The Orchestration of Behaviours using Resources and Priority Levels

Fabrice Lamarche, Stéphane Donikian

The Orchestration of Behaviours using Resources and Priority Levels

F. Lamarche[1] and S. Donikian[2]
[flamarch, donikian]@irisa.fr
IRISA, Campus de Beaulieu, 35042 Rennes, FRANCE

Abstract.
Reproducing daily behaviours requires the ability to schedule behaviours depending on resources (body parts for example) and priority (intentions or physiological parameters) constraints. A simple way is to say that behaviours which are using the same resources are mutually exclusive. This approach is not sufficient to achieve realism purpose, as in real life, we are able to combine them in a much microscopic way. All day long, we mix different behaviours, as for example 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 is created. 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, 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 implicitely by attaching resources to nodes and a priority function to each state machine, and by using a scheduler.

Introduction

The goal of behavioural models is to simulate autonomous entities like organisms and living beings. The issue addressed in our work concerns the specification of a general formalism for behaviour modeling based on psychological studies and compatible with real-time constraints. Information needed to describe the behaviour of an entity depends on the nature of this entity. No theory exists for determining either 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[12] posits that one way to approach sufficiency is by modeling human cognition in computational layers or bands. Reproducing daily behaviours requires to schedule behaviours depending on resources (body parts for example) and priority (intentions or physiological parameters) constraints. A simple way is to say that behaviours which are using the same resources are mutually exclusive. This approach is not sufficient to achieve realism purpose, as in real life, we are able to combine them in a much microscopic way. All day long, we mix different behaviours, as for example 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 is created. This solution becomes rapidly too

[1]University of Rennes I
[2]CNRS

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 becomes possible to describe behaviours independently and to adapt automatically their execution when they are running in parallel, with respect to their priorities.

In the next section, related works are presented, while section three focuses on the HPTS model. The integration of resources and priority levels and the overview of the scheduler are presented in section four. Finally section five focuses on an example to illustrate advantages of this new approach.

1 Related Works

Behavioural models have been developed to describe the human behaviour in specific tasks. The 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 internal state and that of others. The center of such a deliberative agent is its own representation of the world which includes a representation of his mental state and the one of other agents which he is currently interacting with[9]. To achieve such a purpose, Badler et al.[3] 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, exactly like in the sensor effector approach described above. PaT-Nets are essentially finite state automata that can be executed in parallel (for example the control of the four fingers and of the thumb for a grasping task). The planner queries the state of the database through a filtered perception to decide how to elaborate the plan and to select an action. More recently they have introduced Parameterized Action Representation (PAR) to give a description of an action, and these PARs are directly linked to PaT-Nets. It allows a user to control Autonomous Characters actions with instructions given in natural language[4]. In this system, like in others[13], the action is directly associated with each node, which does not allow an explicit management of concurrency.

A lot of models have also been proposed for human like minds in the agent community[11]. 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[15] proposed an architecture of an intelligent agent in different layers (reflexes, automatic processes, resource-limited reflective management processes, meta-management processes), involving different routes through the system form 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. F. Brazier et al. [5] proposed a model of a rational agent using notions such as beliefs, desires and intentions. In their task hierarchy, beliefs and desires influence each other reciprocally, and they both influence intentions and commitments. S. Ambroszkiewicz and J. Komar[2] distinguish six parts in an agent model: perception, desire, knowledge and belief, rational behaviour, reasoning process and intention, and they propose a formal model based on this decomposition.

V. Decugis and J. Ferber[6] 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[10] into hierarchical ASMs. At the bottom of the hierarchy, basic reflexes are found, such as reflex movements orientation and basic perceptive mechanisms, while higher levels integrate more complex behaviours. B. Rhodes [14] has proposed another extension of ASM, called Phish-Nets. This model permits to use parameterized actions and to specify relations between them (inhibiting and preceding). In such models, reactive planning is possible, but the main drawback is the need of an exhaustive specification of all possible interactions between actions.

According to Newell, our goal is to build a model which will allow some adaptative and flexible behaviour to any entity evolving in a complex environment and interacting with other entities. Interactive execution is also fundamental. 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 pre-empting sub-behaviours). HPTS[8], as HCSM[1], is a model based on hierarchical concurrent state machines, and it offers a set of programming paradigms which permit to address hierarchical concurrent behaviours. HPTS offers also the ability to manage time informations (such as reaction time, state frequency, delay, minimal and maximal durations), undeterministic choices[7].

