MODELLING HYBRID CONTROL SYSTEMS WITH BEHAVIOUR NETWORKS

Pierangelo Dell'Acqua, Anna Lombardi
Department of Science and Technology (ITN) - Linköping University
601 74 Norrköping, Sweden

Luís Moniz Pereira
Centro de Inteligência Artificial (CENTRIA) - Departamento de Informática, Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Keywords: Behaviour networks, hybrid control systems, action selection, control architectures for autonomous agents.

Abstract: We present an approach to model adaptive, dynamic hybrid control systems based on behaviour networks. We extend and modify the approach to behaviour networks with integrity constraints, non-ground rules, internal actions, and modules to make it self-adaptive and dynamic. The proposed approach is general, reconfigurable, robust, and suitable for environments that are dynamic and too complex to be entirely predictable, the controlling system having limited computational and time resources.

1 INTRODUCTION

The term hybrid is accepted nowadays to denote systems whose behaviour is defined by processes of diverse characteristics. A survey on hybrid systems can be found in (Antsaklis and Nerode, 1998). In the control area, this signifies the combination of continuous and discrete dynamics, e.g. systems with signals that can take values from a continuous (real numbers) and, respectively, discrete (integer numbers) set. Some of the signals, can also be discrete-event driven, in an asynchronous way. Control systems of this kind, where continuous and discrete dynamics are modelled together, have proved effective in the computer control of continuous processes. They, in fact, represent the typical situation when controlling real systems, that is the system to be controlled is continuous-time while the controller works in a discrete way as it needs a time interval in order to compute the next system control input. Hybrid systems have always been studied both by computer science and control communities. Initially the work has been carried out separately and only recently the efforts have been put together, resulting in formal methods used to design intelligent control systems (Davoren and Nerode, 2000).

Behaviour networks were introduced by Pattie Maes (Maes, 1989) and (Maes, 1991) 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 intelligent systems as a society of interacting, mindless agents, each having its own specific competence (Minsky, 1986) and (Brooks, 1986). 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 distributedness, modular structure, emergent global functionality and robustness (Maes, 1989). 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 this paper we adapt the formalism of behaviour networks to make it possible to model hybrid control systems. In particular, we extend the language of behaviour networks to allow the competence modules contain variables. This feature makes it possible for the controller to receive the value of the parameters from a dynamic environment. Further, we introduce internal actions, and modules (sets of atoms and rules) in such a way that the network can test, and mod-

[1]See pp. 244-255 in (Franklin, 1995) for a summary introduction.

Dell'Acqua P., Lombardi A. and Moniz Pereira L. (2005).
MODELLING HYBRID CONTROL SYSTEMS WITH BEHAVIOUR NETWORKS.
In *Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics*, pages 98-105
DOI: 10.5220/0001188300980105

ify its global parameters (self-tuning) together with those of the competence modules defining its behaviour. Moreover, we introduce integrity constraints to prevent a network in a safe state to enter into an unsafe state (by executing some competence module).

The paper is structured as follows: Section 2 introduces the notion of hybrid control systems, Section 3 presents the language of extended behaviour networks and, for space limitation reasons, only sketches the idea of the algorithm for action selection. Section 4 describes how to model hybrid control systems by extended behaviour networks and provides a few examples. Finally, Section 5 discusses some future work.

2 HYBRID CONTROL SYSTEM

Several approaches to hybrid control systems have been defined in the literature, see e.g. (Antsaklis and Nerode, 1998) where examples of the most common structures are given. A hybrid control system can be seen as a switching system where the dynamics are described by a finite number of dynamical models (given in terms of differential or difference equations) together with a set of rules for switching among these models. These switching rules can be represented by logic expressions. A general hybrid control system is represented in Fig. 1 (Koutsoukos et al., 2000), (Antsaklis and Nerode, 1998).

