A Programming Environment for Behavioural Animation

Frédéric Devillers, Stéphane Donikian, Fabrice Lamarche and Jean-François Taille
IRISA
Campus de Beaulieu
F-35042 Rennes, FRANCE
[fdeville,donikian,flamarch,jtaille]@irisa.fr

Abstract

Behavioural models offer the ability to simulate autonomous agents like organisms and living beings. Psychological studies have showed that the 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. Since a complete mental model based on vision and image processing cannot be constructed in real time using purely geometrical information, higher levels of information are needed in a model of the virtual environment. For example, the autonomous actors of a virtual world would exploit the knowledge of the environment topology to navigate through it. Accordingly, in this paper we present our programming environment for real-time behavioural animation which is compounded of a general animation and simulation platform, a behavioural modelling language and a scenario authoring tool. Those tools has been used for different applications such as pedestrian and car driver interaction in urban environments, or a virtual museum populated by a group of visitors.

1 Introduction

Behavioural animation consists in a high level closed control loop [5, 7, 25, 33], which offers the ability to simulate autonomous entities like organisms and living beings. Such actors are able to perceive their environment, to take decision, to communicate with others [4] and to execute some actions, such as walking in the street or grasping an object. Realistic behaviours of autonomous actors evolving in complex and structured environments can be obtained if and only if the relationship between the actor and its surrounding environment can be simulated. Information which must be extracted or interpreted from the environment depends on the abstraction level of the reasoning performed by autonomous actors.

The first development to put together research works done in the team has started in 1994. The first architecture of a general animation and simulation platform (GASP) can be found in [12]. We have then started to build both the kernel of this platform and a set of tools dedicated to the modelling of different aspects of an autonomous entity: geometry, dynamics, motion control, behaviour, artificial vision. The mechanical aspect is modelled with DREAM, our rigid and deformable bodies modelling system which generates numerical C++ simulation code for GASP [8]. For human motion, a bio-mechanical approach has been preferred [28]. HPTS concerns the modelling of the behavioural part of an actor [27] and is also used as an intermediate level for scenario authoring [13]. 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. 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. In the past years we have integrated all these transportation modes into GASP and applied this to different research projects [1, 14, 15, 21]. In the context of national research projects, GASP has already been used with success in other research labs in France. Our objective is now to make it available to the international community as an open-source version with an API in english, including documentation and tutorials. This new version of GASP (version 3.0) is now available . As GASP is a product already used by a company, we have decided to rename it as OpenMASK [1] (Open Modular Animation and Simulation Kit).

The issue addressed in our work deals with the specification of a general formalism for behaviour modelling based on psychological studies and compatible with real-time constraints. In the next section, a complete programming environment dedicated to behavioural animations is presented. Then, section 3 is devoted to the presentation of an example: a virtual museum. Finally, we present work in progress and conclude.

2 A programming environment for behavioural animation

A behavioural animation is compounded of a large set of dynamic entities evolving and interacting in a complex environment. To be able to model such applications, we need to implement different models: environment models, mechanical models, motion control models, behavioural models, sensor models, geometric models and scenarios.

2.1 OpenMASK: Modular Animation and Simulation Kit

One of the objectives of openMask (modular animation and simulation kit, and multi-threaded animation and simulation kernel) is to provide a common run-time and conception framework for the creation of virtual environments.

In openMask, the building block of a virtual environment is the simulated object, which can be viewed as the container of all the different models used for the simulation of the evolution of the environment. More formally, a simulated object can be seen as a container of a computation function, $(E_e, S, O, \theta_{n+1}) = f(E_r, I, CP, \theta_n)$, where O is a set of outputs, S a set of fired signals, E_r a set of received events, E_e a set of emitted events, I a set of inputs, CP a set of control parameters and θ_n the state of the object at simulation step n. Outputs and control parameters are object representing the public state of an object, its position for example. Inputs are connected (statically or dynamically) to the outputs and control parameters of other objects in order to establish data flow connection between objects. Signals are events sent in the environment without receivers for asynchronous communication between objects, and are received by any registered object. Therefore, objects can communicate using data-flow mechanisms and asynchronous ones at the same time, enabling implementation of different models using the most appropriate communication style for each model.

Object activation (the moment when f is called) can be done by the animation and simulation kernel either on a regular basis (giving the object a simulation frequency), only when the object receives events (if the object has no frequency), or on a regular basis but also when events are received. Therefore, the same framework can be used in a natural way for reactive objects as well as for active (living) objects.

