Dynamic Schema Hierarchies for an Autonomous Robot

José M. Cañas[1] and Vicente Matellán[1]

Universidad Rey Juan Carlos, 28933 Móstoles (Spain)
{jmplaza,vmo}@gsyc.escet.urjc.es

Abstract. This paper proposes a behavior based architecture for robot control which uses dynamic hierarchies of small schemas to generate autonomous behavior. Each schema is a flow of execution with a target, can be turned on and off, and has several parameters which tune its behavior. Low level schemas are woken up and modulated by upper level schemas, forming a hierarchy for a given behavior. At any time there are several awake schemas per level, running concurrently, but only one of them is activated by environment perception. When none or more than one schema wants to be activated then upper level schema is called for arbitration.This paper also describes an implementation of the architecture and its use on a real robot.

1 Introduction

It is not science fiction to see a mobile robot guiding through a museum[1], navigating in an office environment, serving as a pet[2], or even playing soccer[3]. Today we've got best sensors and actuators ever. But how are they combined to generate behaviors? The hardware improvements have made clear the importance of a good control architecture, making it a critical factor to obtain the goal of autonomous behavior.

Robot control architecture can be defined as the organization of robot sensory, actuation and computing capabilities in order to generate a wide set of intelligent behaviors in certain environment. Architectures answer to a main question: what to do next? (action selection), as well as another questions like what is interesting in the environment? (attention), etc. Advances in architecture will lead to even more complex behaviors and increasing reliable autonomy.

In next section main proposals in robot control architectures will be reviewed. Section 3 presents our approach, Dynamic Schema Hierarchies, including its action selection mechanism, its reconfiguration abilities and how perception is organized in this proposal. The software architecture developed to implement DSH is commented on section 4. Some conclusions and future lines end the paper.

[1] Minerva: http://www.cs.cmu.edu/∼minerva
[2] Aibo:http://www.aibo.com
[3] Robocup: http://www.robocup.org/

2 Robot Control Architectures

2.1 Deliberative Architectures

Symbolic AI has influenced on mobile robotics from its beginnings, resulting in the deliberative approach. This makes emphasis on world modeling and planning as deliberation for robot action. In mid eighties this was the main paradigm for behavior generation. The control architecture was seen as an infinite information loop: Sense-Model-Plan-Act (SMPA). In modeling step sensor data are fused into a central world representation, which stores all data about environment, maybe in a symbolic form. Most robot intelligence lie in the planning step, where a planner searched in the state space and found a sequence operators to reach some target state from the current one. Act were seen as a mere plan execution.

There was a single execution flow and a functional decomposition of the problem, where the modules called functions from other modules (vision module, path planning module).

2.2 Behavior Based Architectures

Rooted in connectionist theories in mid eighties new approaches which exhibited impressive demos on real robots were proposed. The common factor of such works was the distribution of control in several basic behavior units, called levels of competence, schemas, agents, etc.

Each behavior unit is a fast loop from sensor to action, with its own partial target. There is no central representation, each behavior processes its own sensory information. Additionally, there are no explicit symbols about the environment. The emphasis is put in real world robots (embodiment) and in interaction with the environment (situated).

Distribution of control poses the additional problem of behavior coordination. Each behavior has its own goal, but usually enters in contradiction with another one. How is the final actuation calculated (action selection)?. There are two major paradigms: arbitration and command fusion. Arbitration establishes a competition for control among all the behaviors and only the winning one determines the final actuation. Priorities, activation networks from Pattie Maes [13] and state based arbitration [3] fall in this category. Command fusion techniques merge all the relevant outputs in a global one that take into account all behavior preferences. Relevant approaches in command fusion include superposition[2], fuzzy blending [15] and voting [14]. Coordination is always a difficult issue.

We will present here in more detail two foundational works leaving apart extensions and refinements. They contain main relevant ideas of the approach.

