Automatic Orchestration of Behaviours through the management of Resources and Priority Levels

Fabrice Lamarche
IRISA / University of Rennes I
Campus de Beaulieu
F-35042 Rennes, FRANCE
flamarch@irisa.fr

Stéphane Donikian
IRISA / CNRS
Campus de Beaulieu
F-35042 Rennes, FRANCE
donikian@irisa.fr

ABSTRACT

Reproducing daily behaviours requires to be able to schedule behaviours depending on resources and priority constraints. A simple way is to say that behaviours which are using the same resources are mutually exclusive. This approach is not sufficient to obtain realism, as in the real life, humans are able to combine them in a more microscopic way. One way consists in a global specificication of all behaviours in one model, integrating all possible combinations. This solution becomes rapidly too complex and has motivated the work presented in this paper. It consists in an extension of HPTS, our behavioural model which integrates several psychological requirements, by the introduction of resources and priority levels. In the contrary of some previous approaches, it is not necessary to specify exhaustively all behaviours that are mutually exclusive; this is done implicitly by attaching resources to nodes, preference values to transitions, and a priority function to each behaviour, and by using a scheduling algorithm at run-time.

Keywords

Believability, Synthetic Agents, Agent Architectures

1. INTRODUCTION

The goal of behavioural models is to simulate autonomous entities like organisms and living beings. The issue adressed in our work concerns the specification of a general formalism for behaviour modelling based on psychological studies and compatible with real-time constraints. Information needed to describe the behaviour of an entity, depends on its nature. No theory exists for determining either the necessary or sufficient structures needed to support particular capabilities and certainly not to support general intelligence. As direction and inspiration towards the development of such a theory, Newell[11] posits that one way to approach sufficiency is by modelling human cognition in computational layers or bands. Reproducing daily behaviours requires to be able to schedule behaviours depending on resources (body parts) and priorities (intentions or physiological parameters). A simple way is to say that behaviours which are using the same resources are mutually exclusive. This approach is not sufficient to obtain realism, as in the real life, humans are able to combine them in a much microscopic way. All day long, human being combines different behaviours, for instance reading a newspaper while drinking a coffee and smoking a cigarette. If all behaviours using common resources were mutually exclusive, an agent could not reproduce this example, except if a specific behaviour, integrating all possible combinations, is created for this purpose. This solution becomes rapidly too complex.

We have proposed in the past the HPTS model which integrates several psychological requirements. In this paper, we propose to extend this model to be able to manage, in a generic way, resources, adaptation and priority levels. It allows to describe behaviours independently one from each other and to adapt automatically their execution when they are running in parallel, in respect with their priorities. In the next section, related works are presented, while section three focuses on the HPTS model. The integration of resources in the model is presented in section four, while section five is devoted to the presentation of the scheduling algorithm. To conclude, section six illustrates the approach on an example.

2. RELATED WORKS

Behavioural models have been developed to describe human behaviour in specific tasks. Common characteristics of these models are reactivity, parallelism and different abstract levels of behaviours. As humans are deliberative agents, purely reactive systems are not sufficient to describe their behaviour. It is necessary to integrate both cognitive and reactive aspects of behaviour. Cognitive models are rather motivated by the representation of the agent's knowledge (beliefs and intentions). Intentions enable an agent to reason about its and others internal state. The centre of such a deliberative agent is its own representation of the world which includes a representation of the mental state of itself and of other agents with whom he is currently interacting[7]. In order to do it, Badler et al.[2] propose to combine Sense-Control-Action (SCA) loops with planners and PaT-Nets. SCA loops define the reflexive behaviour and are continuous systems which interconnect sensors and effectors through a network of nodes. PaT-Nets are essentially finite state automata wich can be runned in parallel.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AAMAS'02, July 15-19, 2002, Bologna, Italy.
Copyright 2002 ACM 1-58113-480-0/02/0007 ...$5.00.

The planner queries the state of the database through a filtered perception to decide how to elaborate the plan and to select an action. In this system, like in others[12], the action is directly associated with each node, which doesn't allow the management of concurrency.

Other planning systems elaborate plans by reasoning on actions [6] and their associated resources [10]. Concurrency on resources is handled through constraints and resources are allocated by the planner while conceiving the plan. The introduction of reactivity in such systems implies to re-plan continuously which is not manageable in real time applications. V. Decugis and J. Ferber[3] address an interesting problem: how to combine reactivity and planning capabilities in a real-time application. They propose to extend the ASM (Action Selection Mechanism) proposed by Maes[8] into hierarchical ASMs. At the hierarchy's bottom, basic reflexes are found, such as reflex movements orientation and basic perceptive mechanisms, while higher levels integrate more complex behaviours. At each level an arbitration mechanism should be used to choose between parallel ASMs. At each time, there is one behaviour selected at each level of the hierarchy. A lot of models have been also proposed for human like minds in the agent community[9]. They are all based on the perception/treatment/action loop, but they mainly differ in the way the treatment unit is built. As in the Newell theory, A. Sloman[13] proposed an architecture of an intelligent agent in different layers, involving different routes through the system from perception to action. In his theory, automatic processes have dedicated portions of the brain and can operate in parallel whenever they need, while different management processes have to share a common working memory, and their parallelism is then restricted.