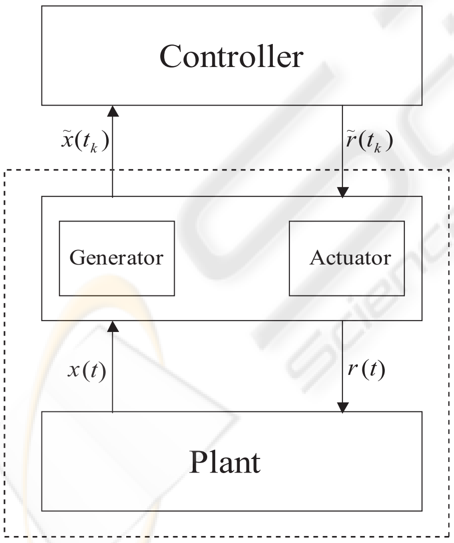

Figure 1: Hybrid control system.

The plant contains all the continuous dynamics including possible traditional controllers. The controller consists of a discrete-decision process described by a finite automaton.

The hybrid control system used in this work consists of a continuous system to be controlled - the plant - and a logic-based controller connected to the plant via an interface, in a feedback configuration.

A simple example of hybrid control system of this kind is the thermostat/furnace system (Franklin et al., 2002). The thermostat regulates the temperature in a room. The furnace and heat flow represent the continuous-time system to be controlled. The controller interacts with the continuous dynamics of the furnace to counteract the heat losses in order to keep the temperature within a desirable range. The thermostat is, therefore, an asynchronous discrete-event driven system which responds to the symbols: {too hot, too cold, normal}. The room temperature is translated into one of these symbols and the thermostat responds by sending electrical signals to the furnace. The voltage of the furnace is controlled and room temperature is increased or decreased, accordingly.

2.1 Plant

The plant is, in general, a nonlinear time-continuous system that can be described by a set of ordinary differential equations

$$\dot{x}(t) = f(x(t), r(t)) \tag{1}$$

where $x(t) \in \mathbf{X} \subset \Re^n$ denotes the state vector and $r(t) \in \mathbf{R} \subset \Re^m$ the input vector. For each fixed $r(t) \in \mathbf{R}$, the function $f(\cdot, r(t)) : \mathbf{X} \to \mathbf{X}$ is continuous in $\mathbf{X}$ and satisfies the conditions for existence and uniqueness of solutions for initial states $x_0 \in \mathbf{X}$.

In the thermostat/furnace hybrid control system introduced above, the plant consists of the furnace and the room. If $x(t)$ denotes the room temperature at time t in degrees Celsius, the plant equation (1) becomes (in a simplified model) (van Beek et al., 2003), (Koutsoukos et al., 2000)

$$\dot{x}(t) = -0.1x(t) + 0.4r(t)$$

where the control input $r(t)$ represents the voltage on the furnace control circuit (0 or 12V).

2.2 Controller

The controller is a discrete-time dynamic system described by a set of the difference equations

$$\begin{array}{rcl} \tilde{s}(t_{k+1}) & = & \delta(\tilde{s}(t_k), \tilde{x}(t_k)) \\ \tilde{r}(t_k) & = & \phi(\tilde{s}(t_k)) \end{array} \tag{2}$$

Tilde denotes representational symbols: plant events that go to the controller and actions sent from the controller to the plant. The functions δ and ϕ are defined in terms of a logic formalism that will be described in detail in Sec. 3.

Let the desired room temperature (T) in the thermostat/furnace example be set to 20 degrees. Then the plant event symbols sent to the controller are:

if room temperature is lower than 20 $\rightarrow$ *too cold*

if room temperature is higher than 20 $\rightarrow$ *too hot*

Each of the above event symbols will activate a rule of the controller that generates an action to be sent to the plant

- *too cold*: switch on the furnace. A voltage is applied to the furnace control circuit: $r(t) = 12V$
- *too hot*: switch off the furnace

The controller in this case has two states, i.e. the vector $\tilde{s}(t_k)$ has two components

$$\tilde{s}(t_k) = \begin{bmatrix} \tilde{s}_1(t_k) \\ \tilde{s}_2(t_k) \end{bmatrix}$$

The output equation in (2) becomes

$$\phi(\tilde{s}_1(t_k)) = \tilde{r}_1(t_k)$$
$$\phi(\tilde{s}_2(t_k)) = \tilde{r}_2(t_k)$$

where $\tilde{r}_1 \Leftrightarrow \mathit{on}$
$\tilde{r}_2 \Leftrightarrow \mathit{off}$

The controller is assumed to be adaptive in the sense that the parameters can change in response to variations of the environment.