Building an openMask application consists in composing simulated objects, by organizing them all in a simulation tree (which can be flat) and giving the configuration parameters necessary to obtain the required results. The simulation tree is used to structure relations between objects, thus creating son-father and brother relations in the simulation. These relations are for example used to establish dynamic connections between the components of a simulated entity whose behavior has been implemented using different

[1]OpenMASK can be downloaded at the following address: http://www.irisa.fr/siames/OpenMASK

simulated object (for example, one object would be responsible for perception, one for decision, and one for simulation of the (bio)-mechanical results of performed actions).

Once the application has been built, the last step before achieving any result is choosing the run-time kernel used. Indeed, the conceptual framework presented here is a solid foundation for many different run-time constraints, as it has been designed to enable off-line as well as on-line simulation, and multi-threaded execution. Indeed, the component responsible for the rendering of a simulation is a simulated object, which can therefore be included or excluded from the application. At the time of this writing, openMask has a number of different run-time kernels, for Irix and linux. These run-time kernels include a real-time kernel (only for Irix), a multi-threaded kernel (for SMP machines), a distributed kernel (using PVM) and a kernel for PC-clusters running linux[2] [26].

Discussion of the performance and tradeoffs made by each of these kernels is beyond the scope of this article, but it should be noted that the distributed and the usual kernels have been extensively used for a number of projects, ranging from behavioral animation to distributed virtual reality.

2.2 HPTS: A Model for Behavioural Animation

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 skills and certainly not to support general intelligence. As direction and inspiration towards the development of such a theory, Newell [30] 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 [24] 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 functions are different in nature: some are purely reactive, while others require more time. Executions 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, locally based on problem spaces constructed on a moment-to-moment basis.

Different approaches have been studied for the decision part of behavioural models in animation: sensor-effector or neural networks, behaviour rules, finite automaton approach. As human behaviour is very complex, none of the preceding models could be applied. More recently, a second generation of behavioural models has been developed to describe the human behaviour in specific tasks. The common features of these new models are: reactivity, parallelism and different abstract levels of behaviours. In [6], authors describe a multi-agent development environment named DASEDIS and use it to describe the behaviour of a car driver. In [31], 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. In the Motivate product proposed by Motion Factory for the Game Design Market, Hierarchical Finite State Machines have also been introduced, and actions associated with the states and transitions can be described by using an object-based programming language, named Piccolo [22]. 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

[2]Open-source versions will exclude the PC-cluster version because its development is partially funded by a private company: IWI

state of itself and of other agents with which he is currently interacting [18]. To do this, 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, 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 Parameterized Action Representation (PAR) have been introduced 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 [3]. In all these systems, the action is directly associated with each node, which doesn't allow the management of concurrency.