Brook's subsumption In 1986 Rodney Brooks proposed a layered decomposition of behavior in competence levels [6]. Since then many robots have been developed using this paradigm (Herbert, Toto), showing a great proficiency in low level tasks such as trash cans collecting, local navigation, etc. A competence level is an informal specification of a desired class of behaviors for a robot. Each

level is implemented by a net of Finite State Machines (FSM), which have low bandwidth communication channels to exchange signals and small variables.

Low levels provide basic behaviors, i.e. avoid obstacles, wander, etc.. More refined behaviors are generated building additional levels over the existing ones. All levels run concurrently, and upper levels can suppress lower level outputs and replace their inputs. This is called subsumption and gives name to the architecture. This action selection mechanism uses fixed priorities hardwired in the FSM net.

Arkin' schemas Following Arbib ideas [1], Ronald Arkin proposed a decomposition of behavior in schemas [2]. His architecture, named AuRA, contains two types of units: motor schemas and perceptive ones. "Each motor schema has an embedded perceptual schema to provide the necessary sensor information" [2].

For instance, the output of a navigation motor schema is a vector with the desired velocity and orientation to advance. The navigation behavior was obtained by the combination of avoid-moving-obstacles, avoid-static-obstacles, stay-on-path and move-to-goal schemas. Each schema can be implemented as a potential field, delivering a force vector for each location in the environment. The commanded movement is the superposition of all fields [2].

Extensions to Arkin's approach include the sequencing of several complex behaviors. A Finite State Acceptor is used for arbitration, where each state means the concurrent activation of certain schemas and triggering events are defined to jump among states [3].

2.3 Hybrid approaches

The trend in last years is an evolution to hybrid architectures that combine the strengths of both paradigms. For instance, planning capabilities and fast reactivity, because they both are important for complex tasks on real reliable robots.

A successful approach is the layered 3T-architecture [5], based on Firby's RAP [9]. The control is distributed in a fixed hierarchy of three abstraction levels that run concurrently and asynchronously. Upper layer includes deliberation over symbolic representations and makes plans composed of tasks. The intermediate level, called sequencer, receives such tasks and has a library of task recipes describing how to achieve them. It activates and deactivates sets of skills to accomplish the tasks. Skills compose the reactive layer. Each one is a continuous routine that achieve or maintain certain goal in a given context (it is situated).

3 Dynamic Schema Hierarchies

We propose an approach named Dynamic Schema Hierarchies (DSH) that is strongly rooted in Arbib [1] and Arkin ideas [2]. The basic unit of behavior is called schema. Control is distributed among a hierarchy of schemas.

An *schema* is a flow of execution with a target. It can be turned on and off, and accepts several input parameters which tune its own behavior. There are perceptual schemas and motor schemas. Perceptual ones produce pieces of information that can be read by other schemas. These data usually are sensor observations or relevant stimuli in current environment, and they are the input for motor schemas. Motor schemas access to such data and generate their outputs, which are the activation signal for other low level schemas (perceptual or motor) and their modulation parameters.

All schemas are *iterative* processes, they perform their mission in iterations which are executed periodically. Actually, the period of such iterations is a main modulation parameter of the schema itself. Digital controllers are an example of such paradigm, they deliver a corrective action each control cycle. Schemas are also *suspendable*, they can be deactivated at the end on one iteration and they will not produce any output until they are resumed again.

A perceptual schema can be in only two states: SLEPT or ACTIVE. When ACTIVE the schema is updating the stimuli variables it is in charge of. When SLEPT the variables themselves exist, but they are outdated. The change from SLEPT to ACTIVE or vice versa is determined by upper level schemas.

For motor schemas things are a little bit more tricky, they can be in four states: SLEPT, CHECKING, READY and ACTIVE. A motor schema has preconditions, which must be satisfied in order to be ACTIVE. CHECKING means the schema is awake and actively checking its preconditions, but they don't match to current situation. When they do, the schema passes to READY and tries to win action selection competition against other READY motor schemas in the same level and so become ACTIVE. Only ACTIVE schemas deliver activation signals and modulation parameters to lower level schemas.

