Representations for a Complex World: Combining Distributed and Localist Representations for Learning and Planning

Joscha Bach

University of Osnabrück
Department of Cognitive Science
jbach@uos.de

Abstract. To have agents autonomously model a complex environment, it is desirable to use distributed representations that lend themselves to neural learning. Yet developing and executing plans acting on the environment calls for abstract, localist representations of events, objects and categories. To combine these requirements, a formalism that can express neural networks, action sequences and symbolic abstractions with the same means may be considered advantageous. We are currently exploring the use of compositional hierarchies that we treat both as *Knowledge Based Artificial Neural Networks* and as localist representations for plans and control structures. These hierarchies are implemented using *MicroPsi node nets* and used in the control of agents situated in a complex simulated environment.

1. Introduction

Plan based control of agents typically requires the localist representation of objects and events within the agent's world model: to formulate a plan, individual steps have to be identified, arranged into sequences and evaluated. The ordering of the plan components asks for some kind of pointer structure that is usually expressed as a symbolic formalism. On the other hand, to have an agent act in a complex dynamic environment with properties and structure unknown to the agent, it is often desirable to use sub-symbolic representations of the environment that lend themselves to autonomous reinforcement learning. These demands result in hybrid architectures [11, 5], which combine symbolic and sub-symbolic layers of description. Usually, these layers are implemented with different techniques, for instance by defining a number of learnable low-level behaviors implemented in neural networks, which are switched and parameterized by a symbolic, non-neural layer.

In our approach, we make use of a different setting: we are using a kind of executable semantic network, called *MicroPsi node nets* [2] that can act both as feed-forward networks suitable for back-propagation learning and as symbolic plan representations. Even the control structures of our agents are implemented within the same networks as are their plans and their representations of the environment. This has a number of advantages: it is not necessary, for example, to draw a sharp

boundary between categorical abstractions and sensory-motor behavior. Rather, we express rules and abstractions as instances of localist neural network structures that may even be used to facilitate neural learning. We may thus mix distributed representations at all descriptional levels with rules, and we can also use rules at the lowest sensory-motor levels, if this is appropriate for a given task.

2. MicroPsi Agents

We are currently developing a cognitive architecture that is called *MicroPsi* [1] and focuses on the autonomous acquisition of grounded representations by agents, based on motivation. MicroPsi is partially derived from ideas of the "Psi"-theory of psychologist Dietrich Dörner [6, 7, 8]. Here, agents do not possess predefined knowledge of the world, but a set of predefined modes of access to it (i.e. sensors and actuators and some low-level processing for sensor data). Additionally, MicroPsi agents have a fixed set of motivational parameters (*urges*), which measure demands (like food or uncertainty reduction). In the pursuit of the demands, the agents' action control establishes consumptive goals (which consist in fulfilling of these demands) and directs the agents' behavior.

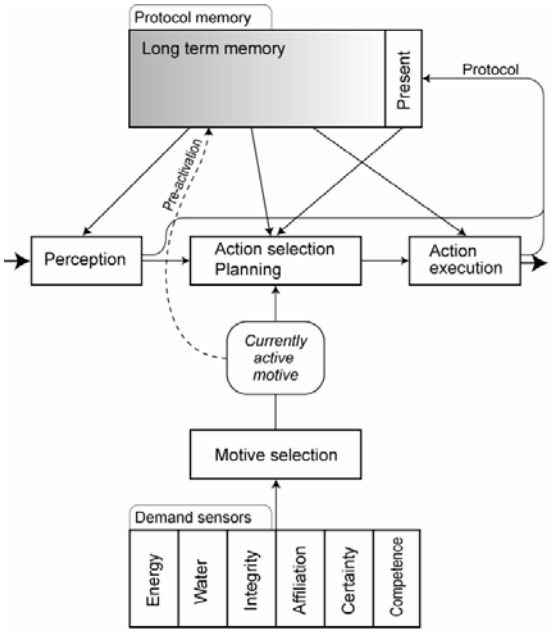

Fig. 1. Agent architecture as suggested by Dörner 2002 [8]

While some of the urges allude to physical needs (like food, water and integrity), there are also urges that are directed on cognitive aspects (like *competence*, i.e. the effectiveness in attaining goals, and *uncertainty reduction*, which measures the degree

of exploration of accessible environmental features). Dörner also suggests a social urge, called *affiliation*, which is directed at receiving positive socially interpreted signals from other agents. Each urge may give rise to a respective *active motive*, where the motive strength depends on the deviation of the demand from its target value, and the chance of selecting it is proportional to both the estimate of reaching a related goal and the intensity of the urge. Initially, the agents do not know which operations on the environment are effective in addressing the demands, so they have to resort to a try-and-error strategy. If actions have an effect (positive or negative) on the demands, a connection between the action, the surrounding perceived situation and the demand is established. If the respective urge signal gives rise to the related motive later on, it will pre-activate the associated situation and action context in the agent's memory. Thus, the urges may govern the behavior of the agent, based on its previous experiences.