According to Newell, our goal is to build a model which will allow some adjustable and flexible behaviour to any entity evolving in a complex environment and interacting with other entities. Real-time execution is also fundamental to allow interactivity. This has lead us to state that paradigms required for programming a *realistic* behavioural model are: reactivity (which encompasses sporadic or asynchronous events and exceptions), modularity in the behaviour description (which allows parallelism and concurrency of sub-behaviours), data-flow (for the specification of the communication between different modules), hierarchical structuring of the behaviour (which means the possibility of preempting sub-behaviours). HPTS[5], like HCSM[1], is a model based on hierarchical concurrent state machines. It offers a set of programming paradigms which permit to address hierarchical concurrent behaviours. HPTS offers also the ability to manage temporal informations and undeterministic choice[4]. Resource and priority management has been integrated recently, and it will be presented in detail in the core of this paper.

3. HPTS

HPTS[5] which stands for Hierarchical Parallel Transition Systems, consists of a reactive system, which can be viewed as a multi-agent system in which agents are organized as a hierarchy of state machines. Each agent of the system can be viewed as a black-box with an In/Out data-flow and a set of control parameters. The synchronization of the agent execution is operated using state machines. To allow an agent to manage concurrent behaviours, sub-agents are organized in sub-state machines. In the following, agents will be assimilated to state machines. Each state machine of the system is either an atomic state machine, or a composite state machine. Though the model may be coded directly with an imperative programming language like C++, we decided to build a language for the behaviour description. Figure 1 presents the syntax of the behaviour programming language which fully implements the HPTS formalism. As this paper focuses on integration and management of resources and priority levels, the behaviour description language is not described in details. For a complete description of the model (except for resources and priorities) refers to [4]. Keywords are written in bold, whereas italic typeface represents a non-terminal rule. A* stands for a $0..n$ repetition while a$^+$ stands for a $1..n$ repetition and a statement enclosed in { } is optional. The description of a state machine is done

SMACHINE *Id* ;
{
  PARAMS *type Id* {, *type Id*}* ; // Parameters
  VARIABLES { {*type Id* ;}* **}** // Variables
  OUT *Id* {, *Id*}* ; // Outputs
  PRIORITY = *numeric expression* ;
  INITIAL *Id* ; **FINAL** *Id* ;
  STATES // States Declaration
  {{
    Id {[*Id* {, *Id*}]} {**RANDOM**} {**USE** *resource list*}; {**{** /* state body */ **}**}
  }$^+$**}**
  {**TRANSITION** *Id* {**PREFERENCE** *Value*};
  {
    ORIGIN *Id* ; **EXTREMITY** *Id* ; {**DELAY** *float* ;} {**WEIGHT** *float* ;}
    read-expr / *write-expr* {**TIMEGATE**} ; {**{** /* transition body */ **}**}
  }}*
}

Figure 1: Syntax of the language.

in the following way: the body of the declaration contains a list of states and a list of transitions between these states. A state is defined by its name and its activity with regard to data-flows. It accepts an optional duration parameter which stands for the minimum and maximum amount of time spent in the state. Resources used by a state are defined by using the instruction *USE resource list*. The management of resources is explained in the next section. A state machine can be parameterized; the set of parameters will be used to characterize a state machine at its creation. Variables are local to a state machine. Only variables that has been declared as outputs can be viewed by the meta state machine. A priority is attached to each state-machine and consists in a numeric expression which allow the priority to evolve during the simulation. The meaning of this priority function will be explained in section 5. A transition is defined by an origin, an extremity, a transition expression, two optional parameters and a transition body. The transition expression consists of two parts: a *read-expr* which includes the conditions to be fulfilled in order to fire the transition, and a *write-expr* which is a list of the generated events and basic activity primitives on the state machine. A preference value is defined for each transition, and its meaning will be explained in section 5.

4. RESOURCES

In order to combine different behaviours, the notions of resources and mutual exclusion have been included in the model. This section focuses on the formalism used to describe resource allocations and the different constraints created to allow safe concurrency on resources among all parallel state machines. Each state of a state machine can use a set of resources noted R and a state machine is defined by the following tuple $A = (E_A, T_A, P_A, i_A, f_A, prio_A)$ in which:

- E_A is a set of nodes.
- $T_A \subset E_A \times E_A \times (() \rightarrow \mathit{bool}) \times [-1.0; 1.0]$ is a set of transitions. Each of them has an associated condition and a degree of preference taking its value in interval $[-1.0; 1.0]$. Notation $() \rightarrow \mathit{bool}$ refers to a function without parameter that returns a boolean value.
- $P_A : E_A \rightarrow 2^R$ is a function returning the set of resources used by each state.
- $i_A \in E_A$, with $P_A(i) = \emptyset$, is the initial state,
- $f_A \in E_A$ is the final state.
- $prio_A : () \rightarrow \mathbb{R}$ is a function without parameter that returns a real value corresponding to the priority of the state machine.

Note that all elements defining a state machine are indexed by the state machine they belong to. Those notations will be used for all equations given in this article.

4.1 Resources handling

For all state machines, a set of resources R is defined by the user. Those resources can be considered as semaphores, thus they are used for mutual exclusion. Function $P_A : E_A \rightarrow 2^R$ associates a set of taken resources to each node of a state machine A. Entering a node implies that resources it uses are marked as taken and exiting it implies that those resources are released. A resource is kept when it is used by two nodes which are connected by a transition. Thus the behaviour description is as precise as simple. As resources are semaphores, it is possible to explain the first constraint that all nodes executed in parallel must respect. Let $e_1, ..., e_n$ be n nodes respectively belonging to n different state machines $A_1, ..., A_n$. States $e_1, ..., e_n$ can be executed in parallel only if the constraint explained below is true:

$$\forall (i,j) \in \{1..n\}^2, i \neq j, P_{A_i}(e_i) \cap P_{A_j}(e_j) = \emptyset \qquad (1)$$

Let note $P_k = \bigcup_{1 \leq i < k} P_{A_i}(e_i)$ the set containing all resources used by nodes e_1 to e_{k-1}. This constraint is verified when the following formula is set to true:

$$\begin{cases} c1_n(e_1, ..., e_n) & = & (P_{A_n}(e_n) \cap P_n = \emptyset) \wedge \\ & & c1_{n-1}(e_1, ..., e_{n-1}) \\ c1_1(e_1) & = & \mathit{true} \end{cases} \qquad (2)$$

Using this constraint, it becomes possible to synchronize behaviours according to resources needed by them. As several state machines may use the same resources, the problem of dead locks has to be studied.