2.3 Interface

Signals in the plant and in the controller are of different kind and therefore the plant and the controller need an interface to communicate with each other. The task of the interface is to translate the output of the plant (plant here includes also sensors used to measure quantities of interest) into symbols that can be understood by the controller and vice-versa. The conversion of the continuous-time output of the plant into symbols is performed by a *generator* (see Fig. 1)

$$x(t) \Rightarrow \tilde{x}(t_k)$$

The controller receives a symbol from the plant through the interface. In the example of thermostat/furnace, let the room temperature be measured and the value be 15 degrees. If the desired temperature is set to 20 degrees, the controller will determine the state *too cold* as active and will send a control signal to the furnace in order to switch on. The *actuator* will carry out this communication by performing

$$\tilde{r}(t_k) \Rightarrow r(t)$$

3 BEHAVIOUR NETWORKS EXTENDED

In this section we extend the approach to behaviour networks proposed by Pattie Maes (Maes, 1991) and (Maes, 1989) to allow rules containing variables, internal actions, integrity constraints, and modules (sets of atoms and rules). This will allow us to model hybrid control systems.

A behaviour network is characterized by five modules: R, P, H, C, and G. 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 and G its goals/motivations. We call the *state* of the network the tuple S=(R, P, H, C, G). We assume given a module Math containing the axioms of elementary mathematics.

3.1 Language L

Let c be a constant, q a predicate symbol of arity n, and x a variable. Then, terms and atoms in L are defined as follows:

term	:=	x | c
atom	:=	q(*term*$_1, \ldots,$ *term*$_n$)

When the arity of q is 0, we write the atom as q. To express in L that an atom belongs to a module, we introduce the notion of indexed atoms. We name (in L) the modules R, P, H, C, G, E and Math as r, p, h, c, g, e and math. Let $\alpha > 0$ be a real number.

*iAtom**	:=	h : *atom* | p : *atom* | g : *goal* | c : *ic* | r : *rule*
iAtom	:=	*iAtom** | e : *atom* | *math* : *atom*
niAtom	:=	h $\div$ *atom* | p $\div$ *atom* | g $\div$ *goal* | c $\div$ *ic* | r $\div$ *rule*
iAtomSeq	:=	*iAtom** | *iAtom*, iAtomSeq*
niAtomSeq	:=	*iAtom* | *iAtom, niAtomSeq* | *niAtom* | *niAtom, niAtomSeq*
goal	:=	*niAtomSeq*
ic	:=	*niAtomSeq*
rule	:=	$\langle$*prec*; *del*; *add*; *action*; α $\rangle$
prec	:=	ε | *niAtomSeq*
del	:=	ε | *iAtomSeq*
add	:=	ε | *iAtomSeq*
action	:=	*atom* | noaction

An iAtom of the form $m{:}X$ states that X belongs to the module M whose name is m, while an niAtom $m{\div}X$ states that X does not belong to M. An niAtomSeq is a sequence of iAtoms and niAtoms separated by the symbol ','. Note that a niAtomSeq may contain e:*atom* or $math$:*atom* while an iAtomSeq cannot. The reason for this is that the modules E and Math cannot be updated. Both goals and integrity constraints (ic) are niAtomSeq. A goal (motivation) expresses some condition to be achieved, while an integrity constraint represents a list of conditions that must not hold. A rule[2] is a tuple of the form:

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

[2]In (Maes, 1991) rules are called competence modules.

where *prec* is a sequence of preconditions (possibly, the empty sequence ε) 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 ε, then the rule has no internal effects. 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.

The following is an example of a rule. E and H are the modules representing the environment and the internal memory of the network.

$$\langle \texttt{h÷on,e:temp(x),math:x<20;}\ \varepsilon\texttt{; h:on; heating(on); 0.5}\rangle$$

The rule states that if the heating is off (not on) and the temperature x is less than 20, then the heating must be turned on. This is achieved by adding `on` to H (since `h:on` belongs to the add list of the rule), and by executing the action `heating(on)`. The strength of the rule is 0.5.

