A Logic-Based Approach to Model Supervisory Control Systems

Pierangelo Dell'Acqua*†, Anna Lombardi*, and Luís Moniz Pereira†

* Department of Science and Technology - ITN
Linköping University, 601 74 Norrköping, Sweden
{pier,annlo}@itn.liu.se

† Centro de Inteligência Artificial - CENTRIA
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. We present an approach to model supervisory control systems based on extended behaviour networks. In particular, we employ them to formalize the control theory of the supervisor. By separating the reasoning in the supervisor and the action implementation in the controller, the overall system architecture becomes modular, and therefore easily changeable and modifiable.

1 Introduction

Hybrid control systems [8] typically arise from the interaction of discrete planning algorithms and continuous processes and they represent the basic structure of discrete event supervisory controllers for continuous systems. Hybrid control systems have been object of research for more than a decade and there are well-established results. Several approaches to hybrid control systems have been defined in the literature, see e.g. [1, 6, 7] where examples of the most common structures are given. In supervisory control [5], the *system* (both plant and controller) consists of a family of subsystems that can be actuated at different time instants. Controller selection is carried out by means of logic-based switching. Switching among candidate controllers is performed by a high-level decision maker called a *supervisor*. In this way one level of abstraction is introduced making explicit the reasoning performed by the supervisor. The task of the controller is to execute the action selected by the supervisor and accordingly steer the plant.

Behaviour networks were introduced by Pattie Maes [9, 10] to address the problem of action selection in environments that are dynamic and too complex to be entirely predictable, and where the system has limited computational resources and time resources[1]. Therefore, the action selection problem cannot be completely rational and optimal. Maes adopted the stance suggestive of building

[1] See pp. 244-255 in [4] for a summary introduction.

intelligent systems as a society of interacting, mindless agents, each having its own specific competence [2, 11]. The idea is that competence modules cooperate in such a way that the society as a whole functions properly. Such an architecture is very attractive because of its distributiveness, modular structure, emergent global functionality and robustness [9]. The problem is how to determine whether a competence module should become active (i.e., selected for execution) at a certain moment. Behaviour networks addressed this problem by creating a network of competence modules and by letting them activate and inhibit each other along the links of the network. Global parameters were introduced to guide the activation/inhibition dynamics of the network. Behaviour networks combine characteristics of traditional AI and of the connectionist approach by using a connectionist computational model on a symbolic, structured representation. In a previous paper [3] we adapted the formalism of behaviour networks to make it possible to model hybrid control systems.

This paper proposes the use of extended behaviour networks to formalize the control theory of the supervisor in supervisory control systems. The reason for doing so is a consequence of the following observation. Extended behaviour networks are used in [3] to implement the controller itself. The resulting structure of the controller is rather complex as the controller has both the task of reasoning and the task of implementing the selected action. By separating the reasoning in the supervisor and the action implementation in the controller, the overall architecture becomes modular and therefore can be easily changed and/or extended. For example, if one new rule is introduced the only module that must be changed is the supervisor while the controller does not need any change.

2 Behaviour Networks

In this section we recap the approach to extended behaviour networks introduced in [3]. This approach extends the original framework of behaviour network proposed by Pattie Maes [9, 10] to allow rules containing variables, internal actions, integrity constraints, and modules (sets of atoms and rules).

A behaviour network is characterized by six modules: R, P, H, C, G and E. The module R is a set of rules formalizing the behaviour of the network, P is a set containing the global parameters, H is the internal memory of the network, C its integrity constraints, G its goals/motivations, and E the module that receives the input from the environment. We call the *state* of the network the tuple S=(R, P, H, C, G, E). We assume given a module Math containing the axioms of elementary mathematics.

2.1 Language

A *term* is a constant c or a variable x. Given a predicate symbol q of arity n and the terms $t_1, \ldots, t_n$, then $q(t_1, \ldots, t_n)$ is an *atom*. When the arity of q is 0, we write the atom as q. To express that an atom belongs to a module, we employ the notion of indexed atoms. In the following, we name the modules of

the network by using small letters. Thus, we name the modules R, P, H, C, G, E and Math as r, p, h, c, g, e and math. Let α be any arithmetic expression. Given a module m and an atom A, an *indexed atom* has the form $m{:}A$ or $m{\div}A$. The first states that A belongs to m, while the second states that A does not belong to m. We write sequences of indexed atoms by separating them with the symbol ',', and we write ε to indicate the empty sequence.

