processing and replace the currently active motive. The replaced motive can be suspended either during a metaplan phase, or during an execution phase, that is be either $m_i \in M_{suspended,meta}$ or $m_i \in M_{suspended,execute}$ respectively. In both cases, the motive remains partially

[6]However, for many situations, heuristic urgency as represented by insistence is sufficient to order motives, such as choosing which of two minibots to rescue from falling into a ditch.

expanded to allow readoption at a later time.

(b) Scheduling. Scheduling involves determining when a motivator should become active, be executed and control current internal or external actions. (Contrast meta-scheduling, a metamanagement function, which schedules scheduling, that is determines *when* to consider a motive). A full implementation of scheduling processes would include developed measures of urgency that could answer questions such as: when will it be too late to satisfy the motivator? need it be satisfied at a particular time? can it be postponed? is it too early to do this? and so forth. MINDER1 bases its scheduling decisions on the insistence and importance of motivators. Therefore, urgency measures are only implicitly and heuristically represented. Nevertheless, the scheduling mechanism dynamically orders the set $M_{surfaced}$ such that a single motive, $m_i \in M_{active}$, is chosen to be activated for metaplanning, $m_i \in M_{active,meta}$, or execution, $m_i \in M_{active,execute}$, depending on its current expansion status. (Beaudoin, 1994) considers limited management parallelism that allows the adoption of more than one active motive, but MINDER1 does not support this desirable feature. Scheduling operates every management cycle; however, the filter mechanism ensures that the number of motives that needs to be considered is always low.

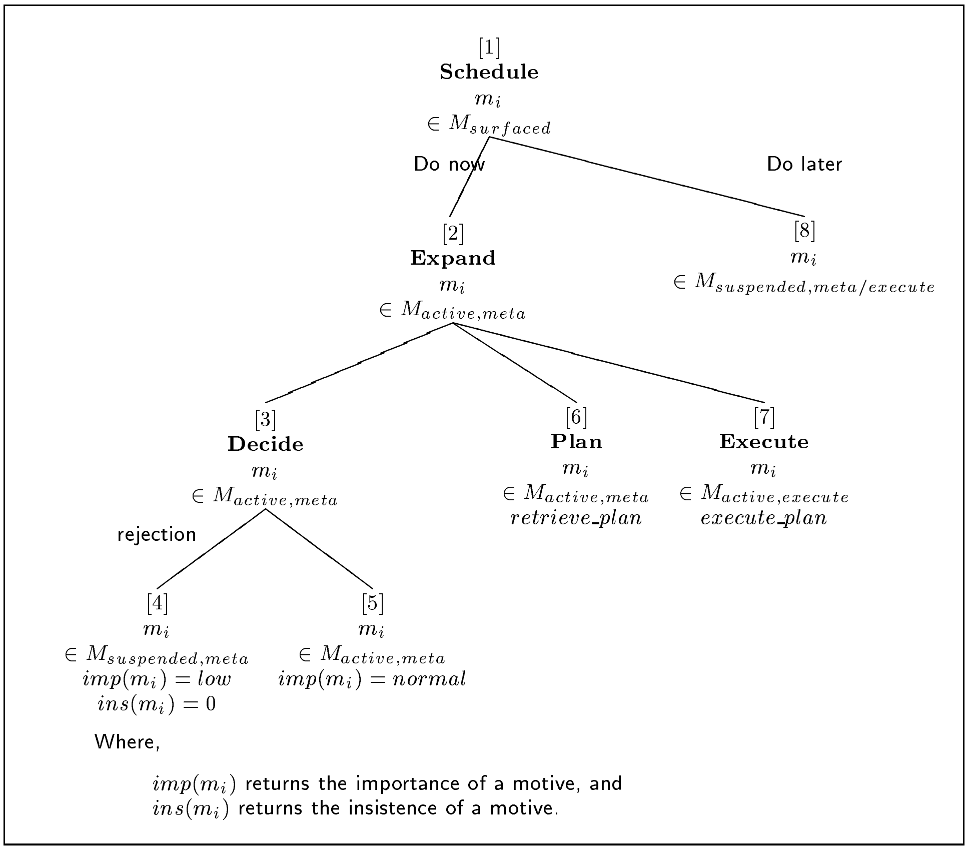


Figure 6: **Management processes on surfaced motives in MINDER1**

(c) Expanding. As stated, the metaplan step *get_plan* retrieves a stored plan from the plan library suited to the particular motive. Currently, MINDER1 has seven plans in its plan library (see appendix, section 8.3). In general, information contained in the motive is unified with plan variables. An example expanded motive to build an enclosure of fences is the following:

```
[MOTIVE motive [no_maginot] insistence 0.05 status suspended
    plan
        [[make_wall 40 60 0 second]
        [make_wall 65 60 0 third]
        [make_wall 90 60 0 fourth]]
    trp [stop]
    importance normal]
```

In this example the partially executed plan consists of three plan steps of the same type, *make_wall*. The plan step *make_wall* is itself a TR program that can be executed by the reactive plan executor. A more complete implementation of MINDER1 would include a planning mechanism that could construct new plans for new situations based on the agent's available action primitives. Planning capabilities would require storing STRIPS-style add and delete lists with both plans and primitives to allow reasoning about chains of behaviours. (Benson & Nilsson, 1995) describes an agent architecture that can learn pre and postconditions for TR programs from observations of the effects of its own behaviour and then dynamically construct novel reactive plans. Such flexibility would be a desirable extension to MINDER1's management processing. There are many planning algorithms in the AI literature, and incorporating a planner would not be difficult. However, this was not a major goal of the project.

3.5.3 Shallow plan execution

The plan steps of the currently adopted motive are executed as TR programs (see right-hand-side of figure 4). For example, the *make_wall* plan involves calls to further TR programs, such as *grab_wall* and *place_bar*. These TR programs themselves call other TR programs; some examples are *search*, which makes the agent search the nursery for a specified object if the agent has no beliefs concerning the object's location, *grab_object*, which makes the agent approach an object and pick it up, and *amble*, which moves the agent to a specified location while avoiding obstacles (see appendix, section 8.4). The leaves of TR program trees are atomic actions, such as *MOVE*, *ROTATE*, *SETSPEED*, *GRAB*, *ROTATE_BAR*, and so forth. Note however that the TR program tree is constructed as a complete circuit on every cycle. In other words, the links in the tree can dynamically change to allow unexpected contingencies to be catered for, such as an object being moved by another agent. Currently, MINDER1 has nine action primitives (see appendix, section 8.2). Building real robots requires much work to develop robust action primitives, for example (Marjanovic, Scassellati & Williamson, 1996) discusses designing primitives for robot arm pointing. Simulation work allows us to abstract from these engineering problems and concentrate on motive management.

This completes the description of the route from perception to action in MINDER1.

3.6 The metamanagement layer

MINDER1 has two metamanagement functions: changing the filter threshold level and detecting perturbant states. The discussion of perturbance detection is postponed until section 4. Metamanagement implementation is extremely shallow when compared to our design. A full implementation would include sophisticated 'self-monitoring' processes that detect, evaluate and control management processes(Wright, Sloman & Beaudoin, 1996).

It is assumed that management processing is resource limited. To reflect this we arbitrarily chose a maximum limit of three surfaced motives together with a maximum cycle limit for management rules. (A cycle limit defines how much processing resources are devoted to a ruleset in each time slice.) A metamanagement process monitors the number of surfaced motives. If the number of surfaced motives is more than three then the filter threshold level is incremented. The threshold level continues to increase every cycle until there are three or less surfaced motives. For example, if a $m_i \in M_{suspended}$ has an insistence level lower than the threshold it will 'dive' and return to status *sub*. Similarly, if there are fewer than three surfaced motives then the filter threshold is decremented. The process continues until the threshold reaches zero or a sufficient number of *sub* motives surface into management processing.