A *substitution* σ is a finite set of bindings of the form $variable/term$. A substitution can be applied to any expression X in L (written as X^σ) by simultaneously replacing any variable v in X with t for every binding v/t in σ. For example, if X is `h:q(x,y,z,c)` and $\sigma = \{x/y, z/d\}$, then X^σ is `h:q(y,y,d,c)`. A *ground* expression is one not containing variables. Two expressions X and Y in L are unifiable (written as $X \approx Y$) iff there exists a substitution σ such that $X^\sigma = Y^\sigma$, where $=$ denotes syntactic equality.

3.2 Rule Selection

At every state, 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. To determine whether a rule is executable, one needs to verify whether its preconditions (*prec*) are true at S. An niAtomSeq l is *true* at a state S iff:

- for every $m{:}X$ in l it holds that $X \in M$, and
- for every $m{\div}X$ in l there exists no substitution σ for which $X^\sigma \in M$.

A state S=(R,P,H,C,G) is *safe* if there exists no substitution σ that makes an integrity constraint in C true at S. Applying a rule $r = \langle prec; del; add; action; \alpha\rangle$ to a state $S = (M_1, \ldots, M_5)$ makes the system move to a new state $S' = (M'_1, \ldots, M'_5)$ obtained as follows. Every M'_i is obtained from M_i by removing X, if present, for every $m_i{:}X$ in del, and by adding Y for every $m_i{:}Y$ in add. We write $r(S)$ to denote the state obtained by applying a rule r to a state S. A rule $r = \langle prec; del; add; action; \alpha\rangle$ is *executable* at state S iff:

- $prec$ is true at S,
- $r(S)$ is a safe state, and
- $action$ is a ground atom.

An executable rule may be selected for execution. To select a rule we extend/modify the algorithm proposed in (Maes, 1989) to take into consideration variables (for space limitation reasons we only sketch the idea). We start by linking the rules in a network through three types of links: successor links, predecessor links, and conflicter links. Let x and y be rules.

- There is a successor link from x to y (x has y as successor) for every iAtom* $m{:}X$ in the *add* list of x and iAtom* $m{:}Y$ in the *prec* list of y such that $X \approx Y$.
- A predecessor link from x to y exists for every successor link from y to x.
- There is a conflict link from x to y for every iAtom* $m{:}X$ in the *prec* list of x and iAtom* $m{:}Y$ in the *del* list of y such that $X \approx Y$.

Rules use these links to activate and inhibit each other. Both the state S of the behaviour network together with the environment E, and the goals can spread activation among the rules through links. The basic idea is that there is input of activation energy coming from the state towards rules (forward propagation) whose preconditions partially match the current state, and from the goals towards rules (backward propagation) whose *add* lists partially match the goals. Furthermore, there is an inhibition by the goals that have already been achieved (protected goals). These goals remove some activation energy from the rules that would undo them.

Besides the activation energy from the state and goals, rules also inhibit and activate each other along the links in the network. The mathematical model for computing the activation level of rules is based on the local strength α of a rule and on several global parameters that are used to tune the spreading of activation energy through the network. There exists a parameter θ specifying the threshold of rules for becoming active, ϕ the amount of energy that a proposition that is true injects into the network, ψ the amount of energy that a goal injects into the network, and δ the amount of energy that a protected goal takes away from the network.

Let $r \in R$ be a rule and σ a substitution. The rule r^σ becomes *active* when: it is executable, its level of strength overcomes θ, and its activation level is higher than the activation level of all other executable rules. Note that only one rule can become active. In case there are several executable rules with the same activation level, then one is randomly selected to become active. When an active rule has been executed, then its activation level is reinitialized to 0. If none of the

[3]This value is used to calculate the activation level of the rules in Sect. 3.2.

rules becomes active, then the threshold θ is lowered by a certain factor.

4 MODELLING HYBRID CONTROLLERS

An adaptive, dynamic hybrid controller can be described by an extended behaviour network. Figure 2 illustrates an hybrid controller consisting of a control unit (CU) and the modules R, P, H, C, G and Math. Besides the modules, the CU of the controller is connected to an external module E containing discrete values from the environment, and to the Actuator.

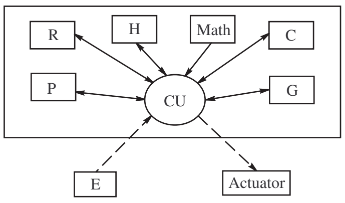

Figure 2: Hybrid controller.