Schemas can be implemented with many different techniques: simple rules from sensor data, fuzzy controllers, planners, finite state machines, etc. The only requirement is to be iterative and suspendable. In the case of a planner, the plan is enforced to be considered a resource, an internalized plan [14] instead of a symbolic one. This is because the schema has to deliver an action proposal each iteration.

3.1 Hierarchy

Schemas are organized in hierarchies. These hierarchies are dynamically built. For instance, if an ACTIVE motor schema needs some information to accomplish its target then it activates relevant perceptual schemas (square boxes in figure 1) in order to collect, search, build and update such information. It may also awake a set of low level motor schemas (circles) that can be useful for its purpose because implement right reactions to stimuli in the environment. It modulates them to behave according to its own target and put them in CHECKING state. Not only the one convenient for current situation, but also all the lower motor schemas which deal with plausible situations. This way low level schemas are recursively woken up and modulated by upper level schemas, forming a unique hierarchy for a given global behavior.

At any time there are several CHECKING motor schemas running concurrently per level, displayed as solid circles in figure 1 (i.e. schemas 5, 6 and 7). Only one of them per level is activated by environment perception or by explicit parent arbitration, as we will see on 3.2. The ACTIVE schemas are shown as filled circles in figure 1 (1, 6 and 15). For instance motor schema 6 in figure 1 is the winner of control competition at the level. It awakes perceptual schemas 11, 16 and motor schemas 14, 15, and sets their modulation parameters.

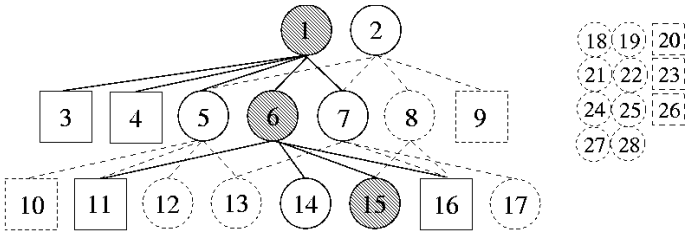

Fig. 1. Schema hierarchy and pool of SLEPT schemas

Schemas unused for current task rest in a pool of schemas, suspended in SLEPT state, but ready for activation at any time. They appear as dashed squares and circles in figure 1 (schemas 8, 9, 10, 12, 13, 17, 18, etc.). Actually, schemas 10, 12, and 13 are one step away from activation, they will be awaken if schema 5 passed to ACTIVE in its level.

Sequence of behaviors can be implemented with DSH using a motor schema coded as a finite state machine. Each state corresponds to one step in the sequence, and makes a different set of lower schemas to be awaken. It also activates the perceptual schemas needed to detect triggering events that change its internal state.

3.2 DSH Action Selection

At any time the ACTIVE perceptual schemas draw a perceptual subspace (attention subspace) that corresponds to all plausible values of relevant stimuli for each level (displayed as white areas in figure 2). It is a subset of all possible stimuli, because it doesn't include the stimuli produced by SLEPT perceptual schemas (shadowed area in figure 2). This subspace is partitioned into activation regions, which are defined as the areas where the preconditions of a motor schema are satisfied. Parent schema sets its child motor schemas activation regions to be more or less non overlapping.

A given situation corresponds to one point in such subspace, and may lie in the activation region of one motor schema or another. Only the corresponding motor schema will be activated, so situation activates only one schema per level among CHECKING ones. This is a coarse grained arbitration based on activation regions.

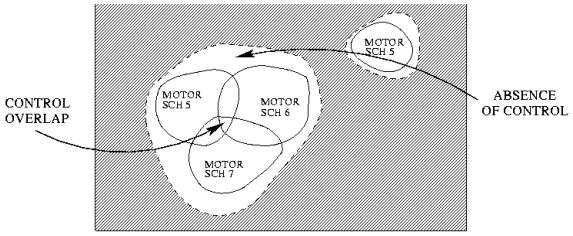

Fig. 2. Perceptual space, attention subspace and activation regions

Despite such coarse grained arbitration may appear situations where more than one motor schema for a level satisfy their preconditions (activation regions overlap in figure 2). Even situations where none of them are READY for activation, that is, not covered by any activation region (absence of control). Child schemas detect such control failures checking their brothers' state, and then parent is called for fine grained arbitration. Parent can change children parameters or just select one of them as the winner of control competition is its level. This is similar to context dependent blending [15], the parent schema knows the context for that arbitration, and so it can be very behavior specific.