2.2.1 HPTS

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 preempting sub-behaviours) and time and frequency handling for execution of sub-behaviours (This provides the ability to model reaction times in perception activities). HPTS [16] which stands for Hierarchical Parallel Transition Systems, is a formalism proposed to specify the decisional part of an autonomous character. It offers a set of programming paradigms, which allow to address hierarchical concurrent behaviours. It consists in a reactive and cognitive model, 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, a set of control parameters and an internal state. The synchronization of the agent execution is operated using state machines. To allow an agent to manage concurrent behaviours, sub-agents are organized through sub-state machines. In the following, agents will be assimilated to their corresponding state machine, as there is not any constraint imposed to the programmer, concerning the body part of the agent. 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. The behavioural description language is not described in details. For a complete description of the model refer to [10], and for the management of resources and priority levels, refer to [23]. 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. Resources used by a state are defined by using the instruction *USE resource list.* 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. A transition is defined by

```
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.

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.

Afterwards, C++ code for our simulation platform GASP [11] 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 with an increase of only ten percent of the execution time.

2.2.2 Behaviour Coordination

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, like 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 soon too complex, and has motivated the recent integration of resources and priority levels into HPTS [23].

In the contrary of some previous approaches, it is not necessary to specify exhaustively all behaviours that are mutually exclusive; this is done implicitely just by attaching resources to nodes, preference values to transitions and a priority function to each state machine, and by using a scheduling algorithm at run-time. Each state of a state machine can use a set of resources which can be considered as semaphores. Thus, resources are used for mutual exclusion between behaviours. Entering a node implies that resources are marked as taken and exiting it implies that thoses resources are released. 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 priority and preference. The degree of preference is a coefficient associated to a transition and corresponds to the proclivity of the state machine to use this transition when the associated transition is true. It allows to describe a behaviour with different ways of realization; possible adaptation depending on resources availability or need can be described. A priority function is associated to each state machine. This function returns a value representing the importance of a behaviour in a given context. It can be used to control a behaviour during the running phase. As it is user defined, it can be correlated with the internal

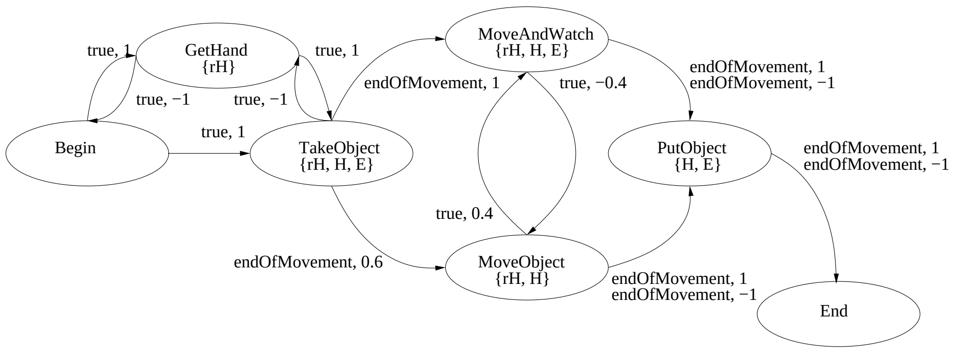

Figure 2: Moving object behaviour.

state of the character (psychological parameters, intentions) or with external stimuli. Combining those two notions allows to create a scheduling method which globally favours the realization of most important behaviours while automatically adapting the execution of running ones. This scheduling system allows to describe independently all behaviours with their different possibilities of adaptation. During running phase, the adaptation of all other 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.