The joint operation of the dynamic filter and generactivators recomputing insistence values ensures there can be a continual movement of motives from preattentive to attentive processing and back again. For example, MINDER1 may have ten motives in total, four of which have surfaced, and one active. In this case the number of surfaced motives exceeds the maximum and the filter threshold is raised on each time step. The filter rises until the least insistent surfaced motive 'dives' and the processing load returns to a manageable level (see figure 7).

The original agent design (Beaudoin, 1994) did not specify a state transition of motive diving. Instead, suspended motives not attended to for some time decayed and were eventually removed. However, this is not entirely satisfactory as suspended motives, whether they be postponed for execution or deciding, will impose extra computational load on management processes. For example, to meet a requirement of mutual compatibility between motives there will need to be processes that consider the relations between all motives in the set $M_{active} \cup M_{suspended,execute}$. For example, motives that have been decided for execution, but are currently suspended, might be incompatible with newly surfaced, active motives (e.g., a person may intend to resume job hunting later in the day, but receives news that a friend is in hospital). Management processes would need to detect and resolve such incompatibilities. Detecting incompatibilities requires considering both M_{active} and $M_{suspended,execute}$. Therefore, if a subset of suspended motives can be removed from management the amount of computation required can be reduced. Allowing motives to dive, in addition to surface, based on heuristic measures of insistence, is a way to achieve this. However, the disadvantage is that the agent may fail to detect a serious conflict between a new action and a non-urgent but very important suspended motive. Therefore, allowing motives to dive could be disastrous. Hence, the implemented solution is not entirely satisfactory. A better solution would allow surfaced motives to be always accessible but use indexing mechanisms to overcome computational expense. But indexing may not be perfect, and relevance could still be missed.

There are many reasons why the filter level changes. For example, an active motive may be removed due to successful completion of its plan, or removed due to the loss of its rationale. The removal of a surfaced motive may stop the filter threshold being raised further and begin lowering it. The interactions between all these kinds of processes can be complicated. A simple illustration is provided below. In this short trace a new motive surfaces that causes metamanagement processes to detect the presence of too many surfaced motives, resulting in the filter being raised, an existing, surfaced motive to dive, and the activation and adoption of the new motive.

```
====================== end of cycle 82 ==================
====================== end of cycle 83 ==================
 ** [[Surfacing --
```

```
                [MOTIVE motive
                        [recharge minibot5]
                        insistence 0.21 status sub]]]
 ** [[RAISING filter threshold to 0.02]]
======================= end of cycle 84 ==================
 ** [[Diving --
             [MOTIVE motive
                     [default]
                     insistence 0.02 status suspended plan
                     [[decide] [get_plan]]
                     trp
                     [stop]
                     importance undef]]]
======================= end of cycle 85 ==================
 ** [[Activated --
                [MOTIVE motive
                        [recharge minibot5]
                        insistence 0.215 status active]]]
======================= end of cycle 86 ==================
```

A filter mechanism of this sort is connected to the folk-psychological concept of focus: a very high filter level would correspond to a high level of focus, during which attentive resources are concentrated on a single, very important and urgent motive while being protected from unnecessary interruption. The situation of fleeing a battleground might engender this state, in addition to causing physiological changes to increase action readiness. A low filter level, however, would correspond to a low level of focus, during which attentive resources are can be readily interrupted, easily shifting from one concern to another, a situation that might occur while talking among friends.

MINDER1's filter level is incrementally altered. However, there are no *a priori* reasons not to use different mechanisms. One possibility is to store the current lowest insistence level of surfaced motives. The filter level would then be made slightly higher than this value forcing the least insistent motive to dive. The problem of unmotivated design decisions is discussed in the next section.

3.7 Evaluation of the architecture

This section briefly describes how the architecture is to be evaluated, and the limitations of the implementation, including how it should and could be extended.

3.7.1 Evaluation

The main aim of building MINDER1 was to show (i) that our high level design could be implemented, albeit in a simplified fashion, (ii) that the design could, in principle, meet the requirements, and (iii) that this kind of motive processing architecture would lead to a processing state called perturbance. We have shown (i), but (ii) is more problematic, and (iii) is discussed in section 4.

A general problem of software engineering is to show or prove that a design, and its corresponding implementation, meets or satisfies a set of requirements. Design validation is difficult. MINDER1 appears to cope in its domain but a full evaluation of the architecture would require tests in a variety of domains, including more complex variants of the nursery domain, and the collection of performance statistics that could be compared with the performance of other possible designs. However, our goal was not to explore design-space searching for the 'best' motive management system (in any event, the best design would vary over niche-space). It is sufficient for current purposes that the prototype implementation demonstrates that it is possible that the original design meets its requirements.

A related point is that many of the detailed design decisions taken during implementation were arbitrary. In (Sloman, Shing, Read & Beaudoin, 1992) six types of design decision were identified: (i) design decisions linked to initial requirements, (ii) decisions linked to empirical data, (iii) decisions linked indirectly to requirements via higher level design decisions, (iv) decisions made in order to test a theory, (v) arbitrary decisions where previous requirements and design decisions do not prescribe a unique decision, and (vi) decisions due to hardware or software limitations[7]. Some of the high level design decisions were motivated. For example, three layers of processing is suggested by empirical evidence, in particular evolutionary neuroscience. Empirical observations of many kinds, such as the difficulty of attending to two conversations at once, justifies the design decision of management resource limits. In addition, there are theoretical reasons for such limits, such as limited physical resources imposing a bottleneck on cognition, limited memory resources imposing a limit on the creation and storage of temporary structures, and the need for mutual compatibility of adopted motives may limit the number that can be considered and adopted at any time. Insistence heuristics can be similarly justified, in particular from the requirement for reactivity in dangerous domains; for example, there is strong evidence of 'quick and dirty, emotional' processing pathways in the brain (LeDoux, 1994)). However, deciding on the precise form of representation of beliefs and other intentional structures was largely a matter of convenience. Accordingly, the implementation should be viewed as an exemplary illustration of our design theory, but the details of the implementation are of secondary importance. However, implementation remains an essential part of the design-based approach: not implementing MINDER1 would be like an engineer producing designs for bridges without ever building one to see if it stays up. It was always a possibility that the implementation would *fail* to manage multiple motives in the nursery domain. If so, we would have learnt why the implementation was inadequate, which may have motivated a revision of the theory. A full analysis, however, would include a study of the surrounding design-space. It must be admitted that it is possible that a fundamentally different design might also meet the requirements, for example if computers of the future have speeds many orders of magnitude faster than now.

MINDER1 is intended to have implications for an understanding of human minds because human minds have evolved to satisfy similar requirements: humans need to manage multiple motives with resource limited attentive processes. The gross mechanisms of the design – reactive motive generation, motive filtering, and motive management and metamanagement – are held to exist in human brains. Note however that there need be no invariant neuronal correlates of these mechanisms. The invention of the computer has demonstrated that the mapping between information processing mechanisms and physical implementation is not straightforward.

MINDER1 is an engineering solution to a control problem, and could be used as a command and control system or put to use in computer games, but this was not the main reason for building it; rather, it is intended as a 'toy designed to stretch our minds' (Sloman, 1978), in particular to help us think about the full complexity

[7] Existing work in software engineering attempts to formalise the relationships between requirements and design decisions, for example (MacLean, Young, Bellotti & Morgan, 1991). Work of this kind can help us think about niche-space and design-space and the mappings between them.

of emotional states. The relevance of MINDER1 to theories of emotion constitutes its primary scientific content and is discussed in section 5.

3.7.2 Some limitations

MINDER1 could be improved in many ways, both from the standpoint of developing agents that handle multiple motives in more intelligent ways, and from the standpoint of developing a richer cognitive model of motive processing.

