

# A programming environment for behavioral animation

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

## Abstract

Behavioral models offer the ability to simulate autonomous agents like organisms and living beings. Psychological studies have showed that the human behavior 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 behavioral animation which is composed of a general animation and simulation platform, a behavioral modelling language, a scenario authoring tool and a city modeler which enables to produce complex urban informed environments.

## 1 Introduction

The goal of the behavioral model is to simulate autonomous entities like organisms and living beings. A behavioral entity has the following capabilities: perception of its environment, decision, action and communication. Most behavioral 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 behavior modeling based on psychological studies and compatible with real-time constraints. In the next section we focus on the nature of data which are requested to simulate behavioral entities, before presenting in the third section a complete programming environment which enables us to specify and execute behavioral animations. Then the last section is devoted to the presentation of an example: a virtual museum. Finally, we present work in progress and conclude.

## 2 Behavioral animation requirements

Behavioral animation consists of a high level closed control loop [23, 8, 6, 31], which offers the ability to simulate autonomous entities. Such actors are able to perceive their environment, to communicate with others [5] 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. Realistic behaviors 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 that must be extracted or interpreted from environment depends on the abstraction level of the reasoning performed by autonomous actors. 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 behaviors, other kind of information must be manipulated. The simplest behavior, 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 it. For the crossing of 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. In the realm of behavioral psychology, there have been a few studies on visual perception, mainly based on Gibson's theory of affordances [18]. 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 behavioral 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" [34], while M. Kallmann [19] introduces smart objects, in which all interesting features of objects are defined during the modelling phase.

Information needed to describe the behavior 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 [22] introduces several paradigms about the way the brain works and controls the remainder of the human body. He explains that human behavior 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 behavioral models in animation: sensor-effector or neural networks, behavior rules, finite automaton approach. As human behavior is very complex, none of the preceding models could be applied. More recently, a second generation of behavioral models has been developed to describe the human behavior in specific tasks. The common characteristics of these new models are: reactivity, parallelism and different abstract levels of behaviors. In [7], authors describe a multi-agent development environment named DASEDIS and use it to describe the behavior of a car driver. In [28], a tennis game application is shown, including the behavior of players and referees. A stack of automata is used to describe the behavior of each actor. 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 [21]. As humans are deliberative agents, purely reactive systems are not sufficient to describe their behavior. It is necessary to integrate both cognitive and reactive aspects of behavior. 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 [17]. 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 behavior 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[4]. In all these systems, the action is directly associated with each node, which doesn't allow the management of concurrency. HCSM [1] and HPTS [15], both based on a hierarchy of concurrent state machines, offer a set of programming paradigms, which permit to address hierarchical concurrent behaviors. According to Newell, our goal is to build a model which will allow some adaptative and flexible behavior 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* behavioral model are: reactivity (which encompasses sporadic or asynchronous events and exceptions), modularity in the behavior description (which allows parallelism and concurrency of sub-behaviors), data-flow (for the specification of the communication between different modules), hierarchical structuring of the behavior (which means the possibility of pre-empting sub-behaviors) and time and frequency handling for execution of sub-behaviors (This provides the ability to model reaction times in perception activities).

### **3 A programming environment for behavioral animation**

#### **3.1 Introduction**

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 [11]. 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, behavior, artificial vision. The mechanical aspect is modelled with DREAM, our rigid and deformable bodies modelling system which generates numerical C++ simulation code for GASP [9]. For human motion, a bio-mechanical approach has been preferred [25]. HPTS concerns the modelling of the behavioral part of an actor [24] and is also used as an intermediate level for scenario authoring [12]. 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 behavioral 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 [2, 13, 20, 14]. In this section, we will first present the main characteristics of the simulation platform and then give an outline of the behavioral part of this environment.

#### **3.2 GASP: a modular programming environment**

A behavioral animation is composed 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, behavioral models, sensor models, geometric models and scenarios. The main objective of GASP is to give the ability to simulate different entities composed