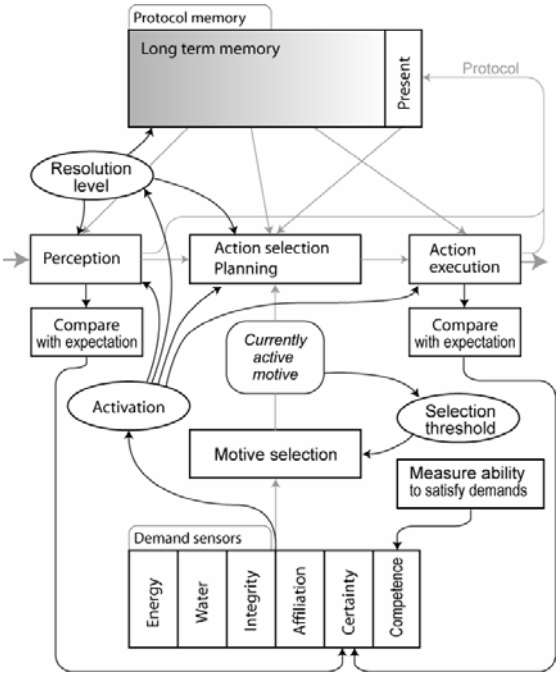

Fig 2. Influences of modulators in the Dörner model

Perception, action control, memory retrieval and planning may furthermore be modified by a set of modulating parameters: the *selection threshold* determines motive stability, the *resolution level* controls the accuracy and speed of perception and planning by affecting the number of features that are tested, the *activation* controls the action readiness and is basically inverse to the resolution level, and the *securing level* affects the ratio of orientation behavior. Together with the current urge changes (which are interpreted as pleasure or displeasure signals) and the competence and uncertainty levels, the state of the modulators might be interpreted as an *emotional configuration* of the agent.

2.1 MicroPsi Node Nets: Executable Spreading Activation Networks

Internally, MicroPsi agents are made up of a spreading activation *node net*. This section gives a (somewhat simplified) description:

$$NN = \langle U, V, DataSources, DataTargets, Act, \mathrm{f}_{\mathrm{net}} \rangle \tag{1}$$

$$U = \{(id, type, I, O, \mathrm{f}_{\mathrm{node}})\}, \mathrm{f}_{\mathrm{node}} : NN \rightarrow NN \tag{2}$$

$$I = \{(slotType, in)\}, in_{i_j^u} = \sum_{n=1}^{k} \omega_{v_n} out_n \tag{3}$$

$$O = \{(gateType, \alpha, out, \theta, min, max, amp, \mathrm{f}_{\mathrm{act}}, \mathrm{f}_{\mathrm{out}})\} \tag{4}$$

$$\mathrm{f}_{\mathrm{act}}^{u,o} : in_{u,i} \times \theta \rightarrow \alpha \tag{5}$$

$$\mathrm{f}_{\mathrm{out}} : \alpha \times Act \times amp \times min \times max \rightarrow out \tag{6}$$

$$V = \{(o_i^{u_1}, i_j^{u_2}, \omega, st)\}, st \in \mathbb{R}^4; st = (x, y, z, t) \tag{7}$$

Its building blocks, the net-entities U, are connected via weighted, directional links V. Net entities possess slots I (this is where links sum up their transmitted activation) and gates O, which take activation values from the slots and calculate output activations with respect to gate specific activators Act. Activators allow controlling the directional spread of activation throughout: there is an activator $act_{gateType} \in Act$ for each gate type, and since the node output function (6) is usually computed as $out = act_{gateType_o} amp \cdot \min(\max(\alpha, min), max)$, only gates with non-zero activators may propagate activation. Vectors of DataSources and DataTargets connect the net to its environment. The net entities come in several flavors:

- *Register nodes* have a single gate of type "*gen*" and a single slot, also of type "*gen*"; they are often used as simple threshold elements.
- *Sensor nodes* and *actuator nodes* provide the connection to the environment. Their activation values are received from and sent to the agent world (which can be a simulation or a robotic environment). Sensor nodes do not need slots and have a single gate of type "*gen*", which takes its value from a *DataSource*. Actuator nodes transmit the input activation they receive through their single slot (also type "*gen*") to a *DataTarget*. At the same time, they act as sensors and receive a value from a *DataSource* that usually corresponds with the actuator's *DataTarget*: The technical layer of the agent framework sends the respective *DataTarget* value to the agent's world-server which maps it to an operation on the world and sends back a success or failure message, which in turn is mapped onto the actuator's *DataSource*.
- *Concept nodes* are like register nodes; they have a single incoming slot, but in addition several kinds of outgoing links (i.e. types of gates). Each kind of links can be turned on or off to allow for directional spreading activation throughout the network, using the corresponding activator. Concept nodes allow the construction of partonomic hierarchies: the vertical direction is made of by the link type "sub", which encodes a part-whole relationship of two nodes, and the link type "sur", which encodes the reciprocal relationship. Horizontally, concept nodes may be connected with "por" links, which may encode a cause-effect relationship, or simply an ordering of nodes. The opposite of "por" links are "ret" links.

Additionally, there are link types for encoding categories ("cat" and "exp") and labeling ("sym" and "ref"). (Note that *link type* translates into a link originating from a gate of the respective type.)

- *Activator* and *associator nodes* are special entities that can be used to manage link weights, control directional spreading of activation and the values of activation of individual or groups of nodes in the network.
  It is possible to write arbitrary control structures for agents using these node types.

