

# HPTS: A Behaviour Modelling Language for Autonomous Agents

Stéphane Donikian<sup>\*</sup>  
IRISA / SIAMES team  
Campus de Beaulieu  
F-35042 Rennes, FRANCE  
donikian@irisa.fr

## ABSTRACT

Behavioural models offer the ability to simulate autonomous agents like organisms and living beings. Psychological studies have shown that human behaviour can be described by a perception-decision-action loop, in which the decisional process should integrate several programming paradigms such as real-time, concurrency, and hierarchy. Building such systems for interactive simulation requires the design of a reactive system treating flows of data to and from the environment, and involving task control and preemption. Accordingly, in this paper we address the adequateness to the decisional part of the behavioural model of Hierarchical Parallel Transition Systems (HPTS).

## Keywords

Believability, Synthetic Agents, Agent Architectures, Behaviour Modelling Language

## 1. INTRODUCTION

The goal of the behavioural model is to simulate autonomous entities like organisms and living beings [4, 5]. A behavioural entity has the following capabilities: perception of its environment, decision, action and communication [17, 7, 1]. Most behavioural models have been designed for some particular examples in which possible interactions between an object and its environment are very simple: sensors and actuators are reduced to minimal capabilities. Another point which is generally not treated is the notion of time. The issues addressed here are wider; we aim to describe a general formalism of behaviour modelling based on psychological studies and compatible with real-time constraints.

In this paper, we will start by analyzing the decisional architectures defined by behavioural psychologists and which may be compatible with real-time. Then we will make a

<sup>\*</sup>CNRS

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.

AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.  
Copyright 2001 ACM 1-58113-326-X/01/0005 ...\$5.00.

short review of the techniques employed in the realm of behavioural animation. After discussing the pros and cons of known approaches, we will introduce the HPTS formalism and show its usefulness thanks to a behaviour description language. Finally, we will illustrate how it has been used in different applications.

## 2. PSYCHOLOGICAL REQUIREMENTS

Mallot [24], among others, describes the interactions between a behavioural entity and its environment (see Figure 1). The overview of the system is a perception-decision-action loop. The first arrow from sensors to actuators is called homeostasy (i.e. the internal regulation feedback) whereas the second arrow stands for the actions required for perception (e.g. turning the head to see something which should be on the left).

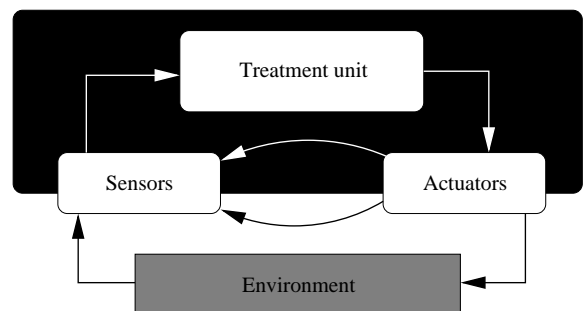


Figure 1: The human organism and its environment.

Information needed to describe the behaviour of an entity depends on the nature of this entity. 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 [27] posits that one way to approach sufficiency is by modelling human cognition in computational layers or bands. He suggests that these computational layers emerge from the natural hierarchy of information processing. Lord [21] introduces several paradigms about the way the brain works and controls the remainder of the human body. He explains that human behaviour is naturally hierarchical, that cognitive functions of the brain are run in parallel. Moreover cognitive func-

tions are different in nature: some are purely reactive, while others require more time. Execution times and frequencies of the different activities are provided. Newell asserts that these levels are linked by transferring information across hierarchical levels, and that each of them operates without having any detailed knowledge of the inner workings of processes at other levels. All that is required is a transfer function to transform the information produced by one level into a form that can be used by another. Particularly important is the notion that symbolic activities occur when they are locally based on problem spaces constructed on a moment-to-moment basis. According to Newell [27], the constraints of a computer behavioural model are the following:

- adaptative and flexible behaviour;
- real-time interaction with the environment;
- complex and rich environment, i.e., a behavioural entity should be able to perceive its environment, should have a knowledge database and several degrees of freedom of action on the environment;
- use of symbols and abstraction;
- autonomy in a social background (ie. possible social interactions with other entities);
- self-consciousness and perception;
- ability to learn;
- constructable as a neural network by an embryological growth process, and having arisen through evolution.