4.2 Dead locks

Dead locks occur when the dependency graph of the resources is cyclic. In that case, resources taking part in dead lock cannot be released, except if a controller empirically decides to release some resources by killing a state machine. However, this system is not sufficient to ensure consistency and continuity in behaviours. Thus, to offer a maximal level of security and make synchronization easier to handle, it is necessary provide a mechanism to ensure that no dead lock can arise. A transition between two nodes using resources creates a dependency between resources associated to those nodes: resources of the first node are released if resources of the second one can be taken. In order to get information about dependencies, the notion of inherited resources associated to a node have been defined. Those inherited resources are computed using a graph representing the structure of each state machine reduced to dependencies between resource allocations. Let $G_A = (E_{G_A}, T_{G_A})$ be the graph associated to the finite state machine A, where

- $E_{G_A} = E_A$ is the set of states,
- $T_{G_A} = \{(e_1, e_2) \mid \exists c \in () \rightarrow \mathit{bool}, \exists p \in \mathbb{R}, (e_1, e_2, c, p) \in T_A \wedge P_A(e_1) \neq \emptyset \wedge P_A(e_2) \neq \emptyset \}$ is the set of state transitions.

As vertices of this graph only connect nodes using resources, it represents all dependencies between resource allocation deduced from the state machine. Let define $H_A \in E_A \rightarrow 2^R$, an application associating to each node its set of inherited resources, by:

$$\forall e \in E_A, H_A(e) = \bigcup_{(s \in E_{G_A}) \wedge (e \xrightarrow[G_A]{+} s)} P_A(s)$$

Note that inherited resources can be precomputed, assuming that the structure of the state machine is known before runtime. By using this information, it is possible to create a constraint which have to be respected to ensure that no dead lock can occur while being at the same timestep in two nodes of parallel state machines. Let A_1 and A_2 be two finite state machines. Let $e_1 \in E_{A_1}$ and $e_2 \in E_{A_2}$ be two states belonging respectively to A_1 and A_2; e_1 and e_2 can be executed in parallel without dead lock if:

$$(P_{A_1}(e_1) \cap H_{A_2}(e_2) = \emptyset) \vee (H_{A_1}(e_1) \cap P_{A_2}(e_2) = \emptyset) \qquad (3)$$

This constraint ensures that at least, one state machine can pass its transitions without conflict on resources. It ensures that no cyclic dependency can occur in resource allocation, and then no dead lock. But the set of inherited resources does not describe precisely resource dependencies. Thus, this constraint detects a superset of dead locks; some nodes which could run in parallel without dead lock, will not. Constraint expressed in formula (3) is extensible to n nodes. Let $e_1, ..., e_n$ be n nodes respectively belonging to n different state machines $A_1, ..., A_n$. Let define $H_i = \cup_{1 \leq k < i} H_{A_k}(e_k)$ and $P_i = \cup_{1 \leq k < i} P_{A_k}(e_k)$. Nodes $e_1, ..., e_n$ can be run in parallel without dead locks if the constraint explained below is true:

$$\begin{cases} c2_n(e_1, ..., e_n) & = & (\ (P_{A_n}(e_n) \cap H_n = \emptyset) \vee \\ & & (H_{A_n}(e_n) \cap P_n = \emptyset)\ ) \\ & & \wedge\ c2_{n-1}(e_1, ..., e_{n-1}) \\ c2_1(e_1) & = & \mathit{true} \end{cases} \qquad (4)$$

As an extension of constraint expressed in formula (3), it also detects a superset of dead locks. Furthermore, it is dependent on the order of presentation of nodes. Nevertheless, the verification complexity is $O(n)$ in term of set operations, where n represents the number of nodes. This make possible to handle intensive computation of different nodes compatibility.

As HPTS is a hierarchical model, each state machine can create sons and wait for their ending; this synchronization creates dependencies between state machines. Thus, there are possibilities of dead locks if a state machine uses common resources with its sons while waiting for their ending. Thus, another constraint has been added: resources used by a state machine have to be different than resources used by its descendants. The respect of this constraint implies that all descendants can be executed and terminated before the ending of its ascendants without dead lock nor conflicts on resources.

4.3 Valid state combination selection

All nodes ending a transition having a valid condition and starting from current node are accessible. Current node plus all accessible nodes are considered as valid for the current time-step. Thus, any of those nodes can become the new active node of its associated state machine, depending on resource availability. Let $\{A_1, ..., A_n\}$ be n state machines running in parallel. Let $\forall k \in [1..n], c_{A_k}$ be the current node of the finite state machine A_k. Let $\mathcal{E}_{A_k}$ be the set of valid nodes associated to the finite state machine A_k at current time-step.

$$\forall k \in [1..n], \mathcal{E}_{A_k} = \{e \mid (c_{A_k}, e, c, p) \in T_{A_k} \wedge c()\} \cup \ \{c_{A_k}\} \tag{5}$$

Let note $\mathcal{P} = \prod_{k=1}^{n} \mathcal{E}_{A_k}$ the set of all possible combinations of valid nodes between all running state machines. Due to the concurrency of state machines on resources, all combinations are not valid. In fact, some combinations can contain nodes using common resources (Cf. 4.1) or can produce a dead lock (Cf. 4.2). Thus, using constraints defined in formula (2) and (4), it is possible to create a subset ($\mathcal{P}_{comp}$) of $\mathcal{P}$ containing all the valid combinations of states:

$$\begin{aligned}\mathcal{P}_{comp} = \{(e_1, ..., e_n) \quad & \mid c2_n(e_1, ..., e_n) \ \wedge c1_n(e_1, ..., e_n) \\ & \wedge \ (e_1, ..., e_n) \in \mathcal{P} \ \}\end{aligned} \tag{6}$$

While state machines are running, this set contains at each time-step all valid combinations available, but the best combination have to be chosen among the others. In order to define this notion of best combination, priorities and degrees of preference are introduced.