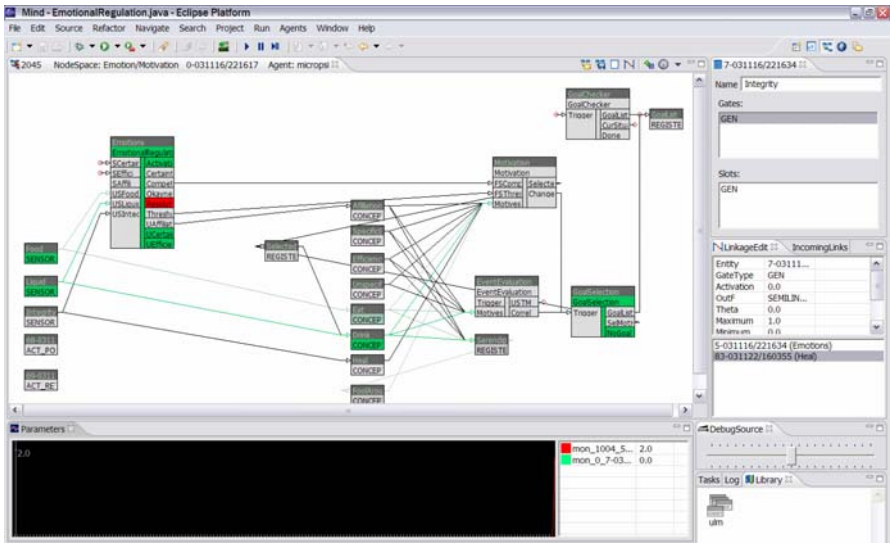

Fig. 3. MicroPsi node net editor

We have implemented a graphical editor/simulator to do this. However, we found that the manual design and debugging of large control programs using individual linked nodes is practically not feasible (trying to do this literally gives the notion of spaghetti code a new meaning). If a structure within the agent is not meant to be accessed as a collection of nodes, it is more straightforward to implement it using a programming language such as Java. This purpose is served by

- *Native modules* – this are net entities encapsulating arbitrary functions $\mathrm{f}_{\mathrm{node}}$ acting on the net, implemented in a native programming language, and executed whenever the module receives activation.

- *Node spaces* are net entities that encapsulate a collection of nodes:

$$S = \left\{U^S, DataSources^S, DataTargets^S, \mathrm{f}_{\mathrm{net}}^S\right\} \tag{8}$$

  Activator and associator nodes act only within the level of their node spaces. Because node spaces can contain other node spaces, agents may be split into modules, which make their innards a lot more accessible to human readers.

A slightly more detailed description of MicroPsi node nets is given in [2].

2.2 Situatedness of the Agents

Most of our experiments take place in a simulated environment that provides necessary resources and some hazards to the agents. Objects within the environment appear as co-located collections of features to the agents, where features correspond to sensory modalities of the agents. The simulation world is a plane consisting of different terrain types (the terrain has an effect on locomotion and may also provide a hazard, i.e. certain areas may cause damage to the agent). Different modes of locomotion are available to the agents, such as simple grid-based movement or a simulation of a pair of stepper motor driven wheels.

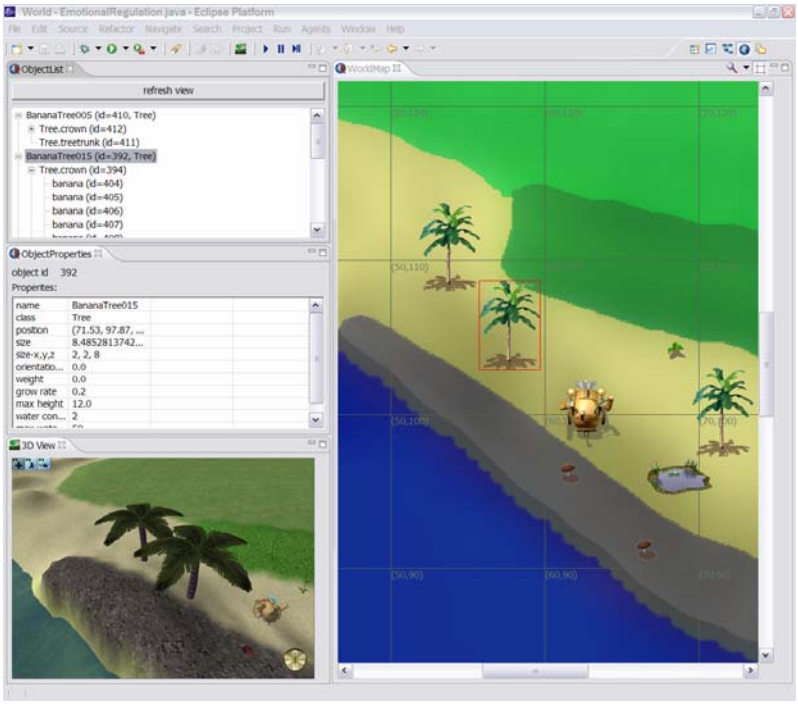

Fig. 4. World editor/simulator

In the past, we have presented agents with identifiers for each object, along with a spatial position relative to the agent. Currently, we start using abstract basic modalities, such as *Gestalt* identifiers, spatial extensions, relative positions, color values and weights, which might be replaced by a more low-level interface (like bitmaps and surface textures) in the future. Besides sensing, agents may probe their environment by acting upon it and examining the outcome. Actually, sensing may be seen as a form of action, and consequently, the definition of the sensory appearance of an object may amount to a script that encodes a sequence of actions necessary to recognize it.