The last three constraints expressed in the last item will not be addressed in this paper. In fact we essentially focus on the first six constraints. Therefore, 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 led us to state that paradigms required for programming a *realistic* behavioural model are the following:

- 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 on transitions in the meta-behaviour as a kind of exception or interruption. It means also that sub-behaviours can notify the meta-behaviour of their activity;
- frequency handling for execution of sub-behaviours. This provides the ability to model reaction times in perception activities.

### 3. RELATED WORKS

Behavioural animation consists of a high level closed control loop [18, 23, 12, 10] which offers the ability to simulate autonomous entities [4, 31]. Such actors are able to perceive their environment, to communicate with others [7] and to execute some actions, such as walking in the street or grasping an object, according to the nature of the environment and with their intentions. Different approaches have been

studied for the decision part of behavioural models in animation: sensor-effector or neural networks (see e.g. [35]), behaviour rules (see e.g. [34]), Finite Automaton Approach (see e.g. [8]). Most of these systems have been designed for some particular examples, in which modularity and concurrency are not necessary. Behavioural entities have only one activity line and because interactions between an object and its environment are very simple; sensors and actuators are reduced to minimal capabilities which, most of the time, only allow obstacle avoidance in a 2D or 3D world.

Another point which is generally not treated is the notion of time, which is however necessary, either during the specification phase (memorisation, prediction, action duration) or during the execution phase (synchronisation of objects with different internal times). More recently, a second generation of behavioural models has been developed to describe human behaviour in specific tasks. As human behaviour is very complex, none of the preceding models could be applied. The common characteristics of these new models are reactivity, parallelism and different abstract levels of behaviours. In [11], authors describe a multi-agent development environment named DASEDIS and use it to describe the behaviour of a car driver. In this environment, data provided by sensors are evaluated in the decision module so that when a state change occurs, an appropriate script is chosen for execution. Each script describes the sequence of actions to perform in order to achieve specific goals or to respond to events. Scripts can be organized in a hierarchical manner and are described by using a graphical notation. Each script is in fact equivalent to a Directed Acyclic Graph (DAG), in which leaves represent *acting* behaviours and other nodes correspond to branching evaluation nodes (priority function or conditional expression). In [28], a tennis game application is shown, including the behaviour of players and referees. A stack of automata is used to describe the behaviour of each actor. It is possible to decompose a complex behaviour into sub-behaviours by using this stack: one automaton can push itself with the current state on the stack and switch to the new automaton which is able to reach the specific sub-goal. In the Motivate product proposed by Motion Factory for the Game Design Market, they have also introduced Hierarchical Finite State Machines and actions associated with the states and transitions can be described by using an object-based programming language named Piccolo [20].

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 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 which he is currently interacting. To do this, 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 linked directly to PaT-Nets. It allows a user to control Autonomous Characters actions by instructions given in natural language[6]. In all these systems, the action is directly associated with each node, which does not allow the management of concurrency. HCSM [1] and HPTS [17], both based on a hierarchy of concurrent state machines, offer a set of programming paradigms, which permit to address hierarchical concurrent behaviours. HPTS offers also the ability to manage time information (such as state frequency, delay, minimal and maximal durations) and undeterministic choice [26].

A lot of models have also been proposed for human like minds in the agent community. 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 [30] 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 from perception to action. In his theory, automatic processes have dedicated portions of the brain and can operate in parallel whenever they need to, while different management processes have to share a common working memory, and their parallelism is then restricted. Over the last few years, multi-agent systems have been a major focus of research in Artificial Intelligence [25]. F. Brazier et al. [9] propose a model of a rational agent using notions such as beliefs, desires and intentions. This model includes task composition, information exchange between tasks, sequencing of tasks, task delegation and knowledge structures. 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 [13] 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 [22] into hierarchical ASMs. At the bottom of the hierarchy, basic reflexes are found, such as reflex movement 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.