All the mechanisms described could be 'deepened'. The management layer should include a planning module, and much more sophisticated scheduling mechanisms, including developing sophisticated measures of the urgency and importance of motives. The plan executor should be extended to notice opportunities or threats to current plans (Pryor & Collins, 1992; Pryor & Collins, 1993; Pryor, 1994), or possibilities for satisfying more than one motive with a single plan ('killing two birds with one stone'). Also, a more intelligent agent should construct its own insistence heuristics. Currently, they are hand-coded. (Humphreys, 1996) describes a reinforcement learning mechanism that learns the relative priorities of various goals in different contexts. The metamanagement layer should include mechanisms for 'self control', that is ways for motive management to be controlled, in particular to handle problematic emergent processing states (see next section). The filter mechanism could be extended to include a facility for 'exception handlers' (discussed in (Beaudoin, 1994)), which would allow management processes to selectively prevent classes of motives surfacing regardless of their insistence.

Finally, one of the difficulties of developing a motive processing architecture is the lack of a comprehensive theory of motive management. AI researchers have developed sophisticated theories of planning, including the extension of such work to cope with complex, uncertain and dynamic domains (e.g., (Nilsson, 1994; Firby, 1989; Pryor, 1994)). However, the problem of managing multiple motives, which includes synthesising such tasks as deciding whether to process a motive, at what time to expand and execute the motive, how to compare the benefits and costs of pursuing a motive compared to other motives, and so forth, has not been sufficiently addressed. A key problem in this context is how to effectively manage limited resources, particularly computational resources, in real-time environments. The mechanisms of reactive, heuristic motive generation, a filter mechanism, and deliberative motive management is our preliminary solution to this problem ((Bratman, Israel & Pollack, 1988) also propose a filter mechanism to meet similar requirements). Other work in decision-theoretic control (rational deliberation under resource limitations), such as anytime algorithms (Boddy & Dean, 1989), amended utility theory (Horvitz, Gregory & Heckerman, 1989), or rational self-government (Doyle, 1989), is also of relevance for developing a theory of motive management. Armed with such a theory we could improve upon MINDER1's primitive motive management mechanisms.

4 Emergent processing states

There is much philosophical wringing of hands over the meaning of the term 'emergence' and what it might really mean. Here the term is used in two ways. First, to refer to unexpected consequences of a design. Normally it is too difficult to deduce all the consequences of a design, which is one reason for the necessity of implementation. Both filter and decision oscillation are emergent in this sense. The second use of the term is to refer to processing states that arise from interactions between submechanisms. Perturbant states were hoped for consequences of the design but there is no 'perturbance-producing' mechanism or module. Emergence of this kind occurs because relations are real, that is the interactions between processes are just as real as the processes themselves (compare thrashing in operating systems, or the laws of supply and demand in economic systems). 'Emergent' is normally reserved for this kind of occurrence. Some of MINDER1's emergent states have psychological relevance.

4.1 Filter oscillation

If there are less than three surfaced motives the filter level is gradually lowered to allow any $m_i \in M_{sub}$ to surface into management. However, there are occasions when many $m_i \in M_{sub}$ have identical insistence values. Consequently, when the filter level is lowered a number of motives may surface at the same time. If this occurs the filter level needs to be raised because there are now *more* than three surfaced motives. However, when the filter is raised *all* the recently surfaced motives dive. The filter will then be re-lowered and the cycle repeats (see dense oscillation regions of the filter level in figure 7). The threshold level continues to oscillate until insistence levels change, or more insistent motives are generated, or the active motive is completed, or a motive is removed, and so forth.

We had not considered the possibility of filter oscillation but in retrospect in seems unavoidable when the implementation of the filter relies on a real number that is incremented or decremented in discrete amounts. Moreover, the granularity of some insistence heuristics is not sufficient to assign unique insistence values to different motives. However, filter oscillation is a feature of this implementation, but not the design. Oscillation could be avoided by using a neural network or 'fuzzy' implementation of the filter mechanism, and (Beaudoin, 1994) discusses a 'filter refractory period' that briefly increases the resistance of the filter after a motive surfaces.

4.2 Decision oscillation

Occasionally, MINDER1 will 'see' two minibots that are close to a ditch. The new sense datum generates new beliefs that generate new motives to rescue the minibots from falling into the ditch and damaging themselves. Normally, MINDER1 will, all other things being equal, adopt the motive with the higher insistence[8], grab the particular minibot, and remove it to a safe distance. However, if in the course of executing the plan for this motive, the other minibot moves even closer to the ditch, MINDER1 may drop what it is doing and attempt to rescue it. Occasionally, however, both minibots are a similar distance from the ditch, which results in a similar magnitude of insistence for each motive. Such a situation results in 'dithering' or 'indecision', both internally, in terms of the repeated adoption and replacement of each motive by the other, and externally, in terms of the agent repeatedly moving first to one dependent then stopping and moving towards the other. Occasionally, such 'indecision' results in neither motive being completed and both minibots falling into the ditch.

There are at least three ways to avoid this problematic motive processing state. Abilities to construct a single plan to satisfy more than one motive would enable MINDER1 to save both minibots. Alternatively, an implementation supporting developed measures of urgency and importance could impose an ordering on motives

[8]Note that, in our design, insistence is not a criterion for adoption but for consideration. However, because MINDER1's management processes do not develop measures of urgency and importance, insistence normally has this role in the implementation.

when insistence heuristics do not. Finally, metamanagement processes could detect states of decision oscillation and arbitrate. However, there are examples from human and animal behaviour of decision oscillation, which suggests that, in general, the state cannot be avoided. For instance, humans are extremely indecisive when confronted with hard ethical problems.

4.3 Perturbant states

The term *perturbant* refers to a state in which a partial loss of control of thought processes is due to the continual surfacing of postponed or rejected, or unwanted, motivators, or possibly disruptive thoughts, images, and the like (e.g., a catchy tune that won't 'go away'). Such disruption can interfere with the management of other, important, motives. Perturbances are the type of information processing state that the 'attention filter penetration' (AFP) theory posits as a *characteristic feature* of many human emotional states (Simon, 1967; Sloman & Croucher, 1981; Sloman, 1987; Sloman, 1992). For example, both intense grief and joy involve perturbant states: the mourner and lottery winner both find it difficult to direct their attention to other concerns, that is there is a notion of 'loss of control' common to both states. Note that perturbances can exist in dispositional forms: the mourner may function normally at work only to break down in the evening. During the day the perturbance was dispositional, in the evening occurrent. Note also that 'perturbant' is not a sufficient condition for an 'emotional' state.

In this section we describe how MINDER1 can *potentially* support perturbant states but lacks the necessary architectural features for a complete simulation of 'loss of control' of thought processes.

4.3.1 Perturbant scenario

Consider the following scenario that could occur in the nursery domain.

> Robby the robot minder notices that a dependent has fallen into a ditch and has been damaged. Robby decides not to retrieve the damaged minibot because it has other pressing things to do. It has various active motives that are more urgent and important, such as recharging other active minibots, building a protective enclosure of fences, and ensuring that more minibots do not fall into ditches. However, the thought of the damaged minibot lying there continues to enter Robby's thoughts, diverting attentive processing resources from the current set of active tasks. Robby can't seem to get the thought out of his mind and finds it difficult to concentrate on the task at hand ... Sometimes, however, Robby is so very busy looking after the minibots that he temporarily forgets that there is a damaged dependent waiting to be repaired ... When things eventually calm down, Robby retrieves the damaged minibot and places it in the dismissal point for repair.

MINDER1 is not as sophisticated as Robby the robot because, unlike Robby, MINDER1 has no mechanisms that can support a notion of 'loss of control' of management processing. There are no metamanagement processes that express goals about what *should* be occurring in management. Without these specific kinds of normative, goal-directed processes there can be no notion of MINDER1 controlling or failing to control its own management processing. Perturbance requires an architecture sufficiently sophisticated to support goals whose object is to control management (or 'attentive') processing.