Both goals and integrity constraints are sequences of indexed atoms. A goal (motivation) expresses some condition to be achieved, while an integrity constraint represents a condition that must not hold. A *rule*[2] is a tuple of the form:

$$\langle prec; del; add; action; \alpha \rangle$$

where *prec*, *del* and *add* are sequences (possibly empty) of indexed atoms. *prec* denotes the preconditions that have to be fulfilled before the rule can become executable. *del* and *add* represent the internal effect of the rule in terms of a delete and add sequence of indexed atoms. When both *del* and *add* are empty (i.e., ε), then the rule has no internal effects. We assume that *del* and *add* contain only indexed atoms of the form $m{:}A$ with $m \neq e$ and $m \neq math$. The reason for this restriction is that the modules E and Math cannot be updated. In contrast, *prec* can contain any indexed atom.

The atom *action* represents the external effect of the rule: an action that must be executed. We employ 'noaction' to indicate that the rule does not have any external effect. Finally, each rule has a level α of strength[3]. Variables in a rule are universally quantified over the entire rule.

At every state S, a rule in R must be selected for execution. To do so, one needs to find all the rules that are executable and select one. A framework of rule selection for behaviour network is discussed in [3].

3 Case study: modelling an artificial animat

In this section we illustrate how to model a virtual animat by means of a supervisory control system. The supervisor of the system, on the basis of the input and output signals of the process, chooses which action to execute. Then the system activates the controller implementing the action selected by the supervisor. Thus, selecting an action to be executed corresponds in supervisory control system terms to select and activate a controller. We employ behaviour networks as a formalism (for the supervisor) to describe controller selection.

Scenario

Consider a virtual marine world inhabited by a variety of fish [12]. They autonomously explore their dynamic world in search for food. Fishes are situated within the environment, and sense and act over it. For simplicity, the behaviour of a fish is reduced to eating food, flocking and escaping from predators and is determined by the motivation of it being safe and satiated.

[2] In [10] rules are called competence modules.
[3] This value is used to calculate the activation level of the rules.

Supervisor

At every time instant t_k the supervisor receives information on the state of the fish, for example *hungry*(t_k), *fear*(t_k) and *distance_obstacle*(t_k). Some of the actions that the supervisor can select are: *searchFor(food)*: to search for food when it is hungry; *flee*: to run away from danger; and *flock*: to move in a flock. Assume that there exists a constraint `C={e:fear(x),math:x>5,h:searching(food)}` needed to prevent the fish to start searching for food in dangerous situations. The module G is {`h:safe, h:satiated`}, and R contains (among others) the following rules.

⟨`e:fear(x),m:x<0.2,h÷safe; ε; h:safe; noaction; 0.4`⟩

⟨`e:fear(x),m:x>0.5,h:flocking; h:safe,h:flocking; fleeing; flee(x*2); x*2`⟩

⟨`e:fear(x),m:x>0.5,h:alone; h:safe,h:searching(food); fleeing; flee(x*5); x*5`⟩

The first rule represents an internal action. It states that if the fish does not have fear, then it will add to its mental state that it is in a safe condition. Instead, the next two rules state that if the fish has fear, then it will flee with different values depending on whether it is flocking or alone. The level of strength of the last two rules is directly proportional to the amount of fear. The next three rules characterize the behaviour of the fish in case it gets hungry, and when it decides to flock.

⟨`e:hungry(x),m:x>0.5; h:satiated; h:searching(food); search(food); 0.5`⟩

⟨`e:hungry(x),m:x<0.2; h:searching(food); h:satiated; noaction; 0.5`⟩

⟨`h÷searching(food); h:alone; h:flocking; flock; 0.4`⟩

The last rule above has a non-monotonic flavour (via ÷). It says that if the fish is not searching for food, then it will start flocking. This is needed to give continuity to the fish behaviour.

Controller

The controller module contains all the controllers. Each controller implements an action selected by the supervisor. In the fish scenario, depending on the chosen action, the controller calculates the direction and velocity of the fish. For example, if the supervisor selects a rule whose action is `flee(0.6)`, then the corresponding controller can be described as:

`flee(fleeFactor)` {
$\overrightarrow{v}$ `= fleeFactor *` $\sum_{n=1}^{N} \frac{\overrightarrow{x}_{fish} - \overrightarrow{x}_n}{\|\overrightarrow{x}_{fish} - \overrightarrow{x}_n\|}$;
`return` $\overrightarrow{v}$; }

where N is the total number of visible predators, and $\overrightarrow{x}_n$ is the vector of the position of the n-th predator. We write $\|\overrightarrow{x}\|$ to indicate the norm of a vector $\overrightarrow{x}$. When a predator comes too close to the fish, the controller `flee` makes the fish flee in the opposite direction. The new direction is calculated wrt. the visible predators. `fleeFactor` determines the strength of the willing of the fish to escape. The velocity $\overrightarrow{v}$ is then passed to the plant where the new position will be calculated.