## 4. A MODEL FOR BEHAVIOURAL ANIMATION

### 4.1 Hierarchical Parallel Transition Systems

The Decisional model consists of a reactive system, which is composed of a hierarchy of agents (possible behaviours). 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 can be defined by the following tuple  $\langle S, \Gamma, IS, OS, CP, LV, IF, MB \rangle$  in which:

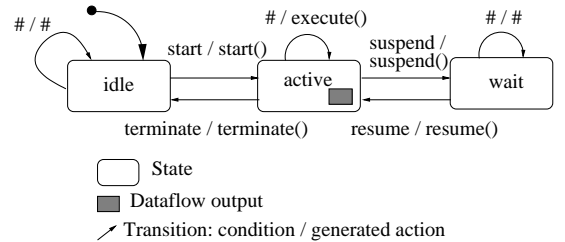
- S** : is a set of sub-state machines,
- $\Gamma$**  : is the activity function,
- IS** : is a set of Input Signals,
- OS** : is a set of Output Signals,
- CP** : is a set of Control Parameters,
- LV** : is a set of Local Variables,
- IF** : is the integration function,
- MB** : is a Mail-Box.

Each agent of the system is either an atomic agent ( $S = \emptyset$ ), or a composite agent. An activity parameter is associated to each state machine which corresponds to its current status. This parameter is accessible by using the function  $status(state, instant)$ , which has three possible values:

$$(\forall s \in S)(\forall k \in \mathcal{N}), status(s, t_k) \in \{active, idle, wait\}$$

There is also an optional duration parameter which is used to force the state machine to keep a state active for a minimum duration. This allows the modelling of reaction times. On the contrary, the maximum duration of the active phase can also be specified. When this value is reached, an event is generated in order to warn the system that something that should have happened actually did not.

The status of a state machine is described by an automaton (cf Figure 2), in which transitions between the three statuses depend on events produced either by the meta-state (preemption) or by the state itself (auto-termination). A specific method is attached to each transition (start(), suspend(), resume() and terminate() the task) and also to the active status (execute()).



**Figure 2: Status Management Automaton.**

Activity of a state evolves during the simulation, and this is determined by an activity function  $\Gamma$ . This function has four parameters and returns the new status:

$$(\forall s \in S)(\forall k \in \mathcal{N}^*), status(s, t_k) = \Gamma(status(s, t_{k-1}), IS, CP, LV)$$

This function permits to represent some transitions between different sub-state machines, but more than one sub-state can be active at each instant (concurrency). This function also handles hierarchical preemption, by the

fact that one argument of the function is a set of Control Parameters (CP) which allows to deal with internal events such as “sub-state  $i$  is terminated ” or external events such as “the traffic lights become red”. Input and output parameters are continuous signals of standard type (e.g. integer, real, boolean for example). The value of an Output signal is undetermined when the state machine is *idle* or *suspended*. Outputs are used to return actions proposed by the state machine. Local variables are standard type variables, whose value is computed by using the  $\Omega$  function. Local variables can either retain their values between activations or be reinitialized on each reactivation (*started* status):

$$(\forall v \in V)(\forall k \in \mathcal{N}^*), \\ value(v, t_k) = \Omega(value(v, t_{k-1}), status(s, t_{k-1}), IS, CP)$$

The integration function has to manage the coherence of the actions proposed by the different sub-processes, and make a synthesis of them. This is in fact a function which takes as inputs the outputs of all sub-processes and delivers the value of the process outputs. In the case of concurrent behaviours proposed by different sub-state machines, this function has to make a choice and to deliver a unified behaviour as output of the state machine (cf figure 3).