For example, a person hears the latest pop single on the radio and soon the tune is constantly 'replayed' in their mind, perhaps diverting attentive resources from other matters. This is not a perturbant state unless the following condition holds: *if* the person were to wish that the tune would 'go away' and tried to put it out of their mind but found it difficult to do so *then* there is a disposition to lose control of attentive resources. For instance, some catchy tunes can become very irritating and annoying. However, the loss of control need not be occurrent for a perturbant state to exist. For example, excited anticipation is a perturbant state even if the person experiencing that state does *not* wish to turn their thoughts to other matters (they may be quite happy continually thinking about the presents they will get for their birthday). The definition of perturbance is therefore counterfactual, and requires an architecture sufficiently sophisticated to support goals whose object is to control management (or 'attentive') processing. MINDER1 does not have this architectural complexity; consequently, it can only partially or potentially support perturbant states. However, it *does* possess mechanisms that support the continual surfacing of motives that are repeatedly rejected by management processes. The repeated interruption of management processing is precisely the kind of state that has the potential to conflict with higher level metamanagement decisions if the mechanisms for such decisions were added to the architecture. It is these potential or *proto-perturbances* that are now described.

4.3.2 Proto-perturbances in MINDER1

To show that MINDER1 could potentially support perturbant states we ensured that motives pertaining to damaged minibots were assigned relatively high insistence values that management processes would subsequently 'disagree' with. That is, management processes would reject these motives by assigning them *low* importance, contradicting their heuristically assigned insistence values (see node 4 in figure 6). The design decision was made in order to test a theory (see section 3.7.1) and is therefore slightly contrived. However, if MINDER1 possessed more sophisticated motive management processing then contradictions between insistence and developed measures of urgency and importance would be commonplace, some of which may be correctable by learning.

In order to detect any proto-perturbances a metamanagement process was devised that measured the *rate of rejection* of motives. If the rate exceeded a threshold then an occurrent proto-perturbant state was detected. If the rate subsequently dropped below the threshold then the state had ended. MINDER1 maintains internal records of proto-perturbant episodes. However, it must be stressed that, at present, the information about the occurrence of a proto-perturbant state is not used by metamanagement to control the state. For example, such information could be used to change the insistence heuristics for such motives, or place exception handlers in the filter mechanism. But this kind of 'self-control' of motive processing was not implemented. If it was then we would have an architecture that had goals directed towards controlling its own management processes, and the beginnings of a simulation of a perturbant state proper.

Figure 7 is a graphical representation of some statistics collected during a single run of MINDER1 in the nursery domain. It shows that MINDER1 supports proto-perturbant states and correlates their detection (shaded areas labelled 'P') with the total number of agent motives (motive axis) and the filter threshold level (filter

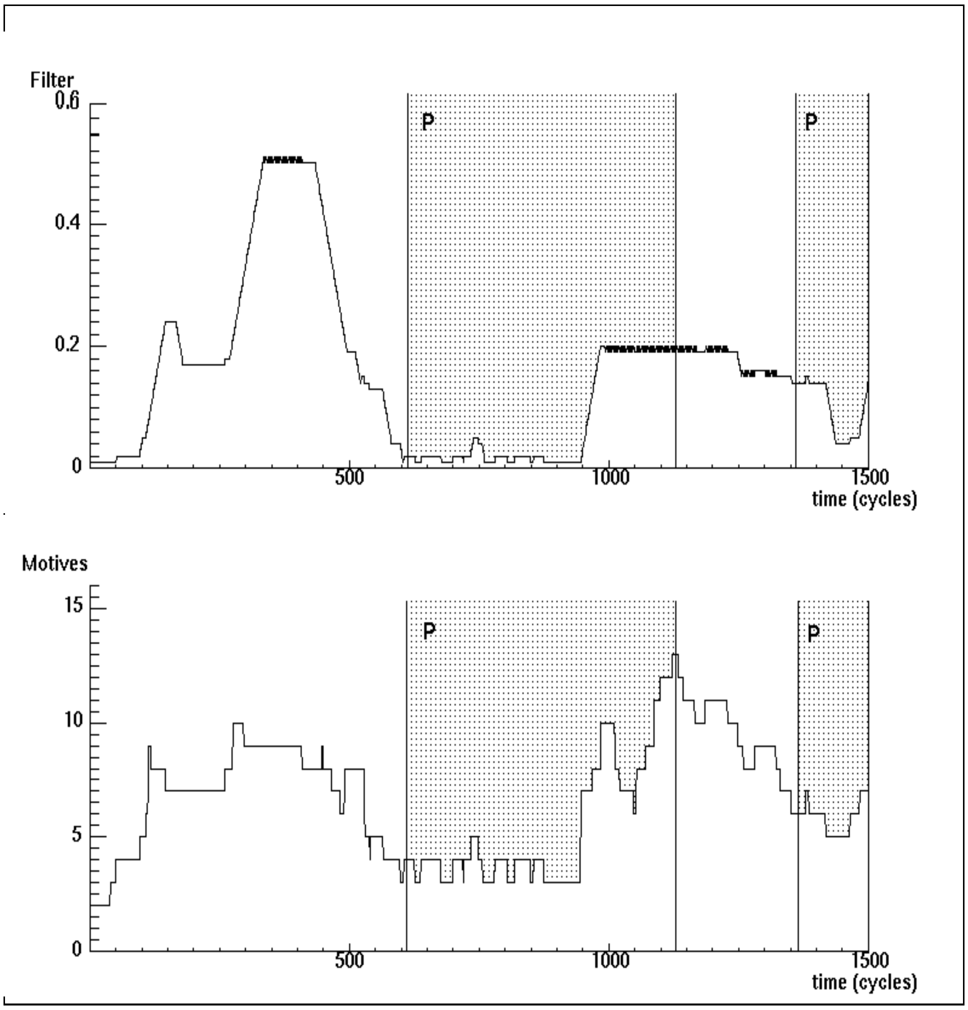


Figure 7: **Filter level, number of motives and proto-perturbant (P) episodes**

axis) over time (cycles axis)[9]. A proto-perturbant state occurs when MINDER1 knows that a minibot has fallen into a ditch. The resultant belief satisfies the preconditions of a generactivator that generates a motive to dismiss it. Management processes decide that the importance of this motive is low compared to other motives and rejects it. However, the insistence level is such that the motive resurfaces into management processing. If this event happens with sufficient frequency, metamanagement processes detect the proto-perturbant state. Yet if there are many other highly insistent motives the filter may be raised to such a level that the dismissal motive cannot surface. When this occurs the proto-perturbant episode temporarily ceases (see cycles 1137 to 1355 in figure 7). When the highly insistent motives have been dealt with, the filter may be lowered allowing the dismissal motive to once again grab management resources, demonstrating that a proto-perturbance may be either occurrent or dispositional.

MINDER1's proto-perturbances are a pale shadow of human perturbances. For instance, MINDER1's proto-perturbances are all of one type. They arise from the difference between the insistence value of the motive and the importance assigned

[9]The run included 5 minibots and 4 fences. These parameters can be varied.

to it by management processes. There are no 'catchy tunes' in MINDER1, or vivid episodic memories, or unsatisfiable motives, and so forth. Compared to the flora and fauna of the human mind, and the corresponding variety of perturbant states, MINDER1 is a simple automaton. But although MINDER1 does not support 'loss of control' it may, from the outside, be judged as dysfunctional when in a proto-perturbant state by an observer who knows what is in the best long term interests of the agent, even if the agent cannot make that assessment (compare parents watching their young children).