The action selection mechanism in DSH has a distributed nature. There is no central arbiter as in DAMN [11], just parent and children. There is a competition per level, that occurs once every child iteration. This allows fast reconfiguration if situation changed.

It is a commitment between purposiveness top-down and reactive motivations. Only schemas awaken from upper level are allowed to gain control, but finally perceived situation chooses one and only one winner among them per level. The winner schema has double motivation, task-oriented and situation-oriented. Schemas without any of them don't add enough motivation for their activation and remain silent, CHECKING or SLEPT. Activation flows top-down in the hierarchy, similar to architecture proposed by Tinbergen and Lorenz [12] for instinctive behavior in animals. Addition of motivation and lateral inhibition among same level nodes also appear in such ethological architectures.

3.3 Perception

In DSH motor schemas make their decisions over information produced by perceptual schemas. Perception is distributed in perceptual schemas, each one may produce several pieces of such information. They exist because at least one motor schema eventually needs the information they produce. It can be sensor data that the schema collects, or more complex stimuli about environment or the robot itself built by the schema (for instance a map, a door, etc.). All the events or stimuli that we want to take into account in the behavior require a computational effort to detect and perceive them. DSH includes them in the architecture as perceptual schemas, each one searches for and describes its stimulus when present, updating internal variables.

Due to hierarchical activation in DSH, perception is situated, and context dependent. Perceptual schemas can be activated at will. The ACTIVE ones focus only on stimuli which are relevant to current situation or global behavior (attention). This filters out huge amounts of useless data from the sensors, i.e. shadowed area in figure 2. It makes the system more efficient because no computational resource is devoted to stimuli not interesting in current context.

There can be symbolic stimuli if they are convenient for the behavior at hand. These abstract stimuli must be grounded, with clear building and updating algorithms from sensor data or other lower level stimuli. Actually, different levels in perceptual schemas allow for abstraction and compounded stimuli: perceptual schemas can have as input the output of other perceptual schemas.

3.4 Reconfiguration

For a given task certain hierarchy generates the right robot behavior. The net of schemas builds relevant stimuli from sensor data and reacts accordingly to environment state. If the situation changes slightly, currently ACTIVE schemas can deal with it and maybe generate a slightly different motor commands. If the situation changes a little bit more maybe one ACTIVE schema is not appropriate anymore and the environment itself activates another CHECKING motor schema, that was ready to react to such change.

In the case of bigger changes they cause a control conflict at a certain level. In that case, the parent is called for arbitration and may decide to sleep useless schemas, wake up another relevant ones, change its children modulation, or propagate the conflict upwards forcing a hierarchy reconfiguration from upper level. The mechanism to solve conflicts may vary from one schema to another (fuzzy logic, simple rules, etc.). This reconfiguration may add new levels or reduce the number of them. This can be seen as a dynamic controller, composed of several controllers running in parallel and triggering events that change completely the controllers net. Each event requires a corresponding perceptual schema to detect it. This is similar to discrete state arbitration from [3], but accounts for hierarchy of schemas not only for a single level.

The levels are not static but task dependent as in TCA task trees [16]. These changes must be designed to work properly. Schemas don't belong to any level in particular. They can be located in a different level, probably with other parameters, for another global behavior. This way the schemas can be reused in different levels depending on the desired final behavior.

4 Implementation issues

We have used a small indoor robot, composed of a pioneer platform, and a off-the-shelf laptop under Linux OS (figure 3). The robot is endowed with a 16 sonar belt, bumpers, and two wheel encoders for position estimation. We have added a cheap webcam connected through USB to the computer. Two DC motors allow robot movements.