Based on the affordances of the agents [9, 10], which are constrained by the sensory modalities and the needs of the agents, internal representations are derived from interactions with the environment.

Internally, world objects may be composed of sub-objects in spatial arrangements; the world maintains interactions between these objects by performing a discrete simulation, typically asynchronous to the agents. Agents, environment and computationally expensive world components (such as a simulation of plant growth) may run in a distributed network.

Using a constructed virtual environment for learning and classification experiments is not without difficulty: in many cases, agents may do nothing but rediscover a portion of the ordering that has been carefully engineered into their environment before, which limits the complexity of what is learned to what has been pre-programmed elsewhere. Additionally, the bias introduced by the artificial world may make it difficult to perform meaningful evaluations of the learning and classification results. On the other hand, because of shortcomings in perceptual and motor abilities, robots tend to be confined to a highly restricted and artificial environment as well. To put it a little provocatively: contemporary robots are often almost deaf, functionally blind, have restricted locomotion and only the simplest of push/grasp interactions available. It seems that many robotic testbeds (such as robotic soccer) can be quite satisfactorily simulated, and even significantly enhanced by incorporating additional modes of interaction and perception.

The simulation world is part of an integral experimental framework, along with the network simulator and a number of tools that aid in performing experiments. [3, 4]. (We are also using the toolkit for other agent designs, for instance in Artificial Life experiments, and to control Khepera robots.)

3. Representation Using Compositional Hierarchies

3.1 Partonomies

In MicroPsi agents, there is no strict distinction between symbolic and sub-symbolic representations. The difference is a gradual one, whereby representations may be more localist or more distributed. For many higher-level cognitive tasks, such as planning and language, strictly localist structures are deemed essential; in these procedures, individual objects of reference have to be explicitly addressed to bring them into a particular arrangement. However, a node representing an individual concept (such as an object, a situation or an event) refers to sub-concepts (using the “sub”-linkage) that define it. These sub-concepts in turn are made up of more basic sub-concepts and so on, until the lowest level is given by sensor nodes and actuator nodes. Thus, every concept acts as a reference point to a structured interaction context; symbols are grounded in the agent’s interface to its outer and inner environment.

There are several requirements to such a representation:

- *Hierarchies:* abstract concepts are made up of more basic concepts. These are referenced using "sub"-links (i.e. these sub-concepts are "part-of" a concept). Because these sub-concepts are in turn made up of sub-concepts as well, the result is a *compositional hierarchy* (in this case, a *partonomy*). For example, a hierarchy representing a face can be made out of a concept of a face, "sub"-linked to concepts for the eyes, the nose, the mouth etc. The concept for the eye points to concepts for eyelids, iris etc. until the lowest level is made up of primitive image sensors like local contrasts and directions. Note that this representation is devoid of categories, we are only representing individual instances of objects. However, if similar instances are encountered later on, the representation may act as a classifier for that object structure.

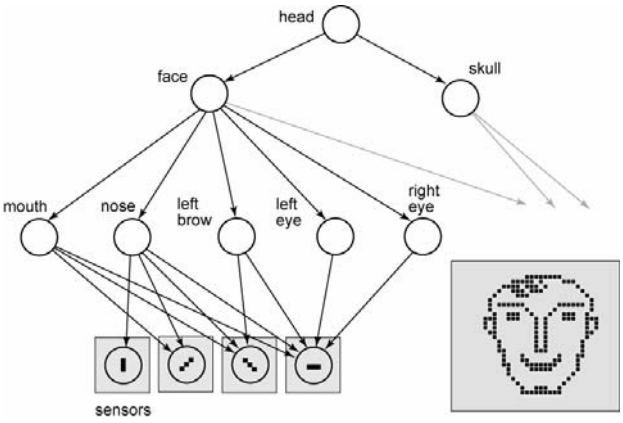

Fig. 5. Hierarchy of nodes, only "sub"-links are shown, reciprocal "sur"-links omitted

- *Sequences:* to encode protocols of events or action sequences, sequences of concepts need to be expressed. This is done by linking nodes using "por"-connections. "por" acts as an ordering relation and is interpreted as a subjunction in many contexts. The first element of such a "por"-linked chain is called the head of a chain and marks the beginning of execution on that level. In our face-example, the "sub"-linked parts of the face concept could be connected using spatially annotated "por"-links that define a plan to first recognize the left eye, then the right eye, then the nose, then the mouth. These sequences may occur on all levels of the hierarchy. The mouth-concept for instance might be made up of a sequence looking for the upper lip and the lower lip etc.