$$OS = IF(output(S), LV, CP)$$

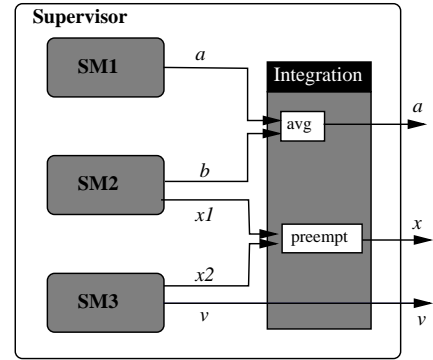
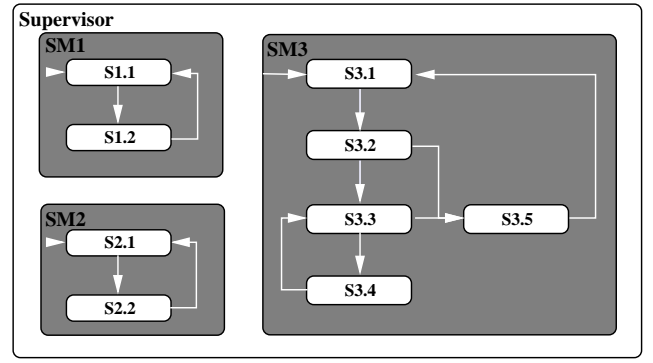
The Mail-Box is used to manage communication between state machines. A message is broadcast to all state machines, and only those concerned by this message will react. An example of a hierarchical state machine is given in Figure 3. The state machine *Supervisor* is a state machine that includes three concurrent state machines ( $SM1$ ,  $SM2$  and  $SM3$ ). These state machines are purely sequential and the activity function may be represented by transitions. On the bottom, the data-flows associated with the state machine are represented. Depending on the nature of the outputs, either numerical or symbolic functions may be applied in the integration function. In this example, the  $a$  output of *Supervisor* is the average of the  $a$  output of  $SM1$  and the  $b$  output of  $SM2$ , while the  $x$  output is the result of a preemption function applied to the  $x1$  output of  $SM2$  and the  $x2$  output of  $SM3$ .

## 4.2 Behaviour description language

Our model of a behavioural entity is based on the HPTS formalism, i.e. Hierarchical Parallel Transition Systems and data-flows. Though they may be coded directly with an imperative programming language like C++, we decided to build a language for behaviour description. Otherwise, the problem is that it quickly becomes quite difficult to update a complex state machine and therefore to reuse it in future developments. Moreover the code of the transition systems becomes unreadable or inefficient. This is why we propose a language that allows the description of both the hierarchical parallel state machines and their associated data-flows. This language is compiled and efficient C++ code is generated. The change of a transition condition is thus quite easy.

### 4.2.1 Architecture of the language

Our language fully implements the HPTS formalism. The goal is the (object-oriented) design of behavioural entities, at least as concerns their reactive part, and the connection with tasks that require more computation time.



**Figure 3: State-Transition and data-flow aspects of the state machine.**

In this section, we will provide a short description of the language. 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 made in the following way: the body of the declaration contains a list of states and a list of transitions between these states.

```
SMACHINE Id ;
{
  // Parameters Declaration
  PARAMS type Id {, type Id}* ;
  VARIABLES
  {
    // Variables Declaration
    {type Id;}*
  }
  // Outputs Declaration
  OUT Id {, Id}* ;
  INITIAL Id ;
  FINAL Id ;
  STATES
  {
    // States Declaration
    Id {[float {, float}] {RANDOM}};
    {
      // state body
    }
  }
}
```

Accepted types of data (*type* rule) are either float, integer, boolean or string, while *Id* corresponds to a string defining the name of a data. A state machine can be parameterized by a set of parameters (**PARAMS** part) that will be used to characterize the machine at its creation. Variables are local to a state machine. Only variables that have been declared as outputs can be viewed by the father state machine in the hierarchy. 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. The **RANDOM** keyword indicates that this state will make a random choice between all possible transitions that can be fired at a timestep, while in the normal mode, the first transition with a true condition will be fired.

```

TRANSITION Id ;
{
  ORIGIN Id ;
  EXTREMITY Id ;
  {DELAY float ;}
  {WEIGHT float ;}
  read-expression / write-expression {TIMEGATE} ;
  {{ //transition body }}
}

```

A transition is defined by an origin, an extremity, a transition expression, two optional parameters and a transition body. The value associated to the parameter **DELAY** is used to specify a delay between the instant at which the condition is chosen to be fired and the instant at which the transition is really fired (useful to take into account reaction time). The value associated to the parameter **WEIGHT** is used to specify the weighting rate of this transition for the random choice between all transitions that could be fired.