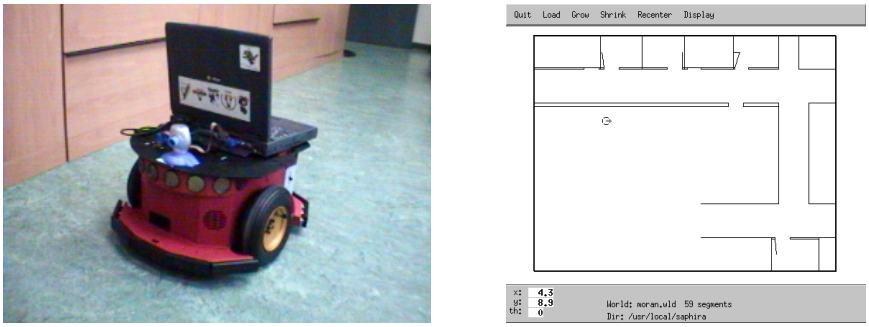

Fig. 3. Supercoco, our robot and Saphira© simulator

A software architecture has been developed to test the cognitive architecture proposed. All our code is written in ANSI-C. We have developed two socket servers which make available camera images, motor commands, raw sonar, bumper, and encoder data to client programs through message protocol. These servers can be connected both to robot simulator (Saphira environment [10]) or real platform, so the same control program can run seamlessly on real robot or on the simulator (without vision). The use of a simulator is very convenient for debugging. Additionally, the laptop is wireless connected, so the control program for real robot could run on-board, or in any other computer.

The DSH control program is a single client process, but with many kernel threads inside. Each schema is implemented as a thread (we use standard Posix threads on Linux) which periodically executes an iteration function. Communication between different threads is done through regular variables, because all the threads share virtual memory space. Motor schemas read variables updated by perceptual schemas, and low level schemas read their own parameters set by a higher level schema.

To implement SLEPT state we have used pthread `condition variables`. Each schema has an associated `condition variable`, so it can sleep on it when needed consuming no CPU cycles. Any schema may ask another one to be halted writing on a shared variable. The next time that recipient schema executes its iteration it will be suspended on its `condition variable`. The schema can be resumed when another thread signal on its condition variable. Typically the parent schema sets the child parameters and then wakes it up signalling on its condition variable.

Activation regions and arbitration are implemented as callback functions. Every iteration the motor schema calls its activation region function to check whether current situation matches its preconditions or not. Parent schema defines the activation region functions for each of its children. Additionally on each iteration the motor schema also checks the state of its brothers in the level to detect control overlaps or control absences. In such case, arbitration functions are invoked to solve the conflict.

4.1 Example

As an example we describe the `gotopoint` behavior developed using DSH: the robot reaches its destination point in the local environment and makes detours around obstacles if needed. It has been implemented as a parent motor schema that activates two perceptual schemas and three motor ones: `stop`, `followwall` and `advance`.

`Stop schema` stops the robot if obstacles are too close. This is the default schema in case of control absence. `Followwall schema` accepts sonar data and moves the robot parallel to closest obstacle. `Advance schema` moves it in certain orientation, faster if there are no obstacle in such angle, and turning to get it if needed. Activation regions are set in parent schema: if there is an obstacle closer than 100 cm in goal angle then `followwall schema` is activated, otherwise `advance schema` sends its motor commands. If a sudden obstacle gets closer than 20 cm then `stop schema` wins control competition.

First perceptual schema collects sonar and encoder data, and second one calculates distance to closest obstacle in goal angle. Actually it can calculate such distance in any orientation, but parent schema modulates it to do it in goal one. Parent schema read encoder data and computes relative distance and orientation to destination from current robot location. When detects the robot is over the target point it suspends all its child schemas.

`Followwall schema` alone makes the robot to follow walls. So this is an example of schema reusing. Used with a different activation region an together with other schemas can help to achieve another global behavior.

5 Conclusions

A new architecture named DSH has been presented, which is based on dynamic hierarchies of schemas. Perception and control are distributed in schemas, and grouped in abstraction levels. These levels are not general but task dependent which allow greater flexibility than fixed hierarchies. Perceptual schemas build relevant information pieces and motor schemas take actuation decisions over them in continuous loops.