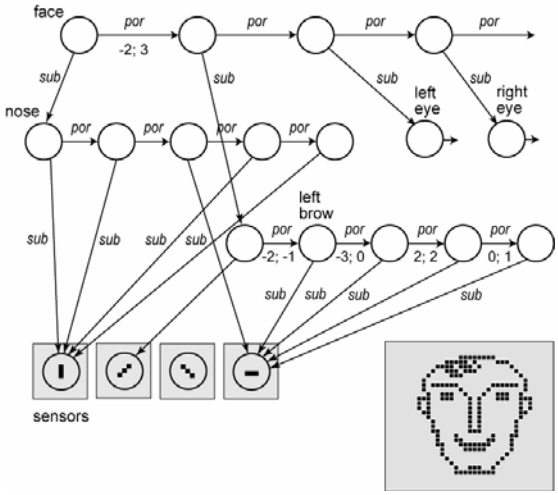

Fig. 6. Sequences in a hierarchy, reciprocal "sur" links and "ret" links omitted

- *Disjunctions:* Since there might be more than one way to reach a goal or to recognize an object, it should be possible to express alternatives. Currently this is done by using "sub"-linked concepts that are *not* "por"-linked, that is, if two concepts share a common "sur/sub" linked parent concept without being members of a "por"-chain, they are considered to be alternatives. This allows to link alternative sub-plans into a plan, or to specify alternative sensory descriptions of an object concept.

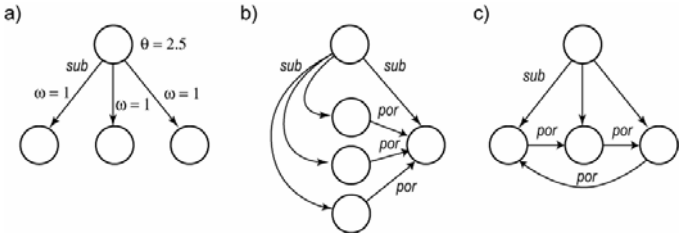

Fig. 7. Expressing conjunctions, reciprocal link directions ("ret" and "sur") have been omitted

- *Conjunctions:* in most cases, conjunctions can be expressed using sequences ("por"-linked chains), or alternatives of the same concepts in different sequence (multiple alternative "por"-linked chains that permute over the possible sequential orderings). However, such an approach fails if two sub-concepts need to be activated in parallel, because the parts of the conjunction might not be activated at the same time. Currently we cope with this in several ways: by using weights and threshold values to express conjunctions (fig. 7a), with branching chains (fig. 7b) or with reciprocal "por"-connections (fig. 7c). In the first case, we encode the relationship to the parent by setting the weights $\omega_{1..n,i}$ of the "sur/sub"-links from the alternatives $u_{1..n}$ to the parent u_i and a threshold value θ_i of u_i such that $\sum\omega_{1..n,i}$

$> \theta_i$ and $\sum\omega_{1..n,i} - \omega_{j,i} < \theta_i$ for all individual weights $\omega_{j,i}$ of an alternative $u_j \in \{u_{1..n}\}$. In the second case, we are using two "por"-links (i.e. two "por"-linked chains) converging onto the same successor node, and in the third, we are defining that fully "por"-connected topologies of nodes are given a special treatment by interpreting them as conjunctive.

- *Temporary binding:* because a concept may contain more than one of a certain kind of sub-concept, it has to be made sure that these instances can be distinguished. Linking a concept several times allows having macros in scripts and multiple instances of the same feature in a sensor schema. In some cases, distinguishing between instances may be done by ensuring that the respective portions of the net are looked at in a sequential manner, and activation has faded from the portion before it is re-used in a different context (for instance, at a different spatial location in a scene). If this can not be guaranteed, we may create actual instances of sub-concepts before referencing them. This can be signaled by combining partonomies with an additional link-type: "cat/ref", which is explained below. Note that sensors and actuators are never instantiated, i.e. if two portions of the hierarchy are competing for the same sensor, they will either have to go through a sequence of actions that gives them exclusive access, or they will have to put up with the same sensory value.

3.2 Taxonomic Relationships

If two different "por"-linked chains share neighboring nodes, and the relationship between these node is meant to be different in each chain (for instance, there is a different weight on the "por" and "ret" links, or the direction of the linkage differs, if they have different orderings in the respective chains), the specific relationship can not be inferred, because "por"-links are not relative to the context given by the parent. This can be overcome by making the chain structure itself specific to the parent, and linking the nodes to the chain structure via "cat/exp"-links (fig. 8).

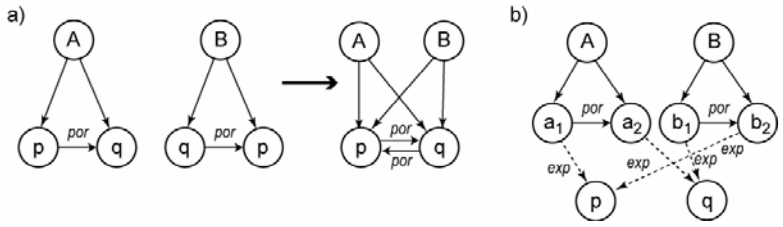

Fig. 8. a) Sharing of differently related features may lead to conflicts. b) Separating features and relationship with respect to parent

Thus, the structural intermediate node may hold activation values of the "exp"-linked actual concept, which itself may be used in other contexts as well. Of course, an intermediate node may have more than one "exp"-link. In this case, the linked concepts become interchangeable (element abstraction). The intermediate node may be interpreted as a category of the "exp"-linked concepts. Using "cat" and "exp"

links, it is possible to build taxonomic hierarchies. In conjunction with "sub" and "sur", MicroPsi node nets may be used to express hybrid or *parse structures* [12].