5 Perturbance and theories of emotion

Perturbant states are emergent phenomena arising from the interaction of resource-limited attentive processing, an automatic subsystem that generates new candidates for such processing, a heuristic filter mechanism, and a higher level system that may attempt to control attentive processing. These design elements are inferred from requirements for and constraints upon human-like autonomy in a complex and dynamic environment. Perturbances do not arise due to a special perturbance generating mechanism. Thus it is misguided to ask what the *function* of perturbant states is or to postulate a perturbance mechanism. However, the mechanisms that generate perturbant states may themselves be functional and have evolved, in natural minds at least, for specific, adaptive purposes. For example, (Aube & Senteni, 1996a; Aube & Senteni, 1996b) view 'emotions' as those control structures that specifically evolved to regulate the flow of commitments (goals to provide other agents with resources) between individuals in animal and human societies. We do not want to argue over what 'emotions' *really* are, for that assumes that our everyday concept of emotion refers to a well-specified set of phenomena. Like many other pre-theoretical terms it can hide much more than it reveals. Instead of arguing over definitions of words we identify some phenomenon and attempt to explain it. Whether those phenomena concur with others' definitions of emotions is a side-issue. Perturbances exist, and they are a ubiquitous feature of many states that are commonly called emotional. However, there are many different kinds of perturbances and a full analysis of their variety would not exhaust an analysis of the emotions.

Approaches to the study of emotions can be very broadly categorised as *semantics*-based, *phenomena*-based and *design*-based (Sloman, 1992). Semantics-based theories analyse the use of language to uncover implicit assumptions underlying emotion words (e.g., (Wierzbicka, 1992)). Phenomena-based theories assume that emotions are a well-specified category and attempt to correlate contemporaneous and measurable phenomena with the occurrence of an emotion, such as physiological changes (an early example is William James' theory – see (Calhoun & Solomon, 1984); for a comprehensive review of many phenomena-based theories, see (Strongman, 1987)). MINDER1 shows that phenomena-based approaches are necessarily limited: the causal relations between perturbant states and observable behaviour[10] are indirect. Any highly complex information processing mechanism will have this property. However, this is not to say that phenomena-based approaches are of no use; on the contrary, they have generated empirical data and driven the development of theories. But to understand fully the complexity of emotional states, which can involve complex internal states not directly linked to observable phenomena, we also require a design-based approach.

[10]That is, the causal relations between internal states and behaviour observable by other agents in the nursery domain. If we possessed tracing facilities for human minds that enabled us to examine their internal processes then the problems of psychology would be largely solved.

5.1 Other design-based simulations

The AFP theory of emotion that underpins MINDER1 was inspired by Herbert Simon's work (Simon, 1967) that focussed on the importance of interrupts and multiple motives for adaptive behaviour in real-time environments, and the relation between interruption of current goals and emotional states. Oatley and Johnson-Laird's communicative theory of the emotions (Oatley & Johnson-Laird, 1985; Oatley, 1992) also views interruption of cognitive systems as a characteristic feature of emotional states (although there are a number of important differences that cannot be discussed here). The common feature of such 'interrupt' theories of emotion is that they emphasise the aspect of *control* over representation. For example, all theories posit a subsystem that can be interrupted by another process that makes demands on its functioning. MINDER1 is the first, albeit simple, example of a fully-functioning implementation of an 'interrupt' theory of emotionality.

There are other design-based simulations of emotional phenomena. However, they tend to emphasise *representation* over emergent processing states and concentrate on the semantics of emotional appraisals or emotion words. For example, Dyer's BORIS, OpEd and DAYDREAMER systems (Dyer, 1987) appraise story or 'daydreaming' scenarios with respect to built-in goals. The appraisals may then generate a prediction of what emotional state is appropriate in the given scenario. These systems, therefore, reason about emotional labels and their semantics. Frijda and Swagerman's ACRES system (Frijda & Swagerman, 1987) is a computer program that stores facts about emotions and reasons about those facts, but, in addition, has various goals or concerns that it attempts to meet, such as a concern to have correctly typed input. The satisfaction or dissatisfaction of the program's concerns may cause it to interrupt current processing and generate new responses, such as a request to the user for correctly typed input. ACRES can be asked for information regarding its internal state, which is a measure of how well its concerns have been met. However, ACRES is not an implementation of an architecture that can support a distinction between attentive and pre-attentive processing, and does not exhibit emergent processing states. Pfeifer's FEELER system, reviewed in (Pfeifer, 1994), also predicts appropriate emotional states given story scenarios. The OZ project has investigated the role of emotion in artificial, believable agents (Bates, 1994; Reilly, 1993), that is agents that make it easy for an audience to suspend their disbelief and accept the 'reality' of the agents before them, much as an audience accepts that an actor is King Lear. However, the mechanisms driving the emotional agents are designed to express emotional states in animated movement, to time and accentuate those expressions for maximum effect, and to perform pre-scripted emotional scenarios. There can be no loss of control of attentive processing in these agents, though they may be able to 'smile' in appropriate scenarios.

Such implementations tend to 'program in' representations of emotional intentionality. The approach adopted here is quite different. The process moves from requirements for complete functioning agents, to designs that meet those requirements, and the testing of implementations. Perturbant states arise from other mechanisms designed to meet those requirements. In the abstract, the requirements for MINDER1 are the same as those for human-like autonomy; hence, perturbant states have greater claim to model actual aspects of human information processing. Other models have difficulties making such claims.

An exception is Moffat and Frijda's WILL architecture (Moffat & Frijda, 1995), partly similar to MINDER1, and a design for a concern realisation system. However, WILL does not exhibit protoemotional states.

The relevance of perturbance to an analysis of the emotions has been discussed in our other publications, in particular an analysis of grief (Wright, Sloman & Beaudoin, 1996), and theories of how an architecture could support painful or pleasurable perturbant states (Wright, 1996b; Wright, 1996a).

6 Conclusion

Implementing agent architectures, even with good tools, is a time consuming exercise and presents difficult software engineering and artificial intelligence problems. However, it is a necessary stage of the design-based approach, an approach that we believe is currently the only way to explore fully the complexity of mental phenomena. By building MINDER1 we have shown that our paper design can meet the discipline of computational realisation, or at least that part of the design that we have managed to build given the available resources; and that it probably meets its requirements, or at least this particular implementation appears to manage multiple motives in the nursery domain. These results give us greater confidence that the gross functional decomposition of our design corresponds to information processing mechanisms that exist in human minds, for human minds also have to manage multiple motives in complex and dynamic domains while maintaining reactivity to current events.

In addition to having relevance to agent architecture research, MINDER1 has the architectural prerequisites to support perturbant states that involve a loss of control of attentive resources. Perturbant states are characteristic features of emotional states. Therefore, MINDER1 can be described as a protoemotional architecture, as long as this is understood not to be a claim about the first-person phenomenology of MINDER1 nor a claim about what constitutes emotionality in general. MINDER1 can enter states in which a motive continually surfaces through a variable threshold attention filter despite being continually rejected by resource limited management processes. This is precisely what is meant by 'protoemotional', nothing more and nothing less.

7 Acknowledgments

Thanks to Brian Logan, Darryl Davies, Luc Beaudoin and Michel Aube.

8 Appendix

8.1 Motives

The general form of a motive is:

```
[MOTIVE motive      <motive descriptor>
        insistence  <insistence value {0..1}]>
        status      <status descriptor {sub,surfacing,suspended,active}>
        plan        <list of plan steps>
        trp         <list of current TR program actions>
        importance  <importance descriptor {normal,low}>
]
```

The seven types of motive in MINDER1 are:

1. **[MOTIVE motive [recharge ?Obj] insistence ?insist status sub plan** [] **trp** []]: A motive to recharge the specified minibot. Generated if MINDER1 believes that an object has low charge.

2. **[MOTIVE motive [enclose ?Obj] insistence ?insist status sub plan** [] **trp** []]: A motive to move the specified minibot to the northern area of the nursery behind the line of fences. Generated if the enclosure has been built and MINDER1 believes a minibot is south of it. (Helps prevent minibots from falling into ditches.)

3. **[MOTIVE motive [dismiss ?Obj] insistence ?insist status sub plan** [] **trp** []]: A motive to move the specified minibot to the dismissal point. Generated if MINDER1 believes a minibot has fallen into a ditch.