The transition expression consists of two parts: a *read-expression* which includes the conditions to be fulfilled in order to fire the transition, and a *write-expression* which is a list of the generated events and basic activity primitives on the state machine. Two execution modes are available: normal and timegate modes. In the normal mode, almost one transition can be fired per state machine at each timestep. In the timegate mode, all transitions that can be fired during the timestep will be fired. To stop the process, the keyword **TIMEGATE** is used in the transition, which means that no other transition can be fired during the same timestep in this state machine. The condition of a transition is a boolean expression and can be composed of:

- **HPTS(end, name)**: become true when the specified sub-state machine (*name*) is terminated;
- **HPTS(endall)**: become true when all sub-state machines are terminated;
- **MSG EVT(type)**: become true when the specified event (*type*) has been received;
- a logical combination of boolean conditions which can use local variables and boolean functions;
- **MAX\_FIRST**: true when the time spent in the initial state of the transition is higher than the maximal allowed duration of this state. This transition is taken into account before other possible valid transitions during the timestep;

- **MAX\_LAST**: true when the time spent in the initial state of the transition is higher than the maximal allowed duration of this state. This transition is taken into account only if no other transition can be fired during this timestep;
- **#**: this means that the condition is always true.

The first four instructions can be combined together in boolean expressions, while the last three should be used alone. The *write-expression* could contain the following instructions:

- **HPTS(start, *L\_name*, *SM\_name* [*parameters*])**: allow a new instance (*L\_name*) of a state machine (*SM\_name*) to be created. This new state machine will be added to the list of children of the current state machine;
- **HPTS(suspend, *L\_name*)**: allow a state machine to suspend the execution of one of its sub-state machines (*L\_name*);
- **HPTS(resume, *L\_name*)**: allow a state machine to restart the execution of one of its sub-state machines (*L\_name*);
- **HPTS(kill, *L\_name*)**: allow a state machine to kill one of its sub-state machines (*L\_name*);
- **MSG EVT(type)**: the event (*type*) is sent to all state machines (broadcast mode);
- **#**: this means that no action is associated to the transition.

The first five instructions can be combined together. 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 external functions or methods. The value of an output of a sub-state machine can be accessed by using the following instruction:

```
#SM_name(L_name, variable_name)
```

#### 4.2.2 Code generation

Once the behavioural entities have been described, the parser builds a state machine tree and performs some rewriting operations on it: for instance, a state *s* which has a minimum duration *d* is transformed into a couple of states (*s*<sub>1</sub>, *s*<sub>2</sub>) with the same actions; but state *s*<sub>1</sub> can only be followed by state *s*<sub>2</sub> after having waited for *d* seconds, whereas the transitions coming from state *s* are now coming from state *s*<sub>2</sub>. The same kind of rewriting operation is performed for maximum durations.

Afterwards, C++ code for our simulation platform GASP [14] is generated. It is totally encapsulated: all transition 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 (display of current state, transitions fired, value of the variables). The advantage of using the object model is that the behaviour simulation code is easy to use in any application. The emphasis in the code generation phase is put on efficiency and connectivity to existing C++ code. An interpreter has also been implemented, which is very useful for the behaviour specification phase as it allows state-machines to be modified during the execution phase with an increase of only ten percent on the execution time.

## 5. APPLICATIONS

HPTS have been used to model the behavioural part of autonomous characters in different kinds of applications, but all of them have been built upon GASP our General Animation and Simulation Platform, which is currently available on both Unix and Linux workstations, including real-time, parallelism and distribution features. GASP offers a modular programming environment, and the kernel of the simulation platform manages the data-flow and event based communications between modules and their synchronization (as modules may have different frequencies). Modules are organized in a simulation tree and, for example, an autonomous pedestrian entity is composed of a visual sensor module, a behavioural module and a biomechanical module. The user can also control the motion of one pedestrian by using a joystick with force feedback.



Figure 4: Interaction of pedestrians and cars.