2 HPTS

HPTS[8] 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 by using state machines. To allow an agent to manage concurrent behaviours, sub-agents are organized inside 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 behavioural programming language which fully implements the HPTS formalism. As this paper focuses on the integration and management of resources and priority levels, the behavioural description language is not described in details. For a complete description of the model (except for resources and priorities) refers to [7]. 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 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. A state accepts an optional duration parameter which stands for the minimum and maximum amount of time spent in the state. 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 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

```
SMACHINE Id ;
{
    // Parameters
    PARAMS type Id {, type Id}* ;
    // Variables
    VARIABLES { {type Id ;}* }
    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.

the transition, and a *write-expr* which is a list of the generated events and basic activity primitives on the state machine. The body of a transition (C++ code) is executed after the action part. As for the body part of a state, it is possible to call extern functions or methods and to access to the value of outputs of sub-state machines. Afterwards, C++ code for our simulation platform is generated. It is totally encapsulated: all transitions systems are included in their own class directly inheriting from an abstract state machine class which provides pure virtual methods for running the state machines and debugging methods. An interpreter has also been implemented, which is very useful for the behaviour specification phase as it allows to modify state-machines during the execution phase.

3 Behaviour synchronization

In order to synchronize behaviours and to allow efficient mixing of behaviours in accordance with their relative importance, notions of resources, degrees of preference and priorities have been introduced. Resources allow to describe exclusions between behaviours while degrees of preference are used to describe different possible realizations or possibilities of adaptation of a behaviour. Using this information, a scheduler automatically synchronize the different behaviours according to their respective priorities.

3.1 Resources

For all state machines running in HPTS hierarchy, a set of resources is defined. Those resources are considered as semaphores, thus they are used for mutual exclusion. A set of resources is associated to each node of a state machine; it contains all resources used by a node. Hence, resource allocations are adapted to the state machine granularity. This type of description allows to handle resource allocations automatically:

- Entering a node implies that its associated resources are taken.

- Exiting a node implies that its resources are released.
- A resource will be kept if two nodes connected by a transition use it.

As resources are semaphores, one constraint has to be respected while running parallel state machines: all nodes executed in parallel have to use different resources. Using this constraint it becomes possible to synchronize the execution of parallel state machines according to the resources they use, by only authorizing transitions in nodes which do not use allocated resources.

Given that several parallel state machines are using the same set of resources, the problem of dead lock has to be studied. A dead lock occurs when the dependency graph of resources is cyclic. Therefore, in order to ensure maximum security and to make synchronization easier to handle, it is necessary to provide a mechanism ensuring that no dead lock can arise. While compiling the different state machines, information about resources allocation dependencies are pre-compiled. Thus, at runtime, the scheduler is able to use this information in order to check upon the fact that executing two nodes in parallel can not lead to a deadlock. This mechanism is very useful because it allows describing behaviours without precise knowledge on other behaviours.

As HPTS is a hierarchical model, each state machine can create sons and wait for their ending; this type of synchronization creates dependencies between state machines. Consequently, it exists risks of dead lock if a state machine using common resources with its son waits until its ending. Thus, another constraint has been added: resources used by a state machine must be different than resources used by its descendants. Respecting this constraint ensures that all descendants can be executed and terminated before the ending of their ascendants. As structure of state machines is known before runtime, this constraint can be checked while compiling state machines. By now those resources are used to describe internal resources of the agent like its hands, eyes or legs. They allow synchronizing behaviour in accordance to body parts they use.