4. **[MOTIVE motive [visit ?Obj] insistence ?insist status sub plan** [] **trp** []]: A motive to visit the specified ditch. Generated periodically. (Patrols the ditches to spot minibots that might fall into them.)

5. **[MOTIVE motive [no_maginot] insistence ?insist status sub plan** [] **trp** []]: A motive to build an enclosure. Generated if an enclosure has not been built.

6. **[MOTIVE motive [save ?Obj2 ?Obj1] insistence ?insist status sub plan** [] **trp** []]: A motive to move the specified minibot to a safe distance from the specified ditch. Generated if MINDER1 believes a minibot is too close to a ditch.

7. **[MOTIVE motive [default] insistence ?insist status sub plan** [] **trp** []]: A motive to wander around the nursery. Always generated (a motive of last resort).

8.2 List of basic actions

MINDER1 has a set of basic actions that are directly executable by effectors. However, they can fail; for example, MINDER1 may attempt to move forward but cannot due to an obstacle. The basic actions are:

1. **move**: Move forward in current direction. Takes no arguments.

2. **rotate ?To**: Rotates to new direction as specified by argument.

3. **setspeed ?To**: Set new travelling speed as specified by argument.

4. **grab_object ?It**: Attempt to grab object identified by argument, for example "minibot2". (This action can fail if the object is not within reach.)

5. **drop ?It**: Drop object identified by argument.

6. **rotate_bar ?It ?To**: Rotate specified fence to specified direction. (MINDER1 must be holding fence identified by first argument.)

7. **charge ?It ?With**: Charge specified object with specified object, for example "charge minibot2 gas1". (This action can fail if first object not held or second object not within reach.)

8. **dismiss ?It ?With**: Dismiss specified object at specified object, for example "dismiss minibot2 exit1".

8.3 List of prestored plans

Plans consist of teleo-reactive program (TRP) plan steps, and most plans have a single plan step. There is a unique plan for each motive, which is a major simplification. Listed below are the seven types of motive and their associated plans.

1. **motive** [**recharge ??Params**] **==>**
plan [[**charge_object ??Params 200**]]:

2. **motive** [**below_maginot ??Params**] **==>**
plan [[**take_object ??Params exit1**]]:

3. **motive** [**dismiss ??Params**] **==>**
plan [[**dismiss_object ??Params**]]:

4. **motive** [**visit ??Params**] **==>**
plan [[**drop**] [**goto_object ??Params**]]:

5. **motive** [**no_maginot ??Params**] **==>**
plan [[**make_wall 15 60 0 first**] [**make_wall 40 60 0 second**] [**make_wall 65 60 0 third**] [**make_wall 90 60 0 fourth**]]:

6. **motive** [**save ??Params**] **==>**
plan [[**put_safe ??Params**]]:

7. **motive** [**default ??Params**] **==>**
plan [[**drop**] [**search**]]:

MINDER1 has two metaplans that specify internal management operations.

8. **DECIDE**: Determines the importance of a motive.

9. **GET_PLAN**: Retrieves the correct plan for a motive from the plan library.

8.4 List of TR programs

Each TRP consists of a set of production rules. Rule conditions match items of knowledge and rule actions can be calls to further TRPs, recursive calls to the same TRP, internal operations that schedule basic actions, or assertions of temporary beliefs to facilitate 'reasoning', for example reasoning about available fences when building an enclosure. Each TRP name and associated arguments is listed, followed by a short description of its function and the basic actions and other TRPs it may call.

1. **TRP_goto ?X ?Y**: Move to the specified coordinates. Uses [move, setspeed, rotate].

2. **TRP_amble ?X ?Y**: Move to the specified coordinates while avoiding obstacles. Calls [1, 2].

3. **TRP_goto_object ?Obj**: Move to the specified object. Calls [2, 7].

4. **TRP_grab_object ?Obj**: Pick up the specified object. Uses [grab_object]; calls [3, 6].

5. **TRP_take_object ?Obj1 ?Obj2**: Take the specified object to the second specified object. Calls [3, 4].

6. **TRP_drop ?Obj**: Drop the specified object. Uses [drop].

7. **TRP_search**: Search the nursery. Calls [2].

8. **TRP_place_bar ?Obj ?X ?Y ?Heading**: Place specified fence at specified coordinates in specified direction. Uses [rotate_bar]; calls [6, 2].

9. **TRP_grab_wall**: Pick up a fence that can serve as a wall. Calls [6, 4, 7].

10. **TRP_make_wall ?X ?Y ?Heading ?Side**: Place a fence so to serve as a wall. Calls [8, 9].

11. **TRP_charge_object ?Obj ?Level**: Charge specified object to specified level. Uses [charge]; calls [6, 5].

12. **TRP_dismiss_object ?Obj**: Remove specified object from the nursery. Uses [dismiss]; calls [5].

13. **TRP_put_safe ?Obj1 ?Obj2**: Remove specified object to a safe distance from specified ditch. Calls [6, 2, 4].

8.5 Example trace

The following short trace shows the state transitions of perturbing motives involved in a cycle of 'rumination' (see section 4.3.2).

```
======================= end of cycle 641 ==================
 ** [[Surfacing --
                [MOTIVE motive
                        [dismiss minibot4]
                        insistence 0.15 status sub plan
                        [[decide] [get_plan]]
                        trp
                        [done decide]
                        importance low]]]
 ** [[Diving --
             [MOTIVE motive
                     [dismiss minibot9]
                     insistence 0 status suspended plan
                     [[decide] [get_plan]]
                     trp
                     [done decide]
                     importance low]]]
 ** [[Management
      rejects --
      [MOTIVE motive
              [dismiss minibot7]
              insistence 0.15 status active plan
              [[decide] [get_plan]]
              trp
```

```
              [done decide]
              importance low]]]

 ======================= end of cycle 642 ==================
 ** [[Surfacing --
               [MOTIVE motive
                       [dismiss minibot9]
                       insistence 0.15 status sub plan
...
...
```

8.6 Example code

Example code is included to show how to implement mechanisms using the SIM_AGENT syntax and production rule semantics. The first example shows some code that generates a motive if MINDER1 has a belief that any minibot is too near a ditch. The second example shows how the teleoreactive program described in figure 3 was actually implemented. Comments are provided throughout.