The example of fig. 3 consists in drinking a coffe and smoking a cigarette while reading sheets of paper. Those behaviours have been described independantly and the overall behaviour as been automatically generated by the scheduler at run time. The fig. 2 shows an example of a behaviour consisting in moving an object, used to move the sheets of paper. This behaviour uses 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. Then a state which only use resource Hr/Hl corresponds to a behaviour of freeing a hand resource. 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. Video sequences related to behaviour coordination can be found at http://www.irisa.fr/prive/donikian/resources.

2.3 Informed Environments

Using 3D modelling systems allow the generation of realistic geometrical models in which walk through is possible in real time. As this modelling operation is still a long, complex and costly task, a lot of work has been done to partially automate the rebuilding process. All these techniques are very useful for the visual realism of virtual urban environments but they are not sufficient due to the lack of life of these digital mock-ups. Walking through these virtual city models do not provide a real life feeling as they are uninhabited. In order to populate these virtual environments, it is necessary to specify the behaviour of autonomous characters with the ability to perceive their surrounding space and act on it. An autonomous actor whose main action is obstacle avoidance in an unstructured environment does not need other knowledge than the geometrical one. In order to simulate more sophisticated behaviours, other kind of information must be manipulated. The simplest behaviour, for a pedestrian walking in a street, consists in minimizing possible interactions, which mean avoiding static and dynamic obstacles. But, even in this simple walking activity, one needs to know the nature of objects he will interact with. For example, a public phone is considered as an obstacle to avoid for most people, but some of them will be interested by its functionality and will use

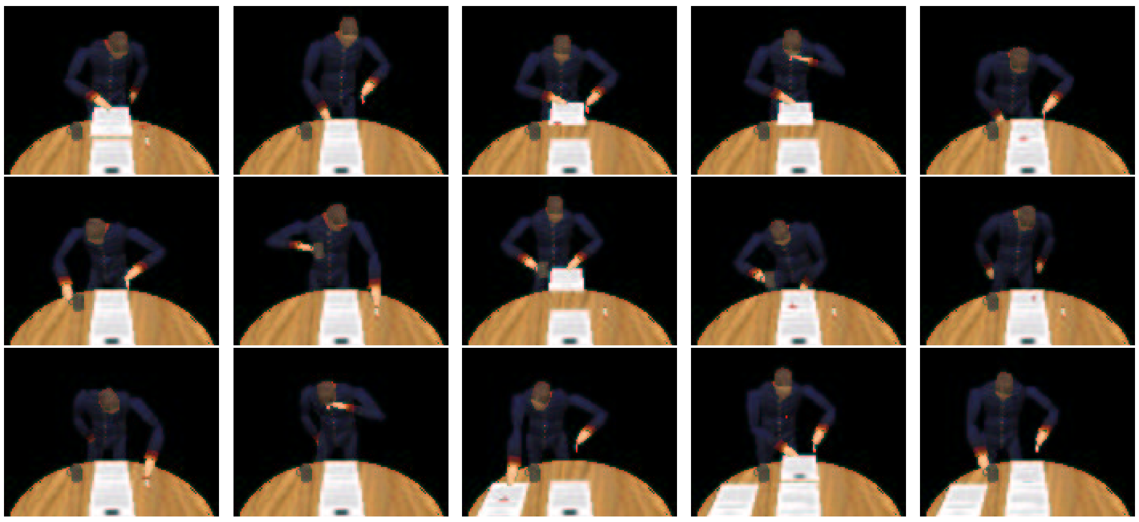

Figure 3: Behavioural Coordination Example.

it. When crossing a street, one activity consists in reading the signals, which mean that it is necessary to associate semantic information to geometric objects in the scene, and to update it during the simulation. N. Farenc [17] has suggested using an informed environment, dedicated to urban life simulation, which is based on a hierarchical breakdown of an urban scene into environmental entities providing geometrical information as well as semantic notions. S. Raupp Musse [29] has used this informed environment to animate human crowds using a hierarchical control: a virtual human agent belongs to a group that belongs to a crowd, and an agent applies the general behaviours defined at the group level. The knowledge on the virtual environment used by the crowd is composed of a set of obstacles (bounding box information of each obstacle to be avoided), a list of interest points (locations that the crowd should pass through and their associated regions) and a list of action points (regions where agents can perform actions).

In the field of behavioural psychology, there have been a few studies on visual perception, mainly based on Gibson's theory of affordances [19]. The theory of affordances is based on what an object of the environment *affords* to an animal. Gibson claims that the direct perception of these affordances is possible. Affordances are relations between space, time and action, which work for the organism. What the world is to the organism depends on what the organism is doing and might do next. For computer scientists, Gibson theory is really attractive because it assigns to each object some behavioural semantic, i.e. what the human being is likely to do with a given object. Associating symbolical information to objects, Widyanto experienced the Gibson's theory of "affordances" [36], while M. Kallmann [20] introduces smart objects, in which all interesting features of objects are defined during the modelling phase.