4.1 Control Unit Engine

The basic engine of CU can be described via the following cycle:

```
Cycle(n, R, P, H, C, G)
1. Load the rules of R into CU and calculate their activation level wrt. the global parameters in $P$.
2. If one rule becomes active, then execute its internal effect, and send its external effect (its action) to the actuator. Let R', P', H', C', G' be the modules after the execution of the rule.
   Cycle with (n+1, R', P', H', C', G').
3. If no rule becomes active, then lower the level of $\theta$ in $P$.
   Cycle with (n+1, R, P', H, C, G).
```

Initially, given the modules R, P, H, C and G, the cycle starts with (1, R, P, H, C, G). We assume that the initial state S=(R, P, H, C, G) is safe. First, the rules of R are loaded into CU and the activation level of each rule is calculated. Then, the rule that becomes active is executed. Its internal effect makes the state of the behaviour network to change, and its external effect (i.e., its action) is sent to the Actuator to be executed. If no rule is active, then the controller cycles by letting all the modules unchanged except P whose value θ (the threshold of rule activation) is lowered.

4.2 Artificial Fish

Consider a scenario where we have a virtual marine world inhabited by a variety of fish. They autonomously explore their dynamic world in search for food. Hungry predator fish stalk smaller fish who scatter in terror. Fish are situated within the environment, and sense and act over it. For simplicity, the behaviour of a fish is reduced to eating food, escaping and sleeping, and is determined by the motivation of it being safe and satiated. The following extended behaviour network models an artificial fish. We employ the module E to represent the stimuli of the fish. The stimuli of hunger and fear are variables with values in the range $[0\ 1]$ with higher values indicating a stronger desire to eat or to avoid predators (Tu, 1999):

- *hungry*: it expresses how hungry the fish is and it is approximated by
$$\text{hungry}(t) = \min\{1 - f(t)r(\Delta T/n^{\alpha}, 1\}$$
where f denotes the amount of food consumed, ΔT the time since the last meal and n^{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 visible predators
$$\text{fear}(t) = \min\{D_0/d(t), 1\}$$
- *tired*: it contains information on whether the fish is tired. It is a boolean variable that becomes T=*true* every 3 hours.

The input vector to the controller is

$$\tilde{x}(t_k) = \begin{bmatrix} \textit{hungry}(t_k) \\ \textit{fear}(t_k) \\ \textit{tired}(t_k) \end{bmatrix}$$

The actions the fish can make are:

- *searchFor(food)*: it searches for food when it is hungry;
- *eat(food)*: once it has found food, it eats the food;
- *sleep*: when it is tired;
- *escape*: when it is in danger.

The module H represents the internal state of the fish. The fish can have food, can be satiated or can be safe.

$$\tilde{s}(t_k) = \begin{bmatrix} \textit{food}(t_k) \\ \textit{satiated}(t_k) \\ \textit{safe}(t_k) \end{bmatrix}$$

Assume that there exist no constraints, therefore C={}. The module G is {`h:safe, h:satiated`}, and R consists of:

$\langle$ `e:hungry(x), math:x>0.5, h÷food;` ε`;`
`h:food; searchFor(food); 0.5`$\rangle$

```
⟨ e:hungry(x), math:x>0.5, h:food; ε;
          h:satiated; eat(food); 0.5⟩
             ⟨ e:tired; ε; ε; sleep; 0.5⟩
   ⟨ e:fear(x), math:x>0.5; ε; h:safe;
                         escape; 0.7⟩
```

The first rule states that if the fish is hungry and it does not have food, then it will search for it. Note that when the fish receives only one stimulus, its behaviour is completely determined. When it is hungry, then it will search for food or it will eat, depending on whether or not it has food. Things change if the fish receives several stimuli simultaneously, e.g., tired and hungry. Then the fish has a competing alternative action, to sleep. The activation levels of the rules determine which one will become active. Suppose that the fish does not have food. In this case, it is likely that the first rule will become active since it will receive energy of G via backward propagation. In fact, the second rule will receive energy of G since its add list partially matches G, this rule in turn will propagate backward energy to the first rule since the add list of the first rule and the precondition of the second one both contain the iAtom `h:food`.