An example generactivator:

```
;;; G_near_ditch----------------------------------------------------

;;; rules-----------------------------------------------------------

;;; This generactivator is composed of one ruleset that is itself
;;; composed of a number of rules. Each rule contains a condition
;;; part that can match with items in MINDER1's database or memory
;;; store, and an action part that can place new items in the database.

define :ruleset G_near_ditch;

rule grule_near_ditch_remove_r1
;;; This rule removes a generated motive if its rationale no longer
;;; holds, that is the minibot has fallen into the ditch (it is too
;;; late to do anything about it!)

    [MOTIVE motive [save ?Obj2 ?Obj1] ==][ ->> Motive ]
        ;;; This line matches a declarative representation of a motive
        ;;; in MINDER1's database.
    [belief == name ?Obj1 type minibot status dead ==]
        ;;; And a belief about a minibot of the same name as the
        ;;; matched motive.
==>
    [DEL ?Motive]
        ;;; If the conditions match items in the database then
        ;;; there is a motive to save a minibot that has already
        ;;; fallen into a ditch; therefore remove it.

rule grule_near_ditch_remove_r2
;;; This rule removes a generated motive if the insistence is
;;; computed to be zero, that is the minibot has moved out of
;;; the danger zone of its own accord.

    [MOTIVE motive [save ?Obj2 ?Obj1] insistence ?Insistence ==]
        [ ->> Motive ]
    [WHERE Insistence = 0 ]
==>
    [DEL ?Motive]

rule grule_near_ditch_add
;;; This rule generates a motive if there is a minibot too close
```

```
;;; to a ditch, and a motive to save it has not already been
;;; generated.

    [belief == name ?Obj1 type minibot status alive == x ?X y ?Y ==
        held false ==]
    [belief == name ?Obj2 type ditch == x ?X1 y ?Y1 == poly_space ?Polygon ==]
    [WHERE
        is_near_ditch(Polygon, X1, Y1, X, Y, near_ditch_proximity) ]
    [NOT MOTIVE motive [save ?Obj2 ?Obj1] ==]
==>
    [LVARS insist]
    [POP11 near_ditch_insistence(Polygon, X1, Y1, X, Y) -> insist; ]
        ;;; The insistence value is computed for this motive
        ;;; (near_ditch_insistence is a function defined elsewhere).
    [MOTIVE motive [save ?Obj2 ?Obj1] insistence ?insist status sub]
        ;;; Place the new motive in the database: it is of status sub
        ;;; and becomes a candidate for surfacing.

rule grule_near_ditch_change
;;; This rule recomputes the insistence values of previously generated
;;; motives (the reactivation part of generactivation).

    [MOTIVE motive [save ?Obj2 ?Obj1] insistence ?Insistence ==]
    [belief == name ?Obj1 type minibot status alive == x ?X y ?Y ==
        held false ==]
    [belief == name ?Obj2 type ditch == x ?X1 y ?Y1 == poly_space ?Polygon ==]
    [LVARS insist]
    [WHERE
        near_ditch_insistence(Polygon, X1, Y1, X, Y) -> insist;
        insist /= Insistence ]
==>
    [MODIFY 1 insistence ?insist]
        ;;; Modify the insistence value of the existing motive.

enddefine;
```

An example TR program:

```
;;; TRP_charge_object ----------------------------------------------

;;; rules -------------------------------------------------------------

;;; Note that these rules satisfy the regression condition; therefore,
;;; the order of the rules matter. For instance, the topmost rule
;;; checks that the action has been accomplished and simply notes this
;;; fact (the null action in figure~\ref{fig:trp}).

define :ruleset TRP_charge_object;

rule TR_charge_object_r1
    [MOTIVE motive == status active == trp [charge_object ?Obj ?Level] ==]
        ;;; There exists a motive to charge an object.
    [NOT held ?Obj]
        ;;; MINDER1 is not holding the object.
    [new_sense_datum == name ?Obj == charge ?Level2 ==]
    [WHERE Level2 >= Level ]
        ;;; MINDER1 can sense that the object has been recharged to
        ;;; the appropriate level.
     ==>
    [MODIFY 1 trp [done charge_object ?Obj ?Level]]
        ;;; Therefore, the task has been accomplished.
    [STOP]

rule TR_charge_object_r2
    [MOTIVE motive == status active == trp [charge_object ?Obj ?Level] ==]
        ;;; There exists a motive to charge an object.
    [held ?Obj]
```

```
        ;;; MINDER1 is holding the object.
    [new_sense_datum == name ?Obj == charge ?Level2 ==]
    [WHERE Level2 >= Level ]
        ;;; MINDER1 can sense that the object has been recharged to
        ;;; the appropriate level.
     ==>
    [MODIFY 1 trp [drop ?Obj]]
    [POP11
        prb_run(TRP_drop, sim_myself.sim_data, false);
    ]
        ;;; Therefore, drop the object because it is sufficiently
        ;;; charged. The call to prb_run is a recursive call to
        ;;; another TR program that drops the specified object.
    [STOP]

rule TR_charge_object_r3
    [MOTIVE motive == status active == trp [charge_object ?Obj ?Level] ==]
        ;;; There exists a motive to charge an object.
    [held ?Obj]
        ;;; MINDER1 is holding the object.
    [new_sense_datum == name ?Obj == x ?X y ?Y ==]
    [new_sense_datum == name gas1 == x ?XX y ?YY ==]
    [WHERE close_enough(X, Y, XX, YY) ]
        ;;; MINDER1 can sense that the object is near enough to the
        ;;; recharge point (gas1) for recharging.
     ==>
    [closure charge ?Obj gas1]
        ;;; Call a primitive action to charge the object at the
        ;;; recharge point. Note that this action will be repeated
        ;;; on subsequent cycles until rule r2 evaluates to true.
    [MODIFY 1 trp [stop]]
    [STOP]

rule TR_charge_object_r4
    [MOTIVE motive == status active == trp [charge_object ?Obj ?Level] ==]
        ;;; There exists a motive to charge an object.
     ==>
    [MODIFY 1 trp [take_object ?Obj gas1]]
    [POP11
        prb_run(TRP_take_object, sim_myself.sim_data, false);
    ]
        ;;; Recursive call to another TR program, take_object, that
        ;;; will itself call other TR programs that will attempt
        ;;; to locate the specified object, pick it up, and take it to
        ;;; the recharge point (or search for the recharge point it
        ;;; its location is unknown).
    [STOP]

enddefine;
```

References

Agre, P. E. & Chapman, D. (1987). Pengi: an implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Seattle. AAAI.

Aube, M. & Senteni, A. (1996a). Emotions as commitments operators: a foundation for control structure in multi-agent systems. In *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agents World, MAAMAW '96, Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Aube, M. & Senteni, A. (1996b). What are emotions for? commitments management and regulation within animals/animats encounters. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., & Wilson, S. W. (Eds.), *From Animals to Animats IV, Proceedings of the Fourth International Conference on the Simulation of Adaptive Behavior*, pages 264–271. The MIT Press.

Bates, J. (1994). The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125.

Bates, J., Loyall, A. B., & Reilly, W. S. (1991). Broad agents. In *Paper presented at AAAI spring symposium on integrated intelligent architectures.* (Available in SIGART BULLETIN, 2(4), Aug. 1991, pp. 38–40).

Beaudoin, L. P. (1994). *Goal processing in autonomous agents.* PhD thesis, School of Computer Science, The University of Birmingham.

Beaudoin, L. P. & Sloman, A. (1993). A study of motive processing and attention. In A.Sloman, D.Hogg, G.Humphreys, Partridge, D., & Ramsay, A. (Eds.), *Prospects for Artificial Intelligence*, pages 229–238. Amsterdam: IOS Press.

Benson, S. & Nilsson, N. J. (1995). Reacting, planning, and learning in an autonomous agent. In Furukawa, K., Michie, D., & Muggleton, S. (Eds.), *Machine Intelligence 14*. Oxford: The Clarendon Press.

Boddy, M. & Dean, T. (1989). Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, vol. 2.*

Bratman, M. E., Israel, D. J., & Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355.

Brooks, R. A. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15.

Brooks, R. A. (1991a). Integrated systems based on behaviours. *SIGART Bulletin*, 2(4):46–50.

Brooks, R. A. (1991b). Intelligence without representation. *Artificial Intelligence*, 47:139–159.

Calhoun, C. & Solomon, R. C. (1984). *What is an Emotion?* Oxford University Press.

Chapman, D. (1989). Penguins can make cake. *AI Magazine*, 10(4):45–50.

Chapman, D. (1990). Vision, instruction and action. Technical Report 1204, Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

Davis, D. N. (1996). Reactive and motivational agents: towards a collective minder. In *Agent Theories, Architectures, and Languages, the Third International Workshop*, Budapest, Hungary. ECAI-96.

Davis, D. N., Sloman, A., & Poli, R. (1995). Simulating agents and their environments. *AISB Quarterly.*

Doyle, J. (1989). Reasoning, representation, and rational self-government. In Ras, Z. W. (Ed.), *Methodologies for intelligent systems*, pages 367–380. New York: Elsevier Science Publishing.

Dyer, M. G. (1987). Emotions and their computations: Three computer models. *Cognition and Emotion*, 1(3):323–347.

Firby, R. J. (1989). *Adaptive execution in complex dynamic worlds.* PhD thesis, Department of Computer Science, Yale University.