In accordance with Gibson's ecological theory, components of the virtual urban environment should be informed. To produce more realistic behaviours, we [34] have specified a city model *affordant* to autonomous actors. In the field of psycho-sociology, M. Relieu [32] has defined the notion of positive and negative affordances for a pedestrian. A positive affordance specifies the fact that the extrapolated trajectory of the corresponding pedestrian is not supposed to intersect the planned trajectory of the other one, while a negative affordance points out that they may collide. M. Relieu says also that a mobile entity uses the urban discrimination [3] to focus his attention, to select pertinent information for his actions inside the current region, while he maintains a secondary task to observe what is happening in regions close to its circula-

[3] A street is composed of connected lanes devoted to distinct mobile entities such as cars and pedestrians

tion area. Thanks to this knowledge, autonomous virtual actors can behave like pedestrians or car drivers in a complex city environment. A city modeller, named VUEMS (Virtual Urban Environment Modelling System), has been designed, using this model of urban environment, and enables complex urban environments for behavioural animation, and their 3D geometric representation, to be automatically produced. 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. The behavioural model of pedestrians, that has been developped, includes social and driving rules of interaction (minimize the interaction and choose in priority the left side to overtake), as explained in [35].

2.4 Scenario Authoring

The scenario component of a behavioural simulation carries out the responsibility for orchestrating the activities of semi-autonomous agents that populate the virtual environment. In common practice, these agents are programmed as independent entities that perceive the surrounding environment and under normal circumstances behave as autonomous agents. However, in most experiments and training runs [21], we want to create a predictable experience. This is accomplished through direction of objects behaviours. To facilitate coordination of activities, objects have to be built with interfaces through which they can receive instructions to modify their behaviours. The scenario manager controls the evolution of the simulation by strategically placing objects in the environment and guiding the behaviours of objects to create desired situations. We have specified a scenario language called SLUHRG (Scenario Language Using HPTS Runing on GASP) [9, 13], which allows to describe scenarios in a hierarchical manner and to schedule them at simulation time. This language has its own syntax and is compounded of a set of specific instructions. The language contains usual instructions such as *if, switch, repeat until, random choice, wait* and which executed inside a timestep. It contains also more specific instructions (*waitfor, eachtime, aslongas, every*) that will spend more than one timestep to be finished and that can run in parallel during the execution of the scenario they belong to. All those instructions can be composed in a hierarchical manner. To manage actors, the language offers also specific instructions to specify the interface of an actor, to reserve and free actors. Using priorities, a scenario can steal an actor to another one, which will be informed of it by a message. Concerning messages, each scenario can subscribe to messages that are of interest for itself.

As the language is built upon C++, it is always possible to include C++ code everywhere inside a scenario by using four specific instructions:

- include . . ., is used to insert C++ include commands;
- declaration . . ., is used to declare C++ variables that will be used inside the scenario (Thus, we did not have to manage our own datatypes);
- implementation . . ., is used to insert c++ code inside a scenario;
- destructor . . ., is used to cleanly terminate the c++ part of the scenario.

Using C++ code inside the scenario provides the ability to describe a large variety of scenaristic elements without being limited to a specific description language. Nevertheless, this fact restricts the use of scenario language to programmers.