5. SCHEDULING WITH PREFERENCES AND PRIORITIES

In order to control the execution of parallel state machines and to offer an automatic adaptation between different behaviours, it is necessary to add notions of priorities and preferences. By combining those two notions, it is possible to create a scheduling method which globally favours the realization of most important behaviours while automatically adapting the execution of running ones.

5.1 Degrees of preference

The degree of preference (p) is a coefficient which takes its value in interval $[-1.0; 1.0]$. This coefficient, is associated to a transition, and corresponds to the state machine proclivity to use this transition when the associated condition is true. Thus, depending on its value, this coefficient has different meanings:

- $p > 0$: the transition favours the realization of this behaviour. By default, the transition having the greatest degree of preference with a condition set to true should be chosen.
- $p < 0$: the transition does not favour the realization of this behaviour. Those transitions are used to describe a coherent way of stopping or adapting behaviour while releasing some resources.
- $p = 0$: the behaviour is quite indifferent to this transition.

This coefficient allows to describe a behaviour with different ways of realization; possible adaptation depending on resources availability or need can be described.

5.2 Priorities

A priority function ($prio_A$) is associated to each state machine. This function returns a real value representing the importance of a behaviour in a given context. Depending on its sign, this function has different meanings:

- $prio_A() > 0$: the behaviour has to be executed, and is adapted to the current context. This value can be interpreted as a coefficient of adequacy between context and behaviour.
- $prio_A() < 0$: the behaviour is inhibited, the value can be interpreted as a coefficient of inadequacy between context and behaviour.

This function can be used to control the behaviour during the running phase. As it is user defined, it can be correlated with the internal state of the character (psychological parameters, intentions) or with external stimuli. It provides an easy way of control on the behaviour realization.

5.3 Scheduling method

Using preferences and priorities, it is possible to create a weight function. This function associates a rate to each valid state combination in $\mathcal{P}_{comp}$ (Cf. 4.3), allowing to choose the best one in respect of degrees of preference and priorities. Let reformulate equation (5) in order to associate to each node the degree of preference associated to its valid transition(s):

$$\begin{aligned}&\forall k \in [1..n], \\ &\qquad \mathcal{E}'_{A_k} = \quad \{(e, p) \mid (c_{A_k}, e, c, p) \in T_{A_k} \wedge c()\} \\ &\qquad\qquad\quad \cup \{(c_{A_k}, 0)\}\end{aligned} \tag{7}$$

By default, a degree of preference equal to 0 is associated to current state machine node. Thus, staying in current state does not favour nor penalize the realization of this behaviour. Then it is possible to create a function $W_{A_k} : \mathcal{E}_{A_k} \rightarrow \mathbb{R}$ associating to each valid node its weight:

$$\forall e \in \mathcal{E}_{A_k}, W_{A_k}(e) = Max_{(e,p) \in \mathcal{E}'_{A_k}} p \times prio_{A_k}() \tag{8}$$

Let consider a weight $W_{A_k}(e) = prio_{A_k}() \times p$ associated to a valid node e of a state machine A_k. This weight has different meanings:

- $W_{A_k}(e) > 0$, automaton is prone to transit in state e:
    - $prio_{A_k}() > 0 \wedge p > 0$: transiting to this node favours the realization of this behaviour.
    - $prio_{A_k}() < 0 \wedge p < 0$: this behaviour is inhibited, transitions which lead to a consistent ending of this behaviour have to be favoured.
- $W_{A_k}(e) < 0$, automaton is not prone to transit to state e:
    - $prio_{A_k}() > 0 \wedge p < 0$: transiting in this state does not favor the realization of the behaviour. This case can be used in order to propose a possibility of adaptation of the behaviour releasing some resources that are not necessary. It can also be used to stop behaviour execution because a behaviour having a greater priority needs resources used by the considered behaviour.
    - $prio_{A_k}() < 0 \wedge p > 0$: the behaviour is inhibited, this case can be used to propose another possibility of stopping behaviour by using less resources.
- $W_{A_k}(e) = 0$, automaton is indifferent to transit to node e.