Urban traffic has a high degree of complexity, as it requires interactions on the same thoroughfare between not only cars, trucks, cyclists and pedestrians, but also public transportation systems such as busses and trams. As our approach is modular, we have started to integrate all these transportation modes into GASP[16]. A set of tools has been developed to produce these models (see Figure 5) and HPTS is one of them. The mechanical aspect of the car is modelled with DREAM, our rigid and deformable bodies modelling system which generates numerical C++ simulation code for GASP. VUEMS is the acronym for Virtual Urban Environment Modelling System, and its main aim is to build a realistic virtual copy of urban environments in which we would perform behavioural simulations. VUEMS produces two complementary outputs: the 3D geometric representation of the scene and its symbolic representation used by sensors and behavioural entities [32]. The scene produced by VUEMS is loaded and is then available for use by all autonomous entities. First, sensors can determine visible objects in their environment and then the behavioural module can have access to the information on these visible objects.

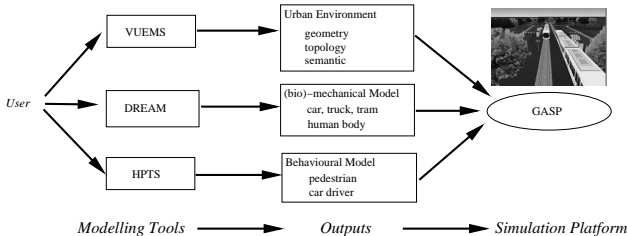


Figure 5: A set of modelling tools.

The twelve pictures of Figure 6 illustrate the crossing of two pedestrian crowds on a sidewalk. The camera viewpoint corresponds to the subjective view of one of the fifty pedes-

trians, which explains why the pedestrians arriving in the opposite direction change their trajectory. The behavioural model of pedestrians includes social and driving rules of interaction (minimize the interaction and choose in priority the left side to overtake), as explained in [33].

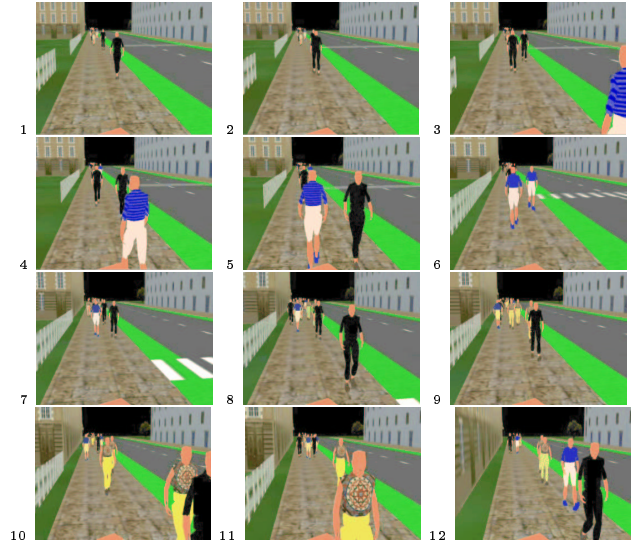
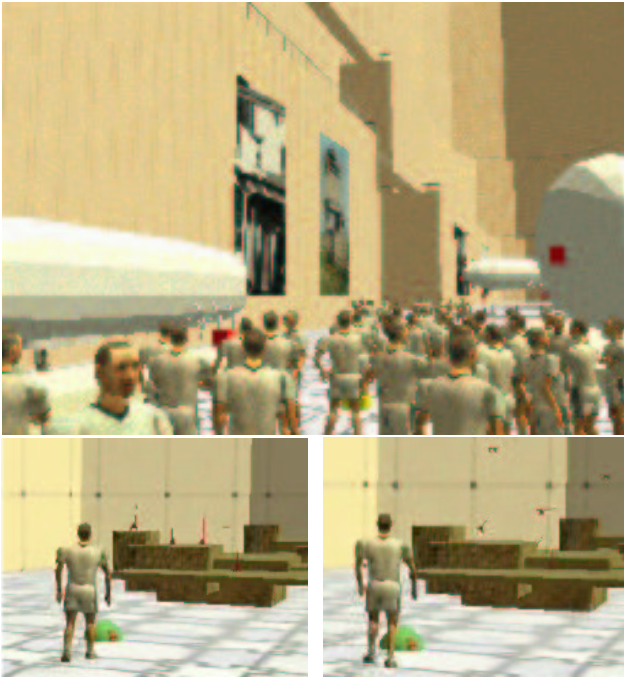


Figure 6: Pedestrians on a sidewalk.