themselves of different modules in different hardware configurations, without any change for the animation modules. When someone specifies a module, he does not have to make any hypothesis on the network location of other modules he must interact with. Nevertheless, he must be able to name them, and for that, modules are structured in a simulation tree. Each module of a simulation is a specialisation of a class named *PsSimulObject*. The *PsSimulObject* class can be viewed as the container of a computation function  $Y = F(X, CP)$ , where  $X$  is a set of inputs,  $Y$  a set of outputs and  $CP$  a set of Control Parameters.  $X$  and  $Y$  determine the data-flow from and to other objects. Each object has its own frequency and is activated periodically to calculate its new state. At each simulation step, the new input values are used to compute the outputs. This requires to connect each input of the object to an output of another object. This data dependency can be static or dynamic, as we cannot know at the beginning of a simulation, which objects might interact later. To take into account dynamic data dependencies, the number of inputs of an object can change during the simulation, unlike the one of outputs. A configuration file is used for each simulation to define which dynamic objects are used and on which hardware. As several processes can be used, this file describes first which processors are used, and then each process is named and located on a processor. As the modules of an entity can be located in separate processes, the location of each module is specified in the configuration file. During the simulation, inputs of an object must be supplied by values of outputs. Rather than to define specifically how each reference object must send the new values of its outputs to interested reference object, it has been preferred an automatic mechanism which is based on a client/server mechanism. Each time the inputs of objects of a process require the value of the outputs of another one reference object, an object which contains only the outputs and control parameters of the reference object is created for this process: we call it a mirror object. The continuous communication between two agents can be managed by a two steps mechanism: firstly, the reference object communicates to its mirror the new value of its outputs and control parameters; secondly, the object interested by outputs or control parameters of another object can contact the embodiment of this object in its own process. As each reference object runs at its own internal frequency, the data-flow communication channel must include all the mechanisms to adapt to the local frequency of the producer and of consumers (over-sampling, sub-sampling, interpolation and extrapolation). With the intention of minimising communications between processes, the frequency of the communication between a reference object and each of its mirrors is computed especially for each mirror. GASP our General Animation and Simulation Platform, 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 communication between modules and their synchronization.

### **3.3 HPTS: A Model for Behavioral Animation**

Hierarchical Parallel Transition Systems, or HPTS [15] is a formalism proposed to specify the decisional part of an autonomous entity. HPTS 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 behaviors, 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. An activity parameter is associated to each state machine which corresponds to the current status of this machine (active, idle or wait). The status of a state machine is described by an automaton, in which transitions between the three status 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()). Activity of a state evolves during the simulation, and

this is determined by an activity function. 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 handles also hierarchical preemption, by the fact that one argument of the function is a set of Control Parameters which allows to deal with internal or external events. Input and output parameters are continuous signals of standard type (e.g. integer, real, boolean for example). Outputs are used to return actions proposed by the state-machine. Local variables are some variables of standard type, which can either retain their values between activations or be reinitialized on each reactivation (*started* status). Control parameters allow to modulate the behavior of an entity, depending on external or internal decision. 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 behaviors proposed by different sub-state machines, this function has to make a choice and to deliver a unified behavior as output of the state machine.

**Behavior description language.** Our model of a behavioral 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 the behavior 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 fully implements the HPTS formalism. The goal is the (object-oriented) design of behavioral entities, at least for their reactive part, and the connection with tasks that require more computation time. 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.

```

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

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

```

Figure 1: Syntax of the language.

Accepted types of data (*type* rule) are either float, integer, boolean and 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 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 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. 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 (usefull 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-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. 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 a transition, which means that no more transition can be fired during the same timestep in this state machine. This ability to fire several following transitions at the same date is useful to express the semantics of a scenario language as it permits to use in the same language temporal and procedural instructions [12]. The condition of a transition is a boolean expression and can be composed of: test of termination of one or all sub-state machines; wait for the reception of a specific event; logical combination of boolean conditions which can use local variables and boolean functions; an expression which become true when the time spent in the initial state of the transition is higher than the maximal allowed duration of this state. The *write-expression* could contain instructions to create a new instance of a state machine<sup>1</sup>, to suspend, resume or kill a state machine and to send an event to all state machines (broadcast mode). 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.

**Code generation.** Afterwards, C++ code for our simulation platform GASP [10] 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 (display of current state, transitions fired, value of the variables). The advantage of using the object model is that the behavior 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 behavior specification phase as it allows to modify state-machines during the execution phase with an increase of only ten percent on the execution time.

### 3.4 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, we have to specify the behavior of dynamic entities such as pedestrians, car drivers or public transportation systems. In accordance with Gibson's ecological theory,

---

<sup>1</sup>This new state machine will be added to the list of childs of the current state machine.

components of the virtual urban environment should be informed. N. Farenc[16] 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 [26] has used this informed environment to animate human crowds by using a hierarchical control: a virtual human agent belongs to a group that belongs to a crowd, and an agent applies the general behaviors 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). We [32] have specified a model of urban environments using structures and information suitable for behavioral animations. Thanks to this knowledge, autonomous virtual actors can behave like pedestrians or car drivers in a complex city environment. A city modeler, named VUEMS (Virtual Urban Environment Modelling System), has been designed, using this model of urban environment, and enables complex urban environments for behavioral 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 behavioral module can have access to the information on these visible objects. The behavioral model of pedestrians, that has been developed, includes social and driving rules of interaction (minimize the interaction and choose in priority the left side to overtake), as explained in [33].