Within MicroPsi agents, "cat/exp" links are also used to reference different instances of the same concept, for instance in plans and in the local perceptual space. Here, "cat" links may act as pointers to the actual concepts in long term memory. "cat" may usually be interpreted as an "is-a" relationship.

3.3 Execution

Behavior programs of MicroPsi agents could all be implemented as chains of nodes. The most simple and straightforward way probably consists in using linked concept nodes or register nodes that are activated using a spreading activation mechanism. Conditional execution can be implemented using sensor nodes that activate or inhibit other nodes. Portions of the script may affect other portions of the script by sending activation to associator nodes or activator nodes. However, for complex scripts, backtracking and re-using portions of the script as macros become desirable.

For our purposes, a hierarchical script consists of a graph of options O, actions A and conditions C. Options might follow each other or might contain other options, so they can be in the relationships $succ(o_1,\ o_2)$, $pred(o_1,\ o_2)$ iff $succ(o_2,\ o_1)$, $contains(o_1,\ o_2)$ and $part\text{-}of(o_1,\ o_2)$ iff $contains(o_2,\ o_1)$. They might also be conjunctive: $and(o_1,\ o_2)$ iff $and(o_2,\ o_1)$, or disjunctive: $or(o_1,\ o_2)$ iff $or(o_2,\ o_1)$. The following restriction applies: $and(o_1,\ o_2) \vee or(o_1,\ o_2) \vee succ(o_1,\ o_2) \rightarrow \exists o_3$: $part\text{-}of(o_1,\ o_3) \wedge part\text{-}of(o_2,\ o_3)$.

Options always have one of the states *inactive*, *intended*, *active*, *accomplished* or *failed*. To conditions, they may stand in the relationship $is\text{-}activated\text{-}by(c,\ o)$, and to actions in $is\text{-}activated\text{-}by(o,\ a)$ and $is\text{-}activated\text{-}by(a,\ o)$. Options become *intended* if they are part of an *active* option and were *inactive*. They become *active*, if they are *intended* and have no *predecessors* that are not *accomplished*. From the state *active* they may switch to *accomplished* if all conditions they are *activated by* become *true* and for options that are *part of* them holds either, that if they are member of a conjunction, all their conjunction partners are *accomplished*, or that at least one of them is not part of a conjunction and is *accomplished* and has no predecessors that are not *accomplished*. Conversely, they become *failed* if they are *active*, one of the conditions they are *activated by* becomes *failed* or if all options that are *part of* them and are neither in *conjunctions* nor *successor* or *predecessor* relationships turn *failed*, or if they contain no options that are not in *conjunctions* or *successions* and one of the *contained* options becomes *failed*. And finally, if an option is *part of* another option that turns from *active* into any other state, and it is not *part of* another *active* option, it becomes *inactive*.

The mapping of a hierarchical script as defined above onto a MicroPsi node net is straightforward: options may be represented by concept nodes, the part-of relationship using "sub" links, the successor relationship with "por" links etc. (In order to use macros, "exp"-links have to be employed as discussed in section 3.2.)

Conditions can be expressed with sensor nodes, and actions with actuator nodes, whereby the activation relationship is expressed using "gen" links. Disjunctions

simply consist in nodes that share the same “sur” relationship, but are not connected to each other. This way, there is no difference between sensory schemas that are used to describe the appearance of an object, and behavior programs: a sensory schema is simply a plan that can be executed in order to try to recognize an object.

Even though the notation of a script is simple, to execute hierarchical scripts, some additional measures need to be taken. One way consists in employing a specific script execution mechanism that controls the spread of activation through the script. We have implemented this as a script execution module that will “climb” through a hierarchical script when linked to it (fig. 9).

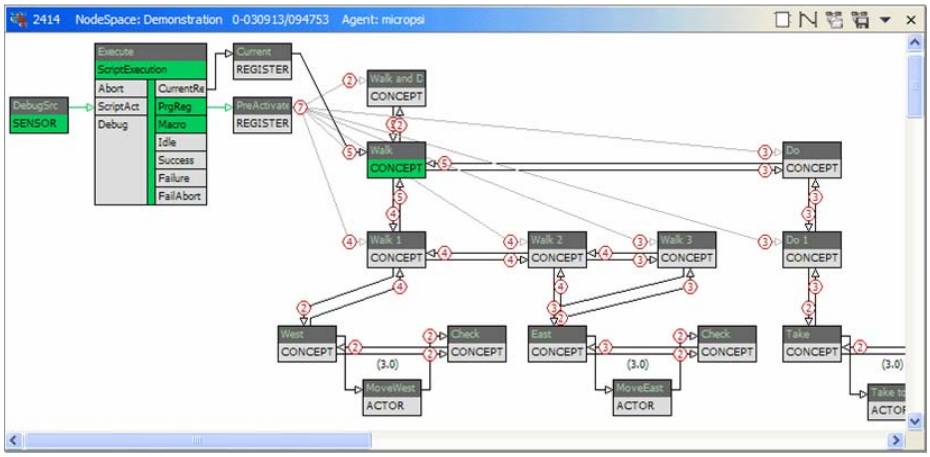

Fig. 9. Using a native module for script execution

Here, the currently active option is marked with a link and receives activation through it. “sub”-linked options get their *intended* status by a small amount spreading activation. By preventing this pre-activation from spreading (for instance by using inhibitory connections from outside the script), it is possible to block portions of the script from execution.