As explained in section 4.3 we need to select the best combination in the set $\mathcal{P}_{comp}$ containing all valid combinations. Equation (8) associates a weight to each proposed node with respect to degrees of preference and priorities. It is possible to create another weighting function $\mathcal{W} : \mathcal{P}_{comp} \rightarrow \mathbb{R}$ associating a weight to each combination of nodes:

$$\forall (e_1, ..., e_n) \in \mathcal{P}_{comp}, \mathcal{W}(e_1, ..., e_n) = \sum_{i=1}^{n} W_{A_i}(e_i) \qquad (9)$$

The result of this function can be interpreted as a global degree of satisfaction of states machines if the corresponding combination is chosen. Thus, combinations maximizing this weighting function are those which globally maximize the degree of satisfaction of many state machines. Behaviours having the greatest priorities will be favoured in their realization while the inhibited ones or those wich have less priority will release their resources if possible and in a consistent way. Adaptation between different concurrent behaviours becomes automatic. Let $\mathcal{M} = Max_{(e_1,...,e_n) \in \mathcal{P}_{comp}} \mathcal{W}(e_1, ..., e_n)$ be the best degree of satisfaction found in $\mathcal{P}_{comp}$. The set of best combinations ($\mathcal{P}_{best}$) is defined by:

$$\mathcal{P}_{best} = \{(e_1, ..., e_n) \mid (e_1, ..., e_n) \in \mathcal{P}_{comp} \wedge \mathcal{W}(e_1, ..., e_n) = \mathcal{M}\} \qquad (10)$$

After the creation of the set $\mathcal{P}_{best}$, the scheduler can choose one of its combination as the new overall state machines state. This scheduling method ensures consistency in behaviours while trying to automatically adapt their execution to other executing behaviours in accordance with priorities and degrees of preference.

5.4 Algorithm complexity

Assuming that scheduling is computed at each time-step, it is usefull to limit its complexity. Using the method described in sections 4.3 and 5.3, the algorithm complexity could appear to be $\prod_{k=1}^{n} card(\mathcal{E}_{A_k})$ in term of computed combinations and constraints verification. But constraints expressed in (2) and (4) only use taken and inherited resources associated to a node. Futhermore, while computing weights associated to nodes (Cf. 5.3), only the best weight obtained by combining preferences and priorities is important. Exploiting those properties allows to reduce the complexity of the algorithm. First of all, it is possible to compute the set $\mathcal{P}_{comp}$ incrementally while verifying constraints expressed in (2) and (4):

$$\begin{cases} \mathcal{P}_{comp}^{(n)} = \{(e_1, ..., e_n) \mid & e_n \in \mathcal{E}_{A_n} \wedge \\ & (e_1, ..., e_{n-1}) \in \mathcal{P}_{comp}^{(n-1)} \wedge \\ & ((P_{A_n}(e_n) \cap H_n = \emptyset) \vee \\ & (P_n \cap H_{A_n}(e_n) = \emptyset)) \wedge \\ & (P_{A_n}(e_n) \cap P_n = \emptyset)\} \\ \mathcal{P}_{comp}^{(1)} = \mathcal{E}_{A_1} & \end{cases} \qquad (11)$$

Incremental computation of the set $\mathcal{P}_{comp}$ allows, at each level of computation, to suppress incompatible combinations as soon as they are found. Moreover, the constraints verification, at level $k > 1$, only use:

- taken and inherited resources associated to a node belonging to $\mathcal{E}_{A_k}$,
- the union of taken resources and the union of inherited resources associated to nodes constituting a combination of $\mathcal{P}_{comp}^{(k-1)}$.

Let note $\mathcal{R}^{(k)}$ a set of couple which are composed of the union of taken resources and the union of inherited resources associated to each combination of $\mathcal{P}_{comp}^{(k)}$. Let consider a couple (P,H) belonging to $\mathcal{R}^{(k)}$. This couple (P,H) is associated to a set of combinations of $\mathcal{P}_{comp}^{(k)}$, let note $\mathcal{S}_{P,H}^{(k)}$ this set of combinations. All combinations in $\mathcal{S}_{P,H}^{(k)}$ are equivalent for constraints verification. The difference between such combinations is their weight, only those having the greatest weight can contribute to the calculus of combinations of the set $\mathcal{P}_{best}$. It is possible to create a filtering function $\mathcal{F}^{(k)}$ in order to filter combinations of the set $\mathcal{P}_{comp}^{(k)}$. This function creates a subset of $\mathcal{P}_{comp}^{(k)}$, keeping for each couple (P,H) of $\mathcal{R}^{(k)}$ one of the combinations of $\mathcal{S}_{P,H}^{(k)}$ wich obtains the best weight. Thus, the following property is deduced: $card(\mathcal{F}^{(k)}(\mathcal{P}_{comp}^{(k)})) = card(\mathcal{R}^{(k)})$. Applying this function on $\mathcal{P}_{comp}^{(k)}$ does not modify the scheduling result. Actually, choosing one of the combinations obtaining the best weight is done during computation instead of at the end of computation. Moreover, $(P, H) \in 2^R \times 2^R$, thus the following property is deduced:

$$card(\mathcal{F}^{(k)}(\mathcal{P}_{comp}^{(k)})) \leq min(\textstyle\prod_{i=1}^{k} \mathcal{E}_{A_i}, 2^{2 \times card(R)}) \qquad (12)$$

This function allows to limit the number of computed and stored combinations, and then limits the algorithm complexity in term of stored combinations and computation. Moreover, it is possible to apply this function on each set $\mathcal{E}_{A_k}$. Finally, the set $\mathcal{P}_{comp}$ is computed as follow:

$$\begin{cases} \mathcal{P}_{comp}^{(n)} = \{(e_1, ..., e_n) \mid & e_n \in \mathcal{F}^{(1)}(\mathcal{E}_{A_n}) \wedge \\ & (e_1, ..., e_{n-1}) \in \mathcal{F}^{(n-1)}(\mathcal{P}_{comp}^{(n-1)}) \wedge \\ & ((P_{A_n}(e_n) \cap H_n = \emptyset) \vee \\ & (P_n \cap H_{A_k}(e_n) = \emptyset)) \wedge \\ & (P_{A_n}(e_n) \cap P_n = \emptyset)\} \\ \mathcal{P}_{comp}^{(1)} = \mathcal{F}^{(1)}(\mathcal{E}_{A_1}) & \end{cases} \qquad (13)$$

After computing the set $\mathcal{P}_{comp}$, the method described in section 5.3 is applied in order to extract the set $\mathcal{P}_{best}$ and to choose a combination. This algorithm allows to schedule a large number of behaviours with about 1 to 9-10 resources without problems of memory and in real time. Note that maximum number of stored combinations has a very small probability to arise, due to the verification of constraints expressed in (2) and (4). Moreover, it is possible to handle more resources by partitionning the set of running state machines as regard to the resources used by each of them and to compute a scheduling for each partition.