Finally, suppose that, besides the first two stimuli, the fish receive also the stimulus for fear. In this case, G cannot determine which rule will become active since all rules (except for the third one) receive energy backward from G. Now, to increase the chance that the last rule will become active, one can give more local strength to it.

4.3 Home Environment

Development in home automation has made people dream for long of a *smart home* where household devices behave in an intelligent way. Technologies have developed since the first ideas appeared and the smart home is becoming a reality. Ambient intelligence is an area of study where the environment is aware of the presence of people and is adaptive, proactive, and responsive to their needs. It is very easy to figure out such a scenario: inhabitants can ring home and program the heating to start at a given time so that they find it warm when they return. Great attention is devoted to ambient intelligence systems nowadays; in a project supported by the European Community, ISTAG,[4] a formal definition of ambient intelligence is provided. Ambient Intelligence should provide technologies to support human interactions and to surround users with intelligent sensors and interfaces. One of the main reasons that has prevented the implementation of ambient intelligence on a large scale is the high cost in programming. The system should be adjusted to the needs of each home and the inhabitants are not willing to learn to program it themselves. They would like simply a system capable to respond to their needs. Many approaches have been proposed aiming at developing such systems. In (Mozer, 1998) the goal is that the home programs itself on the basis of a neural network learning method. It observes the lifestyle and desires of the inhabitants and it learns to anticipate and accommodate their needs. (Hagras et al., 2004) focuses, instead, on developing learning and adaptation techniques for embedded agents. Embedded agents are capable of reasoning, planning, and learning and they can communicate with each other. In this context each embedded agent is connected to sensors and actuators so they can modify actuators on the basis of input vectors. In (Davidsson and Boman, 2000), (Rutishauser et al., 2005) different categories of agents are defined resulting in a multi-agent system with agents toiling in a concurrent way. Each category relates to applications in the environment being monitored and controlled: personal comfort, environmental parameters, and so on.

Let us now apply the approach proposed in the previous sections to a particular case of ambient intelligence. Assume we have a hybrid control system of the kind described in Sec. 2 and represented in Fig. 1. The controller is implemented as extended behaviour networks, as defined in Sec. 3. The goal is to keep the home comfortable with a temperature set at a desired value T, and safe by monitoring the fire detector. If fire is detected then the sprinkler system is activated. The water will flow through the sprinkler heads into the rooms only after the power supply has been switched off. The generator in the interface receives information from the sensors and sends the input vector to the controller with the following symbols:

- *temp*: room temperature at time t_k;
- *people*: detection if there are people at home at time t_k; it is a boolean variable assuming the values T=*true* if there are people and F=*false* otherwise;
- *fire*: fire detection at time t_k; it is a boolean variable assuming the values T=*true* if fire is detected and F=*false* otherwise.
- *alarm*: burglary alarm detection at time t_k; it is a boolean variable assuming the values T=*true* if an intruder is detected and F=*false* otherwise.

The input vector of the controller is given by

$$\tilde{x}(t_k) = \begin{bmatrix} temp(t_k) \\ people(t_k) \\ fire(t_k) \\ alarm(t_k) \end{bmatrix}$$

The controller will act on the heating system by switching it on/off depending on the value of the room temperature and the desired temperature. The controller will also switch off the power supply in case of fire and subsequently the sprinkler system will be activated. In case an intruder breaks in, the controller will switch the alarm bell on if there are people at

[4]see `http://www.cordis.lu/ist/istag.htm` for further details.

home. Otherwise, the controller will load a library of rules that specifically handles the event (e.g., handling the call to a Security Center). The actions produced by the actuators in the interface can be summarized as:

- *heating(on/off)*: set the voltage of the furnace control system to 12V or 0V;
- *power(off)*: switch off the power supply;
- *sprink(on/off)*: activate/disactivate the sprinkler system;
- *alarmBell(on/off)*: activate/disactivate the burglary alarm bell.