Another application concerns a virtual museum visited by a group of autonomous characters and also inhabited by a cloud of suribirds (birds with the behaviour of surikats). This application, which is under development (cf figure 7), will be presented in June 2001 as part of the new image section of the Museum of Sciences in Paris, France. A set of specific characters is currently modelled: an attendant, a thief who wants to steal paintings and sculptures, a photographer who damages the colors of a painting when he takes a picture of it, a mother and her son who have conflicting desires. The purpose of this application is to propose to the real visitor an observation of life evolving in a virtual world and to see how the behaviour modification of one or more autonomous characters can affect this virtual life. This can easily be done due to the preemption mechanism, and the dynamicity and parameterization of state machines.

## 6. CURRENT WORKS

To reproduce daily behaviours it is necessary to schedule behaviours depending on resource (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, but in real life, we are able to combine them in a more microscopic way, as for example smoking, drinking and reading at the same time. That is why we are currently integrating resources and priority management inside HPTS. As yet, we have not had a visual application integrating these new features as it requires the management of affordances not only on space, like in the Urban Traffic Application [33], but also on objects, as proposed by M. Kallmann [19]. However, the algorithm developed for task scheduling is very promising as it is able to perform incremental planning and to avoid deadlocks. Unlike some previous approaches [29], it is not necessary to



**Figure 7: The Virtual Museum Application.**

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.

Despite the benefits of the language approach described earlier, the description of behaviours remains quite difficult for people who are not computer scientists. Therefore we are working on a higher level specification language, in order to allow behavioural specialists to specify and test their models in an interactive simulation. Another work in progress concerns the scenario language whose objective is to partially constrain the behaviour of autonomous characters to fulfill the scenario description [15]. This project is currently underway and is more than a simple graphical tool for drawing state machines, it includes a specification in a subset of natural language, taking into account action and motion verbs and spatio-temporal relations. This description is then translated into an internal scenario language, which uses HPTS in the TIMEGATE mode.

## 7. CONCLUSION

In this paper, we have presented the behavioural model, and more precisely the decisional part of this model. A formal model of a Hierarchical Parallel Transition System has been presented to describe *realistic* behaviours, which requires different programming paradigms: reactivity, concurrency, data-flow and hierarchical preemption. Then we have presented how this formal model has been implemented in a description language which is able to generate efficient C++ code for GASP, our Simulation Platform. The use of Hierarchical Parallel Transition Systems allows us to take into account several programming paradigms important to describe *realistic* behaviours. Because of the integration of our behavioural model in a simulation platform, we have also the ability to deal with real time during the specification and the execution phases.

Our main objective is real time simulations of several entities evolving in realistic informed environments. Many studies have been performed by psychologists to analyse human behaviour. The behavioural model allows us to describe, in the same way, different kinds of living beings, and to simulate them in the same virtual environment, while most behavioural models are currently restricted to the animation of one model in a specific environment. Another important point is that our behavioural 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.

## 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] 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. 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.
- [4] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating Humans : Computer Graphics Animation and Control*. Oxford University Press, 1993.
- [5] N. I. Badler, B. L. Webber, J. Kalita, and J. Esakov, editors. *Making them move: mechanics, control, and animation of articulated figures*. Morgan Kaufmann, 1991.
- [6] R. Bindiganavale, W. Schuler, J. Allbeck, N. Badler, A. Joshi, and M. Palmer. Dynamically altering agent behaviors using natural language instructions. In C. Sierra, M. Gini, and J. Rosenschein, editors, *International Conference on Autonomous Agents*, pages 293–300, Barcelona, Spain, June 2000. ACM Press.
- [7] B. Blumberg and T. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In *Siggraph*, pages 47–54, Los Angeles, California, U.S.A., Aug. 1995. ACM.
- [8] M. Booth, J. Cremer, and J. Kearney. Scenario control for real-time driving simulation. In *Fourth Eurographics Workshop on Animation and Simulation*, pages 103–119, Politechnical University of Catalonia, Sept. 1993.
- [9] 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.
- [10] D. Brogan, R. Metoyer, and J. Hodgins. Dynamically simulated characters in virtual environments. *IEEE Computer Graphics and Applications*, pages 58–69, 1998.
- [11] B. Burmeister, J. Doormann, and G. Matylis.