5.5 Illustration on a case study

Let study the scheduling algorithm on the three state machines (a1,a2,a3) detailed in Fig. 2. They are running in parallel with concurrency on the set of resources $R = \{E, H_r, H_l\}$ (Eyes, Right Hand, Left Hand). Current nodes and priorities of the three state machines are respectively: (s1, 11), (s5, 4) and (s7, 6). Knowing the current node and priority of each state machine, and assuming that all conditions associated to transitions are set to true, the set of valid states with their weights can be computed.

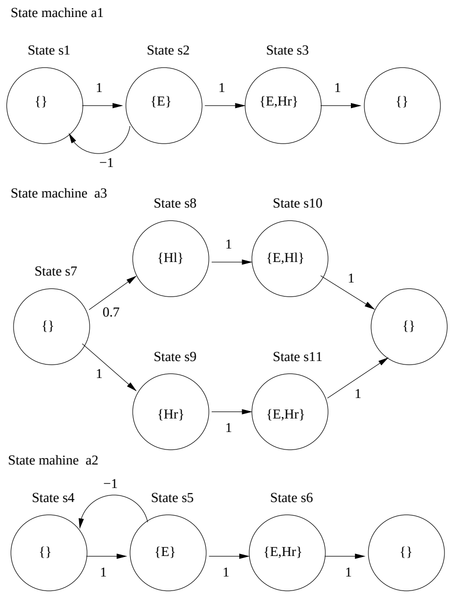

Figure 2: Three state machines running in parallel.

	Valid State	Taken Resources	Inherited Resources	Weight
a1	s1	$\emptyset$	$\emptyset$	0
	s2	$\{E\}$	$\{E, H_r\}$	$1 \times 11 = 11$
a2	s4	$\emptyset$	$\emptyset$	$-1 \times 4 = -4$
	s5	$\{E\}$	$\{E, H_r\}$	0
	s6	$\{E, H_r\}$	$\emptyset$	$1 \times 4 = 4$
a3	s7	$\emptyset$	$\emptyset$	0
	s8	$\{H_l\}$	$\{E, H_l\}$	$0.7 \times 6 = 4.2$
	s9	$\{H_r\}$	$\{E, H_r\}$	$1 \times 6 = 6$

Figure 3: State attributes.

In Fig. 3 each valid node for each state machine is enumerated with its taken and inherited resources and its weight.

First step of the algorithm: This iteration consists in initializing the set $\mathcal{F}^{(1)}(\mathcal{P}^{(1)}_{comp})$ with valid states of state machine a1. In Fig. 4, all nodes contained by $\mathcal{F}^{(1)}(\mathcal{P}^{(1)}_{comp})$ and their associated information are described.

Taken Resources	Inherited Resources	Node Combination	Weight
$\emptyset$	$\emptyset$	$(s1)$	0
$\{E\}$	$\{E, H_r\}$	$(s2)$	11

Figure 4: $\mathcal{F}^{(1)}(\mathcal{P}^{(1)}_{comp})$.

Second step of the algorithm: During this iteration, the compatibility between states of $\mathcal{F}^{(1)}(\mathcal{P}^{(1)}_{comp})$ and valid states of a2 is computed and a weight is associated to each valid combinations. The set $\mathcal{F}^{(2)}(\mathcal{P}^{(2)}_{comp})$ is constructed while filtering valid combinations. Combinations (s5,s1) and (s2,s4) have the same couple of taken and inherited resources. So, the filtering function only keeps combination (s2,s4) because it obtains the best weight. The set $\mathcal{F}^{(2)}(\mathcal{P}^{(2)}_{comp})$ and information associated to its combinations are described in Fig. 5.

Taken resources	inherited resources	node combination	weight
$\{E\}$	$\{E, H_r\}$	(s2,s4)	7
$\{E, H_r\}$	$\emptyset$	(s1,s6)	4
$\emptyset$	$\emptyset$	(s1,s4)	-4

Figure 5: $\mathcal{F}^{(2)}(\mathcal{P}^{(2)}_{comp})$.

Third step of the algorithm: The compatibility between combinations of $\mathcal{F}^{(2)}(\mathcal{P}^{(2)}_{comp})$ and valid nodes of state machine a3 is computed and a weight is associated to each valid combinations. Finally, the set $\mathcal{F}^{(3)}(\mathcal{P}^{(3)}_{comp})$ is computed while filtering valid combinations (cf figure 6).

Taken Resources	Inherited Resources	Node Combination	Weight
$\{E, H_l\}$	$\{E, H_l, H_r\}$	(s2,s4,s8)	11.2
$\{E, H_l, H_r\}$	$\{E, H_l\}$	(s1,s6,s8)	8.2
$\{E\}$	$\{E, H_r\}$	(s2,s4,s7)	7
$\{E, H_r\}$	$\emptyset$	(s1,s6,s7)	4
$\{H_l\}$	$\{E, H_l\}$	(s1,s4,s8)	0.2
$\emptyset$	$\emptyset$	(s1,s4,s7)	-4

Figure 6: $\mathcal{F}^{(3)}(\mathcal{P}^{(3)}_{comp})$.