Actions are handled by sending activation into an actuator node and waiting for a specified amount of time for its response. If the actuator node does not respond with a success signal, the script will fail at the respective level and backtrack; backtracking positions are held in a stack that is stored within the script execution module.

The drawbacks of this approach are obvious:

- There is no parallel processing. Only one option is being activated at a time. In the case of conjunctive nodes, the activation focus is given to the one with the highest pre-activation first. If all conjunctive options have the same activation, one is randomly chosen.
- The activation of the individual nodes poorly reflects the execution state, which is detrimental to some learning methods (like decaying of rarely used links).
- The approach does not seamlessly integrate with distributed representations, i.e. it is for example not advisable to perform back-propagation learning on the

node hierarchy. (It is still possible to add lower, distributed layers that will be interpreted just like sensor and actuator nodes, though.)

On the other hand, it is also possible to devise a specific node type that spreads activation in the following manner: each node has two activation values, the *request activation* a_r, determining whether a node attempts to get confirmed by "asking" its sub-nodes, and a *confirm activation* a_c that states whether a node confirms to its parent concepts, where for each node: $0 \leq a_c \leq a_r$ (or $a_c < 0$ to signal failure). When a node gets first activated, it switches its state from *inactive* to *requested*. It then checks for "por"-linking neighbors (i.e. the corresponding slot): if it has no unconfirmed predecessors (i.e. nodes that possess a "por"-link ending at the current node), it becomes *requesting* and starts propagating its request activation to its "sub"-linked sub-concepts. In the next step, it switches to the state *wait for confirmation*, which is kept until its "sub"-linked children signal either confirmation or failure, or until their "sub"-linking parent stops sending a request signal. After confirmation, the node checks if it has "por"-linked unconfirmed successors. If this is not the case, a_c gets propagated to the "sub"-linking parent node, otherwise a_c is propagated to the successor node only. The node then remains in the state *confirmed* until its parent node stops requesting, then goes back to *inactive*. (Failures are propagated immediately.)

With this mechanism, we can describe conjunctions and disjunctions using weighted links. Since the execution of a script is now tantamount to pre-activating a hypothesis (the portion of the script we want to try) and its failure or success translates into a match with a sensor configuration, we may use the data structure for back-propagation and other neural learning methods. The distributed nature of execution makes supervision of the execution more difficult, but enables parallel distributed processing. (It should be mentioned that we can not use simple chains of "por"-linked nodes with this approach, without also "sub"-linking each of them to the same parent node. This is less of an issue for the script execution module, because it can determine the parent of each element of a sequence by parsing backwards along the "ret" links to the first element. But because this might take additional time in the case of backtracking, it seems always a good idea to declare the part-of relationship of each sequence element explicitly.)

4. Perception

The perceptual mechanism of the agents follows a bottom-up/top-down approach. If a sensor is triggered by some prominent environmental feature, activation spreads upward ("sur") and activates all concepts this sensor is part of. These concepts are then marked as perceptual hypothesis by linking them to an activation source. From now on, script execution is performed as described above: On each level of each of these concepts, activation is spreading down again to test the other parts deemed necessary to identify them. If the sensors connected to these other parts report success (i.e. find a corresponding feature in the environment), then the concept itself becomes confirmed, otherwise it will be marked as "failed". Thus, activation spreads upward,

"suggesting" theories of what is there to be perceived in the environment, then down, "testing" those theories, then up again, confirming or disconfirming them. The remaining confirmed concepts are considered the immediate percepts of the agent and are made available in a distinct node-space for further processing. (In Dörner's work, this mechanism is called "hypothesis based perception".)

4.1 Building Structured Representations of the Environment

Before partonomies can be employed to recognize objects, the agent has to construct them. There are several ways to do this. The first step, called *accommodation*, has been suggested by Dörner [6] and consists in using basic perceptual levels to arrive at a simple division of compositional layers: for instance by putting direction sensitive detectors for line elements at the lowest level. These make up contour segments, which are in turn parts of gestalts, and these may be combined into object schemas, which finally make up more complex objects and situations. However, the main problem turns out not to be the initial construction of such a simple partonomy, but its extension and modification, whenever new examples are encountered by the agent. In realistic environments, perception tends to be partial and somewhat uncertain, and different instances of a perceptual class may have somewhat different appearances. We combine two different approaches:

Whenever we come about a partonomic graph describing a percept that has not been encountered before in exactly the same way, we perform a graph matching procedure with existing object representations to obtain a similarity measure. For this, we are using the MatchBox algorithm [13] that estimates the best match according to node topology and link weights with a Hopfield network.

If we encounter a prototype with sufficient similarity, the new percept is merged with it by adding new nodes where necessary, and by adjusting link weights. If the percept does not exhibit much similarity to anything encountered before, a new prototype is created. (However, if the new object lends itself to similar interactions as another already known object, the perceptual descriptions might be merged as disjunctions in the future.)