A scenario can be decomposed into sub-scenarios and each scenario is corresponding to a set of tasks or actions, ordered on a global temporal referential. Tasks in a scenario can modify characteristics of actors, create them or ask an actor to perform a specific action. Internal representation is based on the use of Allen's logic and of rewriting rules to produce Hierarchical Parallel Transition Systems. A scenario can start at a predefined time given by the author but can also be started when a situation occurs (conditional scenario). Some of those scheduling informations are stored in a dynamic execution graph. A scheduler uses it to start or terminate scenarios. To detect situations, triggers and sensing functions are managed by the simulation observation.

3 Application to a Virtual Museum

The virtual Museum is an application which has been developed for a museum in Paris (Cité des Sciences et de l'Industrie), and has been inaugurated in June 2001 as part of the new permanent exhibition on images of this museum. This application has the advantage to integrate a lot of recent research development of the team (cf figure 5). It consists in a museum visited by a group of autonomous characters and it is also inhabited by a cloud of suribirds (birds with the behaviour of surikats). The humanoid motion control integrates three kinds of technics: a bio-mechanical model for locomotion, inverse kinematics algorithm for grasping or climbing and motion capture for specific gestures. Different motions can be applied simultaneously to separate parts of the human body. As for Urban Environment, an informed environment has been specified to modelize the structure of the building and the interior architecture (cf. map of the museum in figure 4). Doors and art works are described as specific objects which can be perceived and manipulated by the agents. The map of the environment contains information about the topology, the agents and the objects populating the museum, in order to handle perception and path finding.

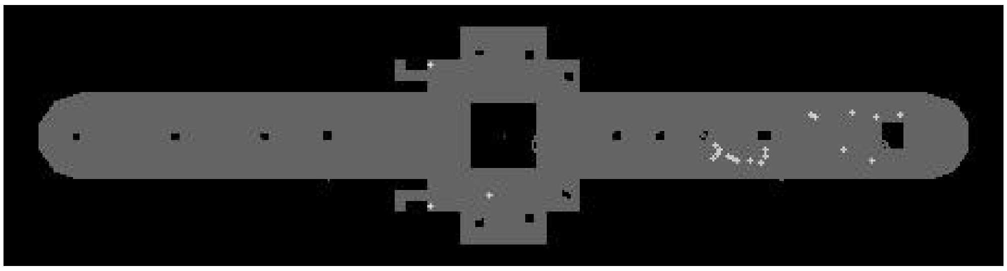

Figure 4: The Map of the Museum.

Thanks to an initial scenario, a set of specific characters has been 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 child who have conflicting desires,
- a guide who present the exhibition to a group of visitors;
- a woman who is in charge of restoring damaged paintings.

The purpose of this application is to propose to the real visitor to observe the life evolving in a virtual world and to see how the modification of behavioural parameters of one or more autonomous characters can affect this virtual life[4]. For example, the mother may have to choose between following the explanation given by the guide and looking at her child. If the child obey to her mother, there is no problem for the

[4]This can easily be done due to the preemption mechanism, and the dynamicity and parameterization of state machines.

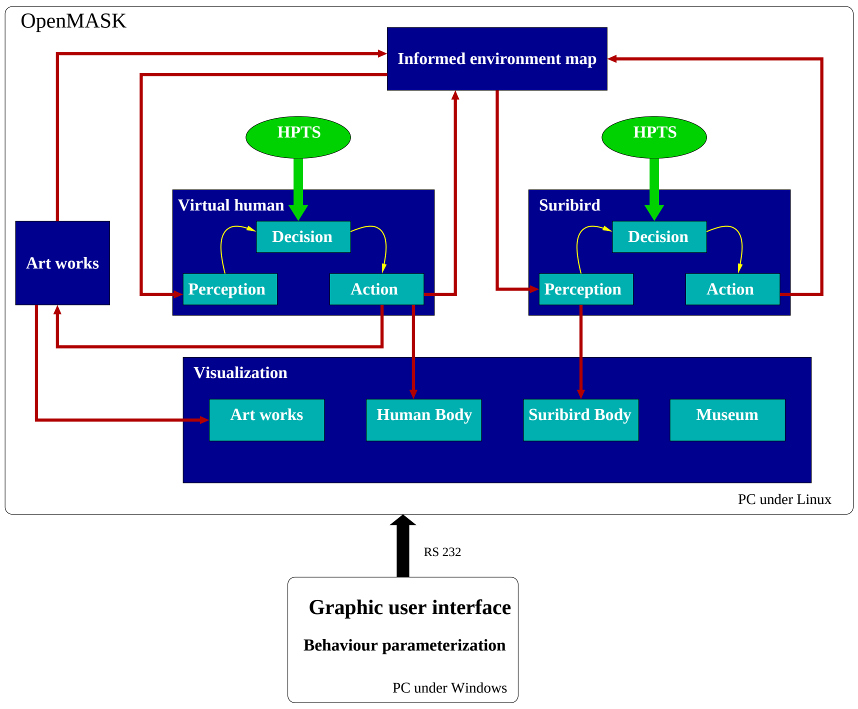

Figure 5: The Virtual Museum Application.

mother to follow the visit. In the contrary, her behaviour will depend on her interest for the visit and on the regard she wanted to have on the activity of her child. Concerning the attendant, it is possible to choose the level of interest (no interest, weak, medium, high, very high) that he will have to survey the thief, the photographer and the child. It is also possible to choose its speed and its weariness. Behavioural parameters can also be modified for the other main characters. It is also possible to specify a global level of disorder inside the museum: this will for example modify the response of visitors to damages caused by the thief and the photographer.

Figure 6 shows some snapshots taken from the application. On the top left image, the thief is in action, while on the top right image the guide is moving inside the museum. On the bottom left image, we can see a group of visitors listening to the explanation given by the guide (including the mother and her child). The bottom right image is an overview of one part of the museum, including around twenty autonomous characters. As this application should run all the day, seven days a week, when the attendant ask the thief or the photographer to leave the museum, a women is asked to restore paintings and sculptures are put back on their pedestal. After a moment, the thief or the photographer are asked to come back into the museum by the entrance. Video sequences of the virtual museum can be found at http://www.irisa.fr/prive/donikian.