### **3.5 Scenario Authoring**

In the theatre [29], action is the basic dynamic unit of a scene. An action is characterized by one or a few number of dynamic events. In movie-making, the basic unit is the shot, which corresponds to a camera filming without interruption. During the shot the camera can be either fixed or moving. In the theater, a scene is a part of a play characterized by spatial, temporal and narrative continuity. In movie-making, since a scene is composed of shots, it inherits the spatial discontinuity which may affect them. When the action does not all occur in a fixed place (two people in different places are taken alternately to show what they are doing at the same time), it is considered to be one scene because the narrative continuity prevails over spatial continuity. A sequence is a group of scenes linked together for reasons established by the director. In a game, the field of view is generally made by a subjective camera attached to the supposed point of view of the game player in the scene. However, actions cannot be planned explicitly by the game designer, and what is usually specified is the game reactivity depending on the player's actions in a specific phase of the game. The scenario component of a behavioral 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 [20], we want to create a predictable experience. This is accomplished through direction of objects behaviors. To facilitate coordination of activities, objects have to be built with interfaces through which they can receive instructions to modify their behaviors. Scenario processes control the evolution of the simulation by strategically placing objects in the environment and guiding the behaviors of objects to create desired situations. We have specified a scenario language [12], which permits to describe scenarios in a hierarchical manner and to schedule them at the simulation time. Tasks in a scenario can modify characteristics of actors, create them or ask an actor to perform a specific action.



behaviors which are using the same resource are mutually exclusive, but in the real life, we are able to combine them in a much microscopic way, as for example smoking, drinking and reading at the same time. That's why we are currently integrating resources and priority management inside HPTS. We didn't have yet a visual application integrating these new features as it requires to manage 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. In the contrary of some previous approach [30], it is not necessary to specify exhaustively all behaviors 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 behavior remains quite difficult to people who are not computer scientists. Therefore we are working on a higher level specification language, in order to allow behavioral specialists to specify and test their models into an interactive simulation. Another work in progress concerns a higher level scenario language, including 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 our internal scenario language, which uses HPTS in the TIMEGATE mode.

## 6 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 behavior. 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. The use of Hierarchical Parallel Transition Systems allows us to take into account several programming paradigms important to describe *realistic* behaviors. Because of the integration of our behavioral 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 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.

## 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] 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.
- [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] 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.
- [5] 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.
- [6] D. Brogan, R. Metoyer, and J. Hodgins. Dynamically simulated characters in virtual environments. *IEEE Computer Graphics and Applications*, pages 58–69, 1998.
- [7] B. Burmeister, J. Doormann, and G. Matylis. Agent-oriented traffic simulation. *Transactions of the Society for Computer Simulation International*, 14(2), June 1997.
- [8] E. Cerezo, A. Pina, and F. Seron. Motion and behaviour modelling: state of art and new trends. *The Visual Computer*, 15:124–146, 1999.

- [9] R. Cozot and B. Arnaldi. A language for multibody systems modelling. In *European Simulation Symposium*, Erlangen-Nuremberg, Oct. 1995.
- [10] 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.
- [11] S. Donikian and R. Cozot. General animation and simulation platform. In D. Terzopoulos and D. Thalmann, editors, *Computer Animation and Simulation'95*, pages 197–209. Springer-Verlag, 1995.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] 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.
- [16] N. Farenc, R. Boulic, and D. Thalmann. An informed environment dedicated to the simulation of virtual humans in urban context. In P. Brunet and R. Scopigno, editors, *EUROGRAPHICS'99*, pages 309–318. Blackwell, Sept. 1999.
- [17] 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.
- [18] J. Gibson. *The ecological approach to visual perception*. NJ: Lawrence Erlbaum Associates, Inc, Hillsdale, 1986.
- [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] J. Kearney, P. Willemsen, S. Donikian, and F. Devillers. Scenario languages for driving simulations. In *DSC'99*, Paris, France, July 1999.
- [21] Y. Koga, G. Annesley, C. Becker, M. Svihura, and D. Zhu. On intelligent digital actors. In *IMAGINA'98*, 1998.
- [22] 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.
- [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] 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.
- [