3.2 Degrees of preferences

The notion of degree of preference has been introduced in order to provide the ability to describe different possibilities of adaptation of a behaviour, depending on resources availability.

A degree of preference (p) is associated to each transition of a state machine. It is a real coefficient with value in interval $[-1; 1]$. This coefficient corresponds to the state machine proclivity to use this transition when the associated condition is true. Depending on its value, it has different meanings:

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

This coefficient allows to concentrate all information about a specific behaviour into one state machine. Due to state machine structure, the concentration ensures consistency during the realization of the behaviour.

Let consider the state machine of figure 2; transitions are labeled with their associated conditions and degrees of preference. It describes a behaviour consisting in moving an object. While moving an object, eyes are necessary when taking the object and putting it somewhere; but while moving it, it is possible to focus on the object or

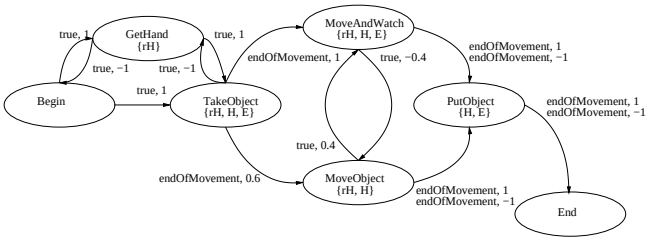

Figure 2. Moving object behaviour.

to look at something else. Transition starting from state *MoveAndWatch* and ending by *MoveObject* has a degree of preference of -0.4 which specifies that, by default, this transition should not be used, except if another behaviour needs to take eyes resource. The reciprocal transition has a degree of preference of 0.4 according the same positive preference to return into state *MoveAndWatch*. Using degrees of preference, this behaviour is totally described. Then the scheduler will choose to force behaviour to transit in the state which does not use eyes or to stay in the state which uses the eyes if no other behaviour needs the eyes.

3.3 Priority

The notion of priority has been introduced to provide the ability to specify the importance of a behaviour relatively to others. A priority function is associated to each state machine running in HPTS hierarchy. 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:

- $priority > 0$: the behaviour must be achieved, and is adapted to the current context. This value can be interpreted as a coefficient of adequacy between context and behaviour.
- $priority < 0$: the behaviour is inhibited, the value can be interpreted as a coefficient of inadequacy between context and behaviour.

This function is user defined. Thus, it can be correlated to different parameters, such as for example:

- physiological parameters;
- a value related to a plan generated by a rational model, in that case, it is correlated to the importance of the goal this action contributes to satisfy:
- a stimuli related to the environment to handle reflex behaviours.

This priority function provides an easy way to control the behaviour realization.

3.4 Scheduling

Notions explained below are used to create a scheduler. It allows to schedule different parallel behaviours in such a way that behaviours having the greatest priority will be favored in their execution and will automatically adapt their execution, if it is possible, to the other ones which are running. State machines are executed at a fixed

frequency. Thus, a scheduling is computed at each time step. At the beginning of the time step, scheduler collects information from each state machine. This information is compounded of the current state of a state machine plus all accessible nodes (i.e. nodes ending a transition with a true condition and starting from current node) ; this results in the creation of a set of possible transitions.

Then the scheduler computes a weight for each transition of this set. Let note $prio$ the current priority of the considered state machine, p the degree of preference associated to the transition ending with the state e. The weight (W) associated to this transition is computed as follow:

$$W = prio * p \tag{1}$$

Let consider a weight W associated to a proposition, this weight has different meanings:

- $W > 0$: state machine is proned to transit in the state associated to the proposition. Two cases can arise:
  - $(prio > 0) \wedge (p > 0)$: transiting to this node favors the accomplishment of the behaviour.
  - $(prio < 0) \wedge (p < 0)$: the behaviour is inhibited; transitions that conduct to a coherent stop of the behaviour have to be favored.