It shares features with both deliberative and behavior based approaches. It may use symbolic stimuli when needed, directly grounded on sensor data or even on other stimuli, growing in abstraction. Also the absence of a central world model overcomes the SMPA bottleneck and avoids the need for such complete model before starting to act. The perception is task oriented, which avoid useless computation on non interesting data.

Timely reactions to environment changes are favored by low level loops and fast arbitration and reconfiguration capabilities. Deliberative schemas can be used too, but they are enforced to deliver an action recommendation each iteration. This prevents the use of plans as programs and enforces its use as resources for action.

A distributed arbitration is used for schema coordination. Each schema is the arbiter for its children, defining non overlapping activation regions. This

combines top down (target oriented) and bottom up (environment oriented) motivations for action selection.

The architecture is extensible. Adding a new schema is quite easy, it requires to define the schema parameters and its iteration, arbitration functions. Also all previous schemas can be reused as building blocks for new behaviors.

We are working to perceive more abstract stimuli, specially vision based (in particular doors) and to extend the schema repertoire with more abstract ones, as narrow door traversal.

References

1. Arbib, M.A., Liaw, J.S.: Sensorimotor transformations in the worlds of frogs and robots. Artificial Intelligence, **72** (1995) 53–79
2. Arkin, R.C.: Motor Schema-Based Mobile Robot Navigation. The International Journal of Robotics Research, **8(4)** (1989) 92–112
3. Arkin, R.C., Balch, T.: AuRA: Principles and Practice in Review. Journal of Experimental and Theoretical Artificial Intelligence, **9(2-3)** (1997) 175–188
4. Ali, K.S., Arkin, R.C.: Implementing Schema-theoretic Models of Animal Behavior in Robotic Systems. Proceedings of the 5th International Workshop on Advanced Motion Control, AMC'98. IEEE, Coimbra (Portugal) (1998) 246–253
5. Bonasso, R.P., Firby, R.J., Gat, E., Kortenkamp, D., Miller, D.P., Slack, M.G.: Experiences with an Architecture for Intelligent, Reactive Agents. Journal of Experimental and Theoretical Artificial Intelligence, **9(2)** (1997) 237–256
6. Brooks, R.A.: A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, **2(1)** (1986) 14-23
7. Corbacho, F.J., Arbib, M.A.: Learning to Detour. Adaptive Behavior, **5(4)** (1995) 419–468
8. Firby, R.J.: Building Symbolic Primitives with Continuous Control Routines. Proceedings of the 1st International Conference on AI Planning Systems AIPS'92. (1992) 62–69
9. Firby, R.J.: Task Networks for Controlling Continuous Processes. Proceedings of the 2nd International Conference on AI Planning Systems AIPS'94. AAAI (1994) 49–54
10. Konolige, Kurt: Saphira Software Manual. SRI International, 2001
11. Langer, D., Rosenblatt, J.K., Hebert, M.: A Behavior-Based System for Off-Road Navigation. IEEE Journal of Robotics and Automation, **10(6)** (1994) 776-782
12. Lorenz, K.: Foundations of Ethology. Springer Verlag (1981)
13. Maes, P.: How to Do the Right Thing. Connection Science Journal (Special Issue on Hybrid Systems), **1(3)** (1989) 291–323
14. Payton, D.W., Rosenblatt, J.K., Keirsey, D.M.: Plan Guided Reaction. IEEE Transactions on Systems Man and Cybernetics, **20(6)** (1990) 1370–1382
15. Saffiotti, A.: The uses of fuzzy logic in autonomous robot navigation. Soft Computing, **1** (1997) 180–197
16. Simmons, R.G.: Structured Control for Autonomous Robots. IEEE Transactions on Robotics and Automation, **10(1)** (1994) 34–43
17. Tyrrell, T.: The Use of Hierarchies for Action Selection. Journal of Adaptive Behavior, **1(4)** (1993) 387–420