The other method makes use of *Knowledge Based Artificial Neural Networks* (KBANN) [14, 15]. It works by using a partial, possibly incorrect domain theory that is given to the agent in the form of propositional logical clauses. These are converted into a hierarchical graph, where the lower level (sensor nodes and actuator nodes) are the precedents which are "sur"-linked to their antecedents. Conjunctions, disjunctions and negations are expressed by using weighted links and threshold values.

When activation is given to the sensor nodes according to the logical values of the antecedents, it spreads along the links and activates the compositional nodes higher up in the hierarchy in such a way as to evaluate the clauses.

The graph is then extended by a few randomly inserted nodes and additional sensor nodes to lend it more flexibility. These new nodes are given link weights close to zero, so they do not affect the outcome of the spreading activation. Furthermore, weak links are added across the hierarchy. The network is then used to classify examples encountered by the agent and is adapted using standard back-propagation. The

additional nodes allow the insertion of additional abstract features, and the additional links make generalization and specialization possible.

4.3 Learning Action Sequences and Planning

The planning capabilities of our agents are still very basic. To achieve at a repertoire of action sequences, they maintain a protocol memory made up of “por”-linked situations in the environment, whereby a situation consists of a spatial arrangement of recognized objects, along with actions performed on them. Whenever a situation is correlated with a positive or negative effect upon the urges of the agent (for example, the agent has reduced its need for water by collecting some), the links to the preceding situations are strengthened according to the impact of this event. Because in protocol memory, links below a certain strength decay over time, event chains leading to situations of importance to the agent (where it satisfied a need or suffered damage) tend to persist.

When the agent establishes a goal from an urgent need, it first looks for a direct chain leading from the current situation to the goal situation. If it can not find such an automatism, it attempts to construct such a chain by combining situations and actions from previous experiences, using a limited breadth-first search. If it does not succeed in constructing a plan, it resorts to either a different goal or to try-and-error behavior to create more knowledge.

5. Outlook

In MicroPsi agents, we combine neural network methods with compositional hierarchies that lead to localist, abstract representations of the environment suitable for planning. However, much work remains to be done. Currently, we are concerned especially with the representation of space and time and the implementation of an inheritance mechanism that allows distributing information along taxonomical hierarchies, as these seem to be a prerequisite for more generalized approaches to perception and memory.

While we are busy extending the simulation environment with more operators, more complex objects and relationships between them, we are planning to use the architecture more extensively with real-world sensor data and for the control of robots in the near future.

Acknowledgements

This work is the result of the efforts of many people, especially Ronnie Vuine, who supervised the implementation of large parts of the technical framework, the planning and the perception mechanism, Colin Bauer, who has implemented support for KBANNs and has designed the graph matching algorithm. Matthias Füssel has implemented the better part of the simulation environment, and David Salz is the creator of a 3D display component for the agent world.

References

1. Bach, J. (2003). The MicroPsi Agent Architecture Proceedings of ICCM-5, International Conference on Cognitive Modeling, Bamberg, Germany (pp. 15-20)
2. Bach, J., Vuine, R. (2003). Designing Agents with MicroPsi Node Nets. Proceedings of KI 2003, Annual German Conference on AI. LNAI 2821, Springer, Berlin, Heidelberg. (pp. 164-178)
3. Bach, J. (2003). Connecting MicroPsi Agents to Virtual and Physical Environments. Workshops and Tutorials, 7th European Conference on Artificial Life, Dortmund, Germany. (pp. 128-132)
4. Bach, J., Vuine, R. (2003). The AEP Toolkit for Agent Design and Simulation. M. Schillo et al. (eds.): MATES 2003, LNAI 2831, Springer Berlin, Heidelberg. (pp. 38-49)
5. Burkhard, H.-D., Bach, J., Berger, R., Gollin, M. (2001). Mental Models for Robot Control, Proceedings of Dagstuhl Workshop on Plan Based Robotics 2001
6. Dörner, D. (1999). *Bauplan für eine Seele*. Reinbeck
7. Dörner, D. (2003). *The Mathematics of Emotion*. International Conference on Cognitive Modeling, Bamberg
8. Dörner, D., Bartl, C., Detje, F., Gerdes, J., Halcour, (2002). *Die Mechanik des Seelenwagens. Handlungsregulation*. Verlag Hans Huber, Bern
9. Gibson, J. J. (1977). The theory of affordances. In R. E. Shaw & J. Bransford (Eds.), Perceiving, Acting, and Knowing. Hillsdale, NJ: Lawrence Erlbaum Associates
10. Gibson, J. J. (1979). The Ecological Approach to Visual Perception. Boston: Houghton Mifflin
11. Maes, P. (1990), Situated Agents Can Have Goals, Robotics and Autonomous Systems, 6 (p. 49-70)
12. Pfleger, K. (2002). On-line learning of predictive compositional hierarchies. PhD thesis, Stanford University
13. Schädler, K., Wysotzki, F. (1998). Application of a Neural Net in Classification and Knowledge Discovery. In: Michael Verleysen (ed.): Proc. ESANN'98, D-Facto, Brussels
14. Towell, G. Shavlik, J. (1992). Using symbolic learning to improve knowledge-based neural networks. In Proceedings of the Tenth National Conference on Artificial Intelligence, 177-182, San Jose, CA. AAAI/MIT Press
15. Towell, G. Shavlik, J. 1994. Knowledge-based artificial neural networks. Artificial Intelligence, 70 (p. 119-165)