- $W < 0$: state machine is not proned to transit in this node. Two cases can arise:
  - $(prio > 0) \wedge (p < 0)$: transiting in this state does not favor the realization of the behaviour. This case can be used for proposing 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.
  - $(prio < 0) \wedge (p > 0)$: the behaviour is inhibited, this case can be used to propose an other possibility of stopping behaviour using less resources.
- $W = 0$: state machine is indifferent to transit in this node.

Once weights associated to each proposition of transition of the state machines are computed, the scheduler is searching for a combination of propositions between all state machines respecting resource constraints (no resource conflict and no possible deadlock) and maximizing the sum of associated weights. Therefore, behaviours having the greatest priorities will be favored in their execution while those having lowest priorities will release their resources, if possible and in a consistent way. Adaptation between concurrent behaviours becomes automatic when the concurrency concerns the sharing of common internal resources.

4 Example

Let us consider a complex behaviour consisting in drinking a coffee and smoking a cigarette while reading a newspaper. This kind of combination of multiple behaviours can not be directly handled by systems using mutual exclusion on behaviours which use themselves common resources. Moreover, a specific behaviour has to be created for systems using only semaphores synchronization without notions of priority and adaptation. In this section, we will study this example and show how the scheduler,

helped with notions of resources, degrees of preference and priorities, is able to reproduce this behaviour just by describing independently the three sub-behaviours and their associated priority functions.

4.1 Behaviour description

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 transition to be freed, for instance, putting down an object to free the hand resource. Thus, in order to free the hand resource Hr/Hl, resource rHr/rHl have to be taken. Then propositions of transitions in states only using resource Hr/Hl propose a sequence of actions freeing hand resources.

The behaviour is compounded of three sub behaviours: read a newspaper, drink a coffee and smoke a cigarette. All state machines used to solve this problem are presented in figures 3, 4 and 5. 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.

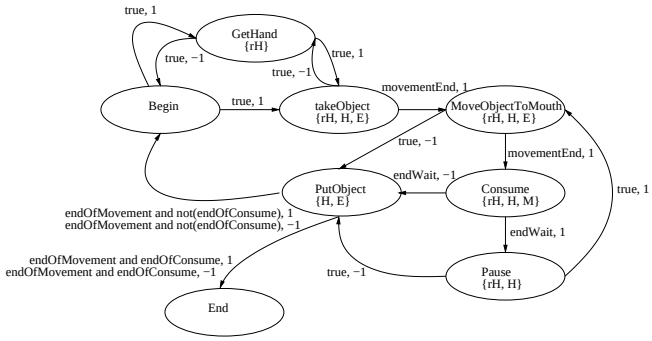

Figure 3. Common behaviour for drinking and smoking.

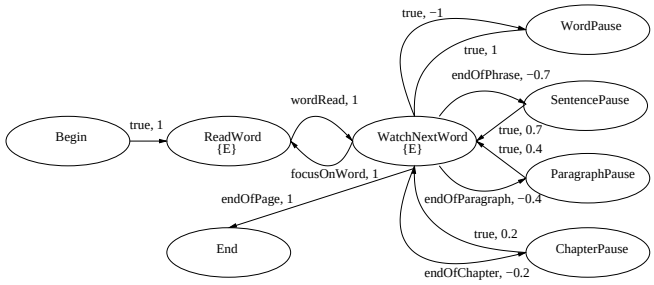

Figure 4. Reading behaviour.

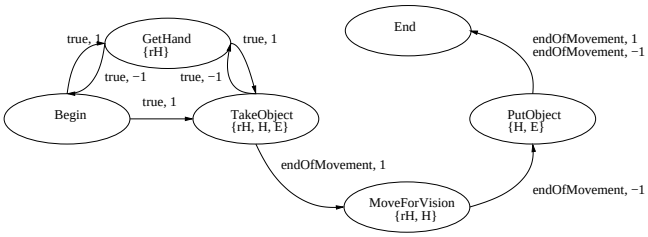

Figure 5. Behaviour handling hands while reading.

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 allowing specifying 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:

$$\begin{cases} p(t) = p(t - dt) + dp_1 \times dt \; if \; not(consuming) \\ p(t) = p(t - dt) - dp_2 \times dt \; if \; consuming \end{cases} \tag{2}$$

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. Then, when instantiating state machines corresponding to those behaviours, dp_2 becomes a parameter as well as the priority function which is linked to the thirst or the need of nicotine.

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.

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.

4.2 Scheduling the example

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 2. The priorities have been defined as follow:

- Moving sheet, followed by reading the sheet have a constant priority of 2.0.
- Manipulating the sheet while reading have a constant priority of 1.5 and can be realized with right or left hand.
- Drinking and smoking behaviours have a variable priority function. It is the function given in equation 2. Their increase rates are respectively set to 0.02/0.05 per second whereas their decrease rates are respectively set to 0.08/0.07 per second. Moreover, corresponding state machine stay in state Consume for a maximum time of one second, and stay in state Pause for a minimum of 10 seconds.

During execution phase, moving sheet behaviour is followed by manipulating and reading sheet behaviours, while drinking and smoking behaviours are continually running.

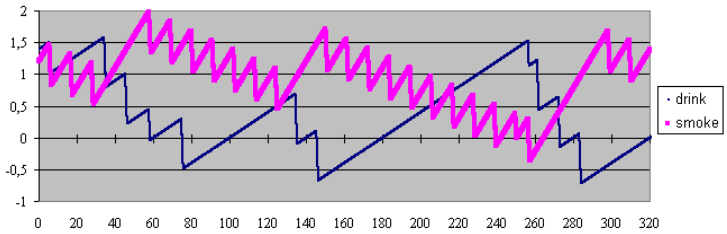

Figure 6. Evolution of drinking and smoking priorities during simulation. x axis corresponds to elapsed time in seconds, y axis corresponds to the priority value.

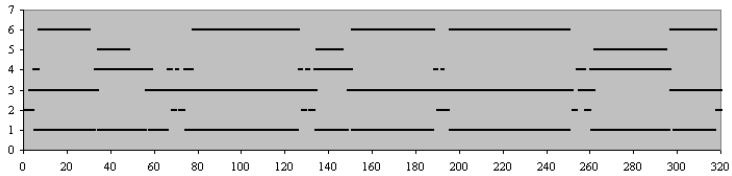

Figure 7. Activity of behaviours during simulation.