Choosing best combination: The combination belonging to $\mathcal{F}^{(3)}(\mathcal{P}^{(3)}_{comp})$ obtaining the best weight is chosen as the new overall state machines state. In this example, combination (s2,s4,s8) will be chosen. State machine a1 returns in node s2 in order to enable transition of state machine a2. State machine a3 adapts its execution by transiting to node s8 whereas it would rather have transited to node s9. This example demonstrates the influence of priorities in the scheduling algorithm and the automatic adaptation of state machines transitions in order to respect resource constraints.

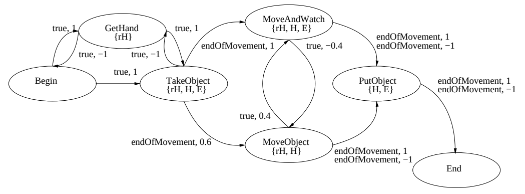

Figure 7: Moving object behaviour.

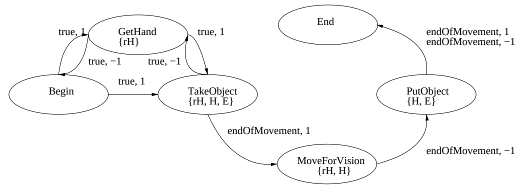

Figure 10: Behaviour handling hands while reading.

6. EXAMPLE

In this section, we will study the example given in introduction (drink a coffee and smoke a cigarette while reading a newspaper) and show how the scheduler is able to reproduce this behaviour just by describing independently the three sub-behaviours and their associated priority functions. All state machines use the following set of resources: Hl (left hand), Hr (right hand), rHl (reserve left hand), rHr (reserve right hand), M (mouth) and E (eyes). Resources rHl/rHr are used to handle releasing of resources Hl/Hr. The scheduler can only act on the next transition of a state machine. Hands are resources that often need more than one timestep to be freed, for instance, putting down an object to free the hand resource. Then a state which only use resource Hr/Hl corresponds to a behaviour of freeing a hand resource. All state machines used to solve this problem are presented in figures 8, 9 and 10. In these state machines, resource H stands for Hr or Hl as it exists the same behaviour for each hand. Note that descriptions of state machines are totally independent one from each other.

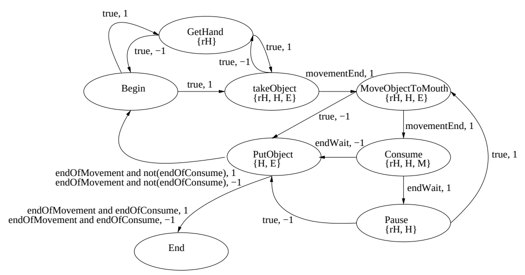

Figure 8: Common behaviour for drinking and smoking.

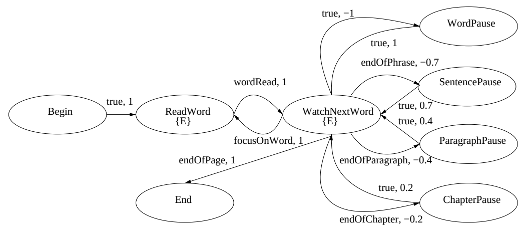

Figure 9: Reading behaviour.

Drinking and smoking: Those two behaviours are described through the same state machine consisting in grasping the object of interest, moving it to the mouth and keeping the object into the hand. The object is put on the table if another behaviour needs hand resource or if the current behaviour becomes inhibited.

Reading the newspaper: This behaviour consists in two parallel sub-behaviours. One consists in reading the newspaper with different possibilities of pause depending on the text structure. Those different levels of interruption are described through degrees of preference. The second consists in manipulating the newspaper taking it into the hand and moving it near the eyes.

Note that thanks to resources, each behaviour is described independently from the others but propose different possibilities of adaptation. Each possibility of adaptation is described through degrees of preference which allow to specify the cost of such adaptation. Moreover, as a mechanism ensures that no deadlock can arise, conceiving a behaviour does not need to know other described behaviours.

Defining priorities: the importance of behaviours described below depends on different parameters. The difficulty arises with the fact of unifying priority functions that depend on different parameters.

Drinking and smoking: those two behaviours have variable priorities evolving with the time. Those priorities are directly correlated to the thirst/need of nicotine. Let note $p(t)$ the priority function where t represents the time. It has the following definition:

$$\begin{cases} p(t) = p(t-dt) + dp_1 \times dt \ if \ not(consuming) \\ p(t) = p(t-dt) - dp_2 \times dt \ if \ consuming \end{cases} \tag{14}$$

where dp_1 (respectively dp_2) is the increase rate (respectively the decrease rate) of the thirst or the need of nicotine. Consuming stands for drinking or smoking depending on the behaviour.

Reading: priority of reading behaviour is correlated to the interest of the reader for the text. In this example, the function is defined as a constant. The behaviour having the greatest priority is the reading one while behaviour allowing to manipulate the sheet has a lower priority.

In order to handle the reading behaviour, another behaviour has been added which consists in moving a sheet of paper with the right hand in front of the agent in order to read, and when the sheet is read, it is moved and the next sheet is taken. Moving the sheet of paper is handled by the behaviour described in figure 7.