Plant

The artificial fish is modelled by several state variables representing the stimuli of the fish: e.g., the stimuli of hunger and fear. They are represented as variables with values in the range [0 1] with higher values indicating a stronger desire to eat or to avoid predators [12]:

- *hunger* : it expresses how hungry the fish is and it is approximated by:

$$H(t) = \min\{\Delta T \cdot \alpha, 1\} \tag{1}$$

  where ΔT denotes the time since the last meal, and α indicates the appetite of the fish.
- *fear* : it quantifies the fear of the fish by taking into account the distance $d(t)$ of the fish to a predator:

$$D(t) = \min\{D_0/d(t), 1\} \tag{2}$$

  where D_0 indicates how brave the fish is: if the fish is brave, it can take some risk by coming close to the danger, for example to reach the food.

Any time the plant receives the velocity $\overrightarrow{v}$ from a controller, the new position of the fish is calculated:

$$\overrightarrow{x}_{fish}(t+h) = \overrightarrow{x}_{fish}(t) + \overrightarrow{v} \cdot h$$

where h is the time interval between two computations. Finally, the new values for the fish stimuli are updated wrt. the new position $\overrightarrow{x}_{fish}(t+h)$.

4 Future Work

An interesting extension to the language L of the behaviour network would be to allow variables to occur in the strength levels of rules. This will allow defining the strength of a rule as a function of state.

One may also consider the possibility of including preference rules into a behaviour network with the aim of contributing to the action selection process. The idea being to compute all the executable rules whose activation level is above a certain threshold, and then to use preference reasoning to select the one that becomes active (and not just the one with greatest activation). This will

allow for additional flexibility, via a new threshold parameter, and an extra level of control by means of context sensitive preferences.

Our method affords a clear declarative, modular, and updatable rule based enactment of control, both at definitional and implementation levels, by separating rules for controller choice from the controllers themselves and the setting of their parameters. Flexibility and strategic leverage is gained in so doing, paving the way for controller monitoring, rule and parameter self-updating, and multi-level control. Furthermore, behaviour networks permit a more reactive or more deliberative form of controller choice, depending on the combination of the relative strengths of top-down and bottom-up "energy" propagations. In the future, meta-rules for updating the controller choice rules will be possible, either explicitly given or formed on the basis of learning or of pro-activeness through simulation.

References

1. Panos J. Antsaklis and Anil Nerode. Hybrid control systems: An introductory discussion to the special issue. *IEEE Transactions on Automatic Control*, 43(4):457–460, April 1998. Guest Editorial.
2. R. A. Brooks. A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation*, 2(1):14–23, 1986.
3. P. Dell'Acqua, A. Lombardi, and L. M. Pereira. Modelling Hybrid Control Systems with Behaviour Networks. In J. Filipe, J.-L. Ferrier, and J. A. Cetto, editors, *2nd Int. Conf. on Informatics in Control, Automation and Robotics (Icinco05). Procs. Intelligent Control Systems and Optimization Vol.1*, pages 98–105. INSTICC Press. ISBN:972-8865-29-5, 2005.
4. Stan Franklin. *Artificial Minds*. MIT Press, 1995.
5. Joao P. Hespanha, Daniel Liberzon, and A. Stephen Morse. Overcoming the limitations of adaptive control by means of logic-based switching. Systems and Control Letters. To appear.
6. W. Kohn and A. Nerode. Models for hybrid systems: automata, topologies, controllability and observability. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, LNCS 736, pages 317–356, Berlin, 1993. Springer-Verlag.
7. W. Kohn and A. Nerode. Multiple agent hybrid control architecture. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, LNCS 736, pages 297–316, Berlin, 1993. Springer-Verlag.
8. Feng Lin. Robust and adaptive supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, 38(12):1848–1852, December 1993.
9. Pattie Maes. How to do the right thing. *Connection Science Journal, Special Issue on Hybrid Systems*, 1(3):291–323, 1989.
10. Pattie Maes. A bottom-up mechanism for behavior selection in an artificial creature. In J. A. Meyer and S. Wilson, editors, *Proceedings of the first International Conference on Simulation of Adaptive Behavior*. MIT Press, 1991.
11. M. Minsky. *The Society of Mind*. Simon and Schuster, New York, 1986.
12. Xiaoyuan Tu. *Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior*. PhD thesis, ACM Distinguished Ph.D Dissertation Series, LNCS 1635, 1999.