- Agent-oriented traffic simulation. *Transactions of the Society for Computer Simulation International*, 14(2), June 1997.
- [12] E. Cerezo, A. Pina, and F. Seron. Motion and behaviour modelling: state of art and new trends. *The Visual Computer*, 15:124–146, 1999.
- [13] 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.
- [14] S. Donikian, A. Chauffaut, R. Kulpa, and T. Duval. Gasp: from modular programming to distributed execution. In *Computer Animation'98*, pages 79–87, Philadelphia, USA, June 1998. IEEE.
- [15] S. Donikian, F. Devillers, and G. Moreau. The kernel of a scenario language for animation and simulation. In *Eurographics Workshop on Animation and Simulation*, Milano, Italia, Sept. 1999. Springer Verlag.
- [16] S. Donikian, G. Moreau, and G. Thomas. Multimodal driving simulation in realistic urban environments. In S. Tzafestas and G. Schmidt, editors, *Progress in System and Robot Analysis and Control Design*, pages 321–332. Lecture Notes in Control and Information Sciences (LNCIS 243), 1999.
- [17] 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.
- [18] 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.
- [19] M. Kallmann and D. Thalmann. Modeling objects for interaction tasks. In *Eurographics Workshop on Animation and Simulation*, Lisbon, Portugal, Sept. 1998. Springer-Verlag.
- [20] Y. Koga, G. Annesley, C. Becker, M. Svihura, and D. Zhu. On intelligent digital actors. In *IMAGINA'98*, 1998.
- [21] R. G. Lord and P. E. Levy. Moving from cognition to action : A control theory perspective. *Applied Psychology : an international review*, 43 (3):335–398, 1994.
- [22] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990.
- [23] P. Maes, T. Darrell, B. Blumberg, and A. Pentland. The alive system: Full-body interaction with autonomous agents. In *Computer Animation'95*, pages 11–18, Geneva, Switzerland, Apr. 1995. IEEE.
- [24] H. A. Mallot. Behavior-oriented approaches to cognition : theoretical perspectives. *Theory in biosciences*, 116:196–220, 1997.
- [25] J. C. Meyer and P. Schobbens, editors. *Formal Models of Agents*, volume 1760 of *Lecture Notes in Artificial Intelligence*. Springer, 2000.
- [26] G. Moreau and S. Donikian. From psychological and real-time interaction requirements to behavioural simulation. In *Eurographics Workshop on Computer Animation and Simulation*, Lisbon, Portugal, Sept. 1998.
- [27] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
- [28] 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.
- [29] B. J. Rhodes. *PHISH-Nets : Planning Heuristically In Situated Hybrid Networks*. PhD thesis, Massachusetts Institute of Technology, 1996.
- [30] 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.
- [31] D. Thalmann and H. Noser. Towards autonomous, perceptive, and intelligent virtual actors. In *Artificial Intelligence Today*, volume 1600 of *Lecture Notes in Artificial Intelligence*, pages 457–472. Springer, 1999.
- [32] G. Thomas and S. Donikian. Modelling virtual cities dedicated to behavioural animation. In M. Gross and F. Hopgood, editors, *EUROGRAPHICS'2000*, volume 19:3, Interlaken, Switzerland, Aug. 2000. Blackwell Publishers.
- [33] G. Thomas and S. Donikian. Virtual humans animation in informed urban environments. In *Computer Animation*, pages 129–136, Philadelphia, PA, USA, May 2000. IEEE Computer Society Press.
- [34] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Computer Graphics (SIGGRAPH'94 Proceedings)*, pages 43–50, Orlando, Florida, July 1994.
- [35] M. van de Panne and E. Fiume. Sensor-actuator networks. In J. T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 335–342, Aug. 1993.