Note that once behaviours are described through state machines, they are controlled through their priority. This property allow to handle every type of executive behaviour without need of information about their internal structure in term of resources or possible adaptations. Figure 11 shows the activity of each running behaviour. Activity is defined by states using resources Hr/Hl/E/M, which can be assimilated to active resources (i.e. resources enabling manipula-

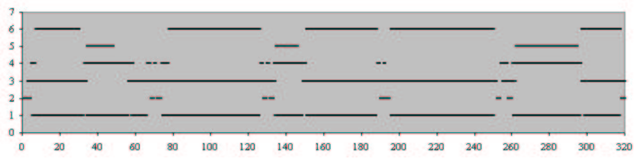

Figure 11: Activity of behaviours during simulation.

tion of body parts). In this figure, 1 stands for the activity of reading the sheet of paper, 2 the activity of moving the sheet of paper, 3 the activity of smoking with the left hand, 4 the activity of drinking with the right hand, 5 the activity of manipulating the sheet of paper with the left hand and 6 the activity of manipulating the sheet of paper with the right hand. Parallelization of actions shown in this figure and mutual exclusion of behaviours are automatically handled by the scheduler. It exploits all propositions of transitions of state machines describing behaviours. For example, at time $30-40$, interruption of smoking behaviour, whereas its priority is active, is due to request of left hand resource by behaviour consisting in manipulating the sheet, which has a greater priority. This organization of behaviours has been automatically generated by the scheduler such as the overall realization of the example (Cf. figure 12).

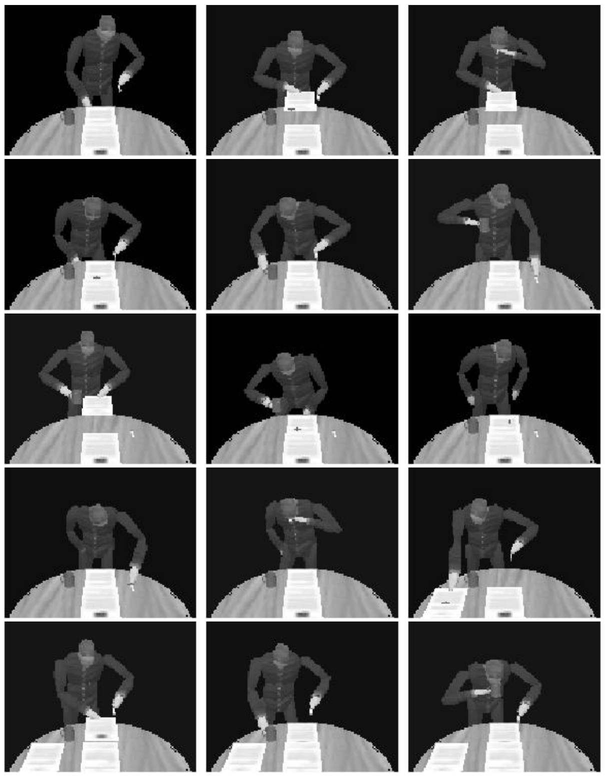

Figure 12: Behavioural Coordination Example.

7. CONCLUSION

We have presented in this paper a generic approach to integrate the management of resources and priority levels into HPTS, our formal model. In the contrary of some previous approach, it is not necessary to specify exhaustively all behaviours that are mutually exclusive; this is done implicitely just by attaching resources to nodes and a priority function to each state machine, and by using a scheduler. The example has illustrated the advantage of the scheduling system which allows to describe independently all behaviours with their different possibilities of adaptation. During running phase, the adaptation of all running behaviours is automatic. Moreover, consistency is ensured because the scheduler can only exploit consistent propositions of transition for each behaviour depending on the others. By now, our scheduler is able to handle a fixed number of resources (about ten for real time constraints) declared at compilation time. An extension will consist in allowing resource declaration at runtime in order to handle external resources. Another extension will be to connect this work to a higher level of reasoning process, in order to determine automatically which behaviour should be activated or inhibited.

8. REFERENCES

[1] O. Ahmad, J. Cremer, S. Hansen, J. Kearney, and P. Willemsen. Hierarchical, concurrent state machines for behavior modeling and scenario control. In *Conference on AI, Planning, and Simulation in High Autonomy Systems*, Gainesville, Florida, USA, 1994.

[2] N. Badler, B. Reich, and B. Webber. Towards personalities for animated agents with reactive and planning behaviors. *Lecture Notes in Artificial Intelligence, Creating Personalities for synthetic actors*, (1195):43–57, 1997.

[3] V. Decugis and J. Ferber. Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. In *Autonomous Agents'98*, pages 354–361, Minneapolis, USA, 1998. ACM.

[4] S. Donikian. HPTS: a behaviour modelling language for autonomous agents. In *Fifth International Conference on Autonomous Agents*, Montreal, Canada, May 2001. ACM Press.

[5] S. Donikian and E. Rutten. Reactivity, concurrency, data-flow and hierarchical preemption for behavioural animation. In E. B. R.C. Veltkamp, editor, *Programming Paradigms in Graphics'95*, Eurographics Collection. Springer-Verlag, 1995.

[6] A. Finzi, F. Pirri, and R. Reiter. Open world planning in the situation calculus. In *AAAI/IAAI*, pages 754–760, 2000.

[7] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *SIGGRAPH'99*, pages 29–38, Los Angeles, Aug. 1999.

[8] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990.

[9] J. C. Meyer and P. Schobbens, editors. *Formal Models of Agents*, volume 1760 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.

[10] A. Nareyek. A planning model for agents in dynamic and uncertain real-time environments. In *AIPS-98 Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, pages 7–14, 1998.

[11] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[12] H. Noser and D. Thalmann. Sensor based synthetic actors in a tennis game simulation. In *Computer Graphics International'97*, pages 189–198, Hasselt, Belgium, June 1997. IEEE Computer Society Press.

[13] A. Sloman. What sort of control system is able to have a personality. In R. Trappl and P. Petta, editors, *Creating Personalities for Synthetic Actors*, volume 1195 of *Lecture Notes in Artificial Intelligence*, pages 166–208. Springer-Verlag, 1997.