Frijda, N. H. (1986). *The Emotions.* Cambridge: Cambridge University Press.

Frijda, N. H. & Swagerman, J. (1987). Can computers feel? theory and design of an emotional system. *Cognition and Emotion*, 1:235–257.

Georgeff, M. P. & Ingrand, F. F. (1989). Decision-making in an embedded reasoning system. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972–978, Detroit, MI. IJCAI.

Georgeff, M. P. & Lansky, A. L. (1989). Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 677–682, Seattle. AAAI.

Horvitz, E. J., Gregory, F. C., & Heckerman, D. E. (1989). Reflection and action under scarce resources: theoretical principles and empirical study. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, vol. 2.*

Humphreys, M. (1996). Action selection methods using reinforcement learning. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., & Wilson, S. W. (Eds.), *From Animals to Animats IV, Proceedings of the Fourth International Conference on the Simulation of Adaptive Behavior.* The MIT Press.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64.

LeDoux, J. E. (1994). Emotion, memory and the brain. *Scientific American*, pages 32–39.

Logan, B. S. (1996). Personal communication.

MacLean, A., Young, R. M., Bellotti, V. M. E., & Morgan, T. P. (1991). Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6:201–250.

Marjanovic, M., Scassellati, B., & Williamson, M. (1996). Self-taught visually-guided pointing for a humanoid robot. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., & Wilson, S. W. (Eds.), *From Animals to Animats IV, Proceedings of the Fourth International Conference on the Simulation of Adaptive Behavior*, pages 35–44. The MIT Press.

Moffat, D. & Frijda, N. H. (1995). Where there's a will there's an agent. In Wooldridge, M. & Jennings, N. (Eds.), *Intelligent Agents.* Berlin: Springer-Verlag.

Nilsson, N. J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, 1:139–158.

Norman, T. J. (1994). Position paper: motivated goal and action selection. Presented at AISB workshop, Models or Behaviours, which way forward for robotics? Leeds, April 1994, and University College London research note RN/94/18.

Norman, T. J. (1996). *Motivation-based direction of planning attention in agents with goal-autonomy.* PhD thesis, Department of Computer Science, University College London.

Norman, T. J. & Long, D. P. (1995). Goal creation in motivated agents. In Wooldridge, M. J. & Jennings, N. R. (Eds.), *Intelligent Agents: Proceedings of the ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, pages 277–290. Springer-Verlag. Volume 890 of Lecture Notes in Artificial Intelligence.

Oatley, K. (1992). *Best Laid Schemes.* Studies in Emotion and Social Interaction. Cambridge: Cambridge University Press.

Oatley, K. & Johnson-Laird, P. N. (1985). Sketch for a cognitive theory of emotions. Technical Report CSRP 045, School of Cognitive Science, University of Sussex.

Pfeifer, R. (1994). The 'fungus eater approach' to emotion: a view from artificial intelligence. Cognitive Studies*: Bulletin of the Japanese Cognitive Science Society*, 1(2):42–57. Extended and revised version of an invited talk at *AISB-91*, Leeds, UK. Also available as a technical report from AI Lab, Institute for Informatics, University of Zurich-Irchel.

Pryor, L. (1994). *Opportunities and planning in an unpredictable world.* PhD thesis, Northwestern University.

Pryor, L. & Collins, G. (1992). Reference features as guides to reasoning about opportunities. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society.* Lawrence Erlbaum Associates.

Pryor, L. & Collins, G. (1993). Cassandra: planning for contingencies. Technical Report 41, The Institute for Learning Sciences, Northwestern University.

Rao, A. S. & Georgeff, M. P. (1991a). An abstract architecture for rational agents. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, Boston.

Rao, A. S. & Georgeff, M. P. (1991b). Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., & Sandewall, E. (Eds.), *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, Cambridge, MA, USA. Morgan Kaufmann Publishers.

Reilly, W. S. (1993). Emotions as part of a broad agent architecture. In *Proceedings of the Workshop on Architectures Underlying Motivation and Emotion, WAUME93*, Birmingham, UK.

Simon, H. A. (1967). Motivational and emotional controls of cognition. Reprinted in *Models of Thought,* Yale University Press, 29–38, 1979.

Sloman, A. (1978). *The Computer Revolution in Philosophy: Philosophy, Science and Models of Mind.* Hassocks, Sussex: Harvester Press (and Humanities Press).

Sloman, A. (1985). Real time multiple-motive expert systems. In Merry, M. (Ed.), *Expert Systems 85*, pages 1–13. Cambridge: Cambridge University Press.

Sloman, A. (1987). Motives mechanisms and emotions. *Cognition and Emotion*, 1(3):217–234. Reprinted in M.A.Boden (ed), *The Philosophy of Artificial Intelligence*, OUP, 1990.

Sloman, A. (1992). Prolegomena to a theory of communication and affect. In Ortony, A., Slack, J., & Stock, O. (Eds.), *Communication from an Artificial Intelligence Perspective: Theoretical and Applied Issues*, pages 229–260. Heidelberg, Germany: Springer.

Sloman, A. (1994). Explorations in design space. In *Proceedings 11th European Conference on AI*, Amsterdam.

Sloman, A. (1995a). Exploring design space & niche space. In *Proc. 5th Scandinavian Conf. on AI, Trondheim*, Amsterdam. IOS Press.

Sloman, A. (1995b). Poprulebase help file. Available at URL ftp://ftp.cs.bham.ac.uk/pub/dist/poplog/prb/help/poprulebase.

Sloman, A. (1995c). Rulesystems help file. Available at URL ftp://ftp.cs.bham.ac.uk/pub/dist/poplog/prb/help/rulesystems.

Sloman, A. (1995d). Sim_agent help file. Available at URL ftp://ftp.cs.bham.ac.uk/pub/dist/poplog/sim/help/sim_agent.

Sloman, A. (1995e). Sim_agent web-page. Available at URL http://www.cs.bham.ac.uk/ axs/cog_affect/sim_agent.html.

Sloman, A., Beaudoin, L. P., & Wright, I. P. (1994). Computational modeling of motive-management processes. In Frijda, N. (Ed.), *Proceedings of the Conference of the International Society for Research in Emotions*, Cambridge. ISRE Publications.

Sloman, A. & Croucher, M. (1981). Why robots will have emotions. In *Proceedings of the Seventh International Joint Conference on Aritificial Intelligence*, Vancouver.

Sloman, A. & Poli, R. (1995). Sim_agent: a toolkit for exploring agent designs. In Wooldridge, M., Mueller, J., & Tambe, M. (Eds.), *Intelligent Agents Vol II*, pages 392–407. Springer-Verlag.

Sloman, A., Shing, E., Read, T., & Beaudoin, L. (1992). Six types of design decision. Notes from a *Cognition and Affect* project meeting. Department of Computer Science, University of Birmingham.

Strongman, K. T. (1987). *The Psychology of Emotion*. John Wiley and Sons Ltd.

Wierzbicka, A. (1992). Defining emotion concepts. *Cognitive Science*, 16:539–581.

Wright, I. P. (1994). An emotional agent: the detection and control of emergent states in autonomous resource-bounded agents. Technical Report RP-94-21, School of Computer Science and Cognitive Science Research Centre, University of Birmingham.

Wright, I. P. (1996a). Design requirements for a computational libidinal economy. Technical Report CSRP-96-11, School of Computer Science and Cognitive Science Research Centre, University of Birmingham. Submitted to *Cognition and Emotion*.

Wright, I. P. (1996b). Reinforcement learning and animat emotions. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., & Wilson, S. W. (Eds.), *From Animals to Animats IV, Proceedings of the Fourth International Conference on the Simulation of Adaptive Behavior*, pages 272–281. The MIT Press.

Wright, I. P., Sloman, A., & Beaudoin, L. P. (1996). Towards a design based analysis of emotional episodes. *Philosophy Psychiatry and Psychology*, 3(2):101–137.