4 Conclusion

Our main objective is real-time simulations of several entities evolving in realistic informed environments. Many studies have been performed by psychologists to analyse the human behaviour. The behavioural 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 behavioural models are presently restricted to the animation of one model in a specific environment. The use of Hierarchical Parallel Transition Systems allows us to take

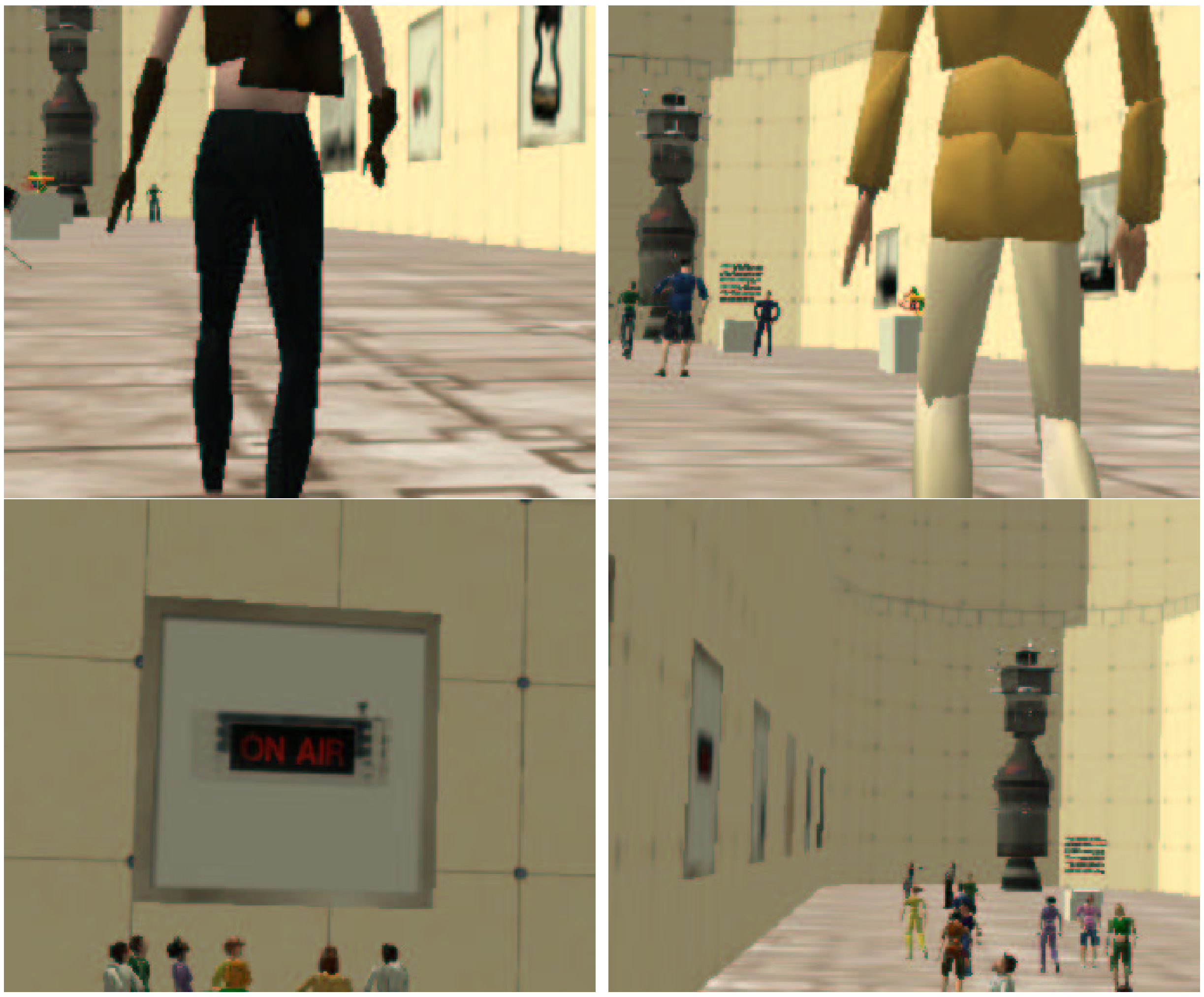

Figure 6: Images of the Virtual Museum Application.

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. Another important point is that our behavioural model has been built to generate dynamic entities which are both autonomous and controllable by a scenario, allowing us to use the same model in different contexts and moreover with different levels of control.

Concerning behaviour coordination, our scheduler is currently only 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 would be to connect this work to a higher level of reasoning, in order to determine automatically which behaviour should be activated or inhibited.

Despite the benefits of the language approach described earlier, the description of behaviour remains quite difficult to 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 into an interactive simulation. Work in progress concerns scenarios in natural languages to be able to offer an authoring tool to scenarists of interactive drama. This requires to take into account semantics of natural language (action and motion verbs and spatio-temporal relation) and the theory of drama as expressed by structuralists. The text in natural language will be analysed and translated in SLUHRG, our scenario language.

References

1. B. Arnaldi, R. Cozot, S. Donikian, and M. Parent. Simulation models for the french praxitele project. In *Transportation Research Board Annual Meeting*, Washington DC, USA, Jan. 1996.
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*, 1997; **1195**:43-57.
3. R. Bindiganavale, W. Schuler, J. Allbeck, N. Badler, A. Joshi, and M. Palmer. Dynamically altering agent behaviors using natural language instructions. In *International Conference on Autonomous Agents*, C. Sierra, M. Gini, and J. Rosenschein (ed.); ACM Press: Barcelona, Spain, June 2000; pp 293-300
4. B. Blumberg and T. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. In *Siggraph*; ACM: Los Angeles, California, U.S.A., Aug. 1995; pp 47-54.
5. D. Brogan, R. Metoyer, and J. Hodgins. Dynamically simulated characters in virtual environments. In *IEEE Computer Graphics and Applications* 1998; pp 58-69.
6. B. Burmeister, J. Doormann, and G. Matylis. Agent-oriented traffic simulation. In *Transactions of the Society for Computer Simulation International* June 1997; 14(2).
7. E. Cerezo, A. Pina, and F. Seron. Motion and behaviour modelling: state of art and new trends. In *The Visual Computer* 1999; **15**:124-146.
8. R. Cozot and B. Arnaldi. A language for multibody systems modelling. In *European Simulation Symposium*; Erlangen-Nuremberg, Oct. 1995.
9. F. Devillers. *Langage de scénario pour des acteurs semi-autonomes*. PhD thesis, Université de Rennes I, Sept. 2001.
10. S. Donikian. HPTS: a behaviour modelling language for autonomous agents. In *Fifth International Conference on Autonomous Agents*; ACM Press: Montreal, Canada, May 2001.
11. S. Donikian, A. Chauffaut, R. Kulpa, and T. Duval. Gasp: from modular programming to distributed execution. In *Computer Animation'98*; IEEE: Philadelphie, USA, June 1998; pp 79-87.
12. S. Donikian and R. Cozot. General animation and simulation platform. In *Computer Animation and Simulation'95*, D. Terzopoulos and D. Thalmann (ed.); Springer-Verlag, 1995; pp 197-209.