Figure 6 describes evolution of drinking and smoking priorities during the simulation. By default thirst and need of nicotine levels are increasing. Decrease is due to consumption linked to drinking and smoking behaviours. Note that thirst and need of nicotine never decrease at the same time, due to mutual exclusion on mouth noticed in node Consume of the corresponding state machine.

Figure 7 shows 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 manipulation 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 8).

The overall example has been designed in one day, helped by a high level pilot allowing to control character animation through simple primitives. It shows the advances of the scheduling system which allows to describe independently all behaviours with their different possibilities of adaptation. During running phase, their adaptation to all other running behaviours is automatic. Moreover, consistency is ensured because the scheduler can only exploit propositions of transition of each state machine which are supposed to be consistent.

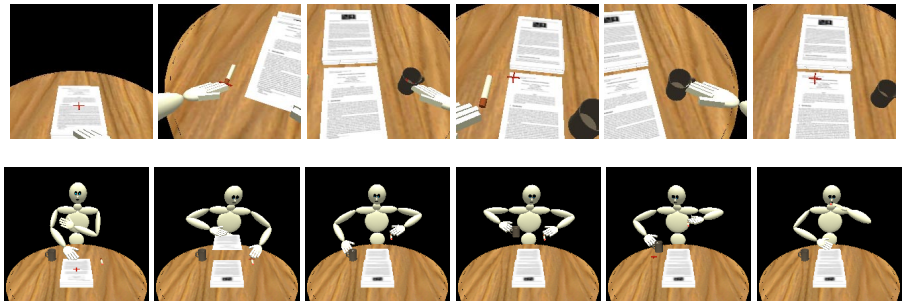

Figure 8. Behavioural Coordination Example.

5 Conclusion

We have presented in this paper a generic approach to integrate the management of resources and priority levels into HPTS, our formal model. This formal model has been implemented in a description language which is able to generate efficient C++ code for GASP, our Simulation Platform. The behavioral model allows us to describe, in a same way, different kinds of living beings, and to simulate them in the same virtual environment, while most of behavioral models are presently restricted to the animation of one model in a specific environment.

Another important point is that our behavioral model has been built to generate dynamic entities which are both autonomous and controllable, allowing us to use the same model in different contexts and moreover with different levels of control. Resources and priorities are described in an easy way, and behaviour incompatibilities are automatically detected. The scheduling algorithm enables us to combine together and to orchestrate several behaviours, depending on the human character intentions and desires.

In the contrary of some previous approach, it is not necessary to specify exhaustively all behaviours that are mutually exclusive; this is done implicitly just by attaching resources to nodes and a priority function to each state machine, and by using a scheduler. Actually, our scheduler is able to handle a fixed number of resources declared at compilation time. An extension would be to allow resource declaration at runtime in order to handle external resources. Another extension will be to connect this work to a higher level of reasoning.

Video sequences related to the example can be found at http://www.irisa.fr/prive/donikian/resources/.

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. S. Ambroszkiewicz and J. Komar. *Formal Models of Agents*, volume 1760 of *Lecture Notes in Artificial Intelligence*, chapter A Model of BDI-Agent in Game-Theoretic Framework, pages 8–19. Springer, 2000.
3. N.I. Badler, B.D. Reich, and B.L. 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.
4. R. Bindiganavale, W. Schuler, J. Allbeck, N.I. Badler, A.K. Joshi, and M. Palmer. Dynamically altering agent behaviors using natural language instructions. In C. Sierra, M. Gini, and J.S. Rosenschein, editors, *International Conference on Autonomous Agents*, pages 293–300, Barcelona, Spain, June 2000. ACM Press.
5. F. Brazier, B. Dunin-Keplicz, J. Treur, and R. Verbrugge. *Formal Models of Agents*, volume 1760 of *Lecture Notes in Artificial Intelligence*, chapter Modelling Internal Dynamic Behaviour of BDI Agents, pages 36–56. Springer, 2000.
6. 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.
7. S. Donikian. HPTS: a behaviour modelling language for autonomous agents. In *Fifth International Conference on Autonomous Agents*, Montreal, Canada, May 2001. ACM Press.
8. S. Donikian and E. Rutten. Reactivity, concurrency, data-flow and hierarchical preemption for behavioural animation. In E.H. Blake R.C. Veltkamp, editor, *Programming Paradigms in Graphics'95*, Eurographics Collection. Springer-Verlag, 1995.
9. J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *SIGGRAPH'99*, pages 29–38, Los Angeles, August 1999.
10. P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990.
11. J.J. Ch. Meyer and P.Y. Schobbens, editors. *Formal Models of Agents*, volume 1760 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.
12. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
13. 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.
14. B. J. Rhodes. *PHISH-Nets : Planning Heuristically In Situated Hybrid Networks*. PhD thesis, Massachusetts Institute of Technology, 1996.
15. 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.