The module H represents the state vector of the controller:

$$\tilde{s}(t_k) = \begin{bmatrix} \textit{heat}(t_k) \\ \textit{el}(t_k) \\ \textit{w}(t_k) \\ \textit{bell}(t_k) \end{bmatrix}$$

where *heat* indicates whether the heating is on or off, *el* gives information on the power supply, *w* on the sprinkler system, and *bell* on the alarm system. The state with power supply on and the sprinkler system activated is not safe and therefore must be avoided. Initially, it is assumed that the heating is on, the power supply is on while the sprinkler system is disactivated, and the alarm bell is off. The state at initial time t_0 is

$$\tilde{s}_0 = \tilde{s}(t_0) = \begin{bmatrix} h\text{:}\textit{heat(on)} \\ h\text{:}\textit{el(on)} \\ h\div\text{h(on)} \\ h\div\text{bell(on)} \end{bmatrix}$$

Note that the initial state is safe. The controller behaviour is determined by the set of modules defined in Sec. 3. The integrity constraint module C contains the following sequence

`C={h:el(on),h:w(on)}`

Meaning it is not allowed to have both the power supply on and the sprinkler system activated, as this engenders a state that is not safe. The following rules are contained in the module R:

```
⟨ h÷heat(on),e:temp(x),math:x<T,
   e:people; ε; h:heat(on); heating(on);
                                  0.7 ⟩

⟨ h:heat(on),e:temp(x),math:x>T;
      h:heat(on); ε; heating(off); · ⟩

⟨ e:fire; ε; h:w(on); sprink(on); 0.9 ⟩

⟨ e:fire; h:el(on); ε; power(off); 0.9 ⟩

⟨ e÷fire,h:w(on); h:w(on); ε;
                      sprink(off); 0.7 ⟩
```

The first two rules decide whether the heating should be switched on. This depends on whether the temperature measured by the sensor is smaller or greater than the desired temperature T. The last three rules, instead, decide what should be done if fire is detected. The actions to be taken in case of fire are of high importance and therefore these three rules have a high strength level (set to 0.9). The third rule says that if fire is detected then the sprinkler system must be activated. This is not allowed by the integrity constraint module C. In this case the fourth rule is necessary, which switches off the power supply in case of fire. Finally, the fifth rule states to disactivate the sprinkler system once the fire has been switched off.

The following rules illustrate a situation where the controller can dynamically change its own rules in R.

```
⟨ e:alarm,e:people; ε; h:alarm(on);
              alarmBell(on); 0.8 ⟩

⟨ e:alarm,e÷people; ε;
   r:load(alarmLib); noaction; 0.8 ⟩
```

The last rule says to load a library of rules into R to handle a burglary alarm situation in case no one is at home. Those rules will be loaded into CU and used at the next CU cycle.

5 DISCUSSION

In this paper we have extended and modified the formalism of behaviour networks to make it suitable to model adaptive, dynamic, hybrid control systems. Because of space limitations, we have only sketched the mathematical model for calculating the activation level of rules to address the problem of rule selection. Currently, we are developing an implementation for the extended behaviour networks by using XSB Prolog (XSB-Prolog, 2004) to implement the part based on the unification algorithm, and Java (Java Technology, 2004) to implement the part calculating the activation level of rules. We are going to integrate them via Interprolog (InterProlog, 2004). Finally, we are going to field test the system, in particular the model for calculating the activation level of rules, on a number of computer simulations, and to compute complexity results.

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. Clearly, this will be computationally more expensive.

```
⟨ h÷on,e:temp(x),math:x<20; ε; h:on;
           heating(on); 0.5*(20-x)⟩
```

This rule states that if the heating is not on and the temperature is below 20 degrees, then we need to switch the heating on. The strength level of the rule depends on current temperature `x`. The bigger the difference `20-x`, the bigger the strength level of the rule, and consequently the better is the chance of the rule to become active.

We are also considering the possibility to include preference rules into a behaviour network with the aim of contributing to the action selection process. The use of preference rules has been extensively studied in Logic Programming, both on the theoretical and practical side. The idea is 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 allows for more flexibility, with a new candidate threshold parameter, and an extra level of control with context sensitive preferences. Moreover, these could be updated by the system (by allowing preference rules in R).