13. S. Donikian, F. Devillers, and G. Moreau. The kernel of a scenario language for animation and simulation. In *Eurographics Workshop on Animation and Simulation*; Springer Verlag: Milano, Italia, Sept. 1999.
14. S. Donikian, S. Espie, M. Parent, and G. Rousseau. Simulation studies on the impact of acc. In *5th World Congress on Intelligent Transport Systems*; Séoul, Corée du sud, Oct. 1998.
15. S. Donikian, G. Moreau, and G. Thomas. Multimodal driving simulation in realistic urban environments. In *Progress in System and Robot Analysis and Control Design*, S. Tzafestas and G. Schmidt (ed.); Lecture Notes in Control and Information Sciences (LNCIS 243), 1999; pp 321-332.
16. S. Donikian and E. Rutten. Reactivity, concurrency, data-flow and hierarchical preemption for behavioural animation. In *Programming Paradigms in Graphics'95*, E. B. R.C. Veltkamp (ed.), Eurographics Collection; Springer-Verlag, 1995.
17. N. Farenc, R. Boulic, and D. Thalmann. An informed environment dedicated to the simulation of virtual humans in urban context. In *EUROGRAPHICS'99*, P. Brunet and R. Scopigno (ed.); Blackwell, Sept. 1999; pp 309-318.
18. J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. In *SIGGRAPH'99*; Los Angeles, Aug. 1999; pp 29-38.
19. J. Gibson. *The ecological approach to visual perception*. NJ: Lawrence Erlbaum Associates, Inc, Hillsdale, 1986.
20. M. Kallmann and D. Thalmann. Modeling objects for interaction tasks. In *Eurographics Workshop on Animation and Simulation*; Springer-Verlag: Lisbon, Portugal, Sept. 1998.
21. J. Kearney, P. Willemsen, S. Donikian, and F. Devillers. Scenario languages for driving simulations. In *DSC'99*; Paris, France, July 1999.
22. Y. Koga, G. Annesley, C. Becker, M. Svihura, and D. Zhu. On intelligent digital actors. In *IMAGINA'98*; 1998.
23. F. Lamarche and S. Donikian. The orchestration of behaviours using resources and priority levels. In *Computer Animation and Simulation 2001*, M. Cani, N. Magnenat-Thalmann, and D. Thalmann (ed.); Springer-Verlag: Manchester, UK, Sept. 2001; pp 171-182.
24. R. G. Lord and P. E. Levy. Moving from cognition to action : A control theory perspective. In *Applied Psychology : an international review* 1994; 43 **3**:335-398.

25. P. Maes, T. Darrell, B. Blumberg, and A. Pentland. The alive system: Full-body interaction with autonomous agents. In *Computer Animation'95*; IEEE: Geneva, Switzerland, Apr. 1995; pp 11-18.
26. D. Margery. *Environnement logiciel temps-réel distribué pour la simulation sur réseau de PC*. PhD thesis, Université de Rennes 1, 2001.
27. 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.
28. F. Multon, L. France, M. Cani-Gascuel, and G. Debunne. Computer animation of human walking : a survey. In *Journal of Visualization and Computer Animation* 1999; **10**:39-54.
29. S. R. Musse. *Human Crowd Modelling with Various Levels of Behaviour Control*. PhD thesis, EPFL, Lausanne, Suisse, Jan. 2000.
30. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
31. H. Noser and D. Thalmann. Sensor based synthetic actors in a tennis game simulation. In *Computer Graphics International'97*; IEEE Computer Society Press: Hasselt, Belgium, June 1997; pp 189-198.
32. M. Relieu and L. Quéré. *Les Risques urbains : Acteurs, systèmes de prévention*, chapter Mobilité, perception et sécurité dans les espaces publics urbains. anthropos: Collection VILLES, 1998.
33. D. Thalmann and H. Noser. Towards autonomous, perceptive, and intelligent virtual actors. In *Artificial Intelligence Today*, volume 1600 of *Lecture Notes in Artificial Intelligence*; Springer, 1999; pp 457-472.
34. G. Thomas and S. Donikian. Modelling virtual cities dedicated to behavioural animation. In *EUROGRAPHICS'2000*, M. Gross and F. Hopgood (ed.); Blackwell Publishers: Interlaken, Switzerland, Aug. 2000; vol. 19:3.
35. G. Thomas and S. Donikian. Virtual humans animation in informed urban environments. In *Computer Animation*; IEEE Computer Society Press: Philadelphia, PA, USA, May 2000; pp 129-136.
36. T. Widyanto, A. Marriott, and M. West. Applying a visual perception system to a behavioral animation system. In *Eurographics Workshop on Animation and Simulation*; Vienna, Austria, 1991; pp 89-98.