Other techniques developed by the Logic Programming community could be applied here. For example, belief revision techniques could be used to resolve cases of conflicting rules when more than one are allowed to become active (at the moment only one rule can become active and consequently only one action at a time can be sent to the actuator); and rule update techniques in the spirit of EVOLP (Alferes et al., 2002). The generalization of the language L to full EVOLP would allow for non-deterministic evolutions (chose one arbitrarily or according to some probability). Further, genetic algorithms could be used to tune the global parameters of the network to select the most effective action selection from a population. In this way, a set of parameters can be evolved instead of being tuned by hand (see (Singleton, 2002) for a discussion).

Finally, to tackle the problem of modelling very complex environments we may design and construct networks of behaviour networks, either with hierarchical or distributed structure, or even behaviour networks that fight on another to acquire control.

For example, in a scenario where we need to control a complex building consisting of several floors, we may employ a number of behaviour networks, each controlling a different apartment at every floor, and then organize them in a hierarchical network where the behaviour networks higher up in the hierarchy have the role of supervising those at lower levels.

REFERENCES

Alferes, J. J., Brogi, A., Leite, J. A., and Pereira, L. M. (2002). Evolving logic programs. In *Proceedings of the 8th European Conf. on Logics in Artificial Intelligence (JELIA'02), LNCS 2424*, pp. 50–61.

Antsaklis, P. J. and Nerode, A. (1998). Hybrid control systems: An introductory discussion to the special issue. *IEEE Trans. on Automatic Control*, 43(4):457–460. Guest Editorial.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE J. of Robotics and Automation*, 2(1):14–23.

Davidsson, P. and Boman, M. (2000). A multi-agent system for controlling intelligent buildings. Proc. *4th Int. Conf. on MultiAgent Systems*, pp. 377–378.

Davoren, J. M. and Nerode, A. (2000). Logic for hybrid systems. *Proc. of IEEE Special Issue on Hybrid Systems*, 88(7):985–1010.

Franklin, G. F., Powell, J. D., and Emami-Naeini, A. (2002). *Feedback Control of Dynamic Systems*. Prentice hall.

Franklin, S. (1995). *Artificial Minds*. MIT Press.

Hagras, H., Callaghan, V., Colley, M., Clarke, G., Pounds-Cornish, A., and Duman, H. (2004). Creating an ambient-intelligence environment using embedded agents. *IEEE Intelligent Systems and Their Applications*, 19(6):12–20.

InterProlog (2004). Declarativa. Available at `www.declarativa.com/InterProlog/default.htm`.

Java Technology (2004). Sun microsystems. Available at http://java.sun.com.

Koutsoukos, X. D., Antsaklis, P. J., Stiver, J. A., and Lemmon, M. D. (2000). Supervisory control of hybrid systems. *Proc. of IEEE, Special Issue on Hybrid Systems*, 88(7):1026–1049.

Maes, P. (1989). How to do the right thing. *Connection Science Journal, Special Issue on Hybrid Systems*, 1(3):291–323.

Maes, P. (1991). A bottom-up mechanism for behavior selection in an artificial creature. In Meyer, J. A. and Wilson, S. (eds.), *Proc. of the first Int. Conf. on Simulation of Adaptive Behavior*. MIT Press.

Minsky, M. (1986). *The Society of Mind*. Simon and Schuster, New York.

Mozer, M. M. (1998). The neural network house. an environment that adapts to its inhabitants. In Coen, M. (ed.), *Proc. of the Ameriacan Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pp. 110–114.

Rutishauser, U., Joller, J., and Douglas, R. (2005). Control and learning of ambience by an intelligent building. *IEEE Trans. on Systems, Man and Cybernetics, Part A*, 35(1):121–132. Special Issue on Ambient Intelligence.

Singleton, D. (2002). *An Evolvable Approach to the Maes Action Selection Mechanism*. Master Thesis, University of Sussex. Available at `http://www.informatics.susx.ac.uk/easy/Publications`.

Tu, X. (1999). *Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior*. PhD thesis, ACM Distinguished Ph.D Dissertation Series, LNCS 1635.

van Beek, B., Jansen, N. G., Schiffelers, R. R. H., Man, K. L., and Reniers, M. A. (2003). Relating chi to hybrid automata. In S. Chick, P.J. Sánchez, D. F. and Morrice, D. (eds.), *Proc. of the 2003 Winter Simulation Conference*, pp. 632–640.

XSB-Prolog (2004). XSB Inc. Available at `xsb.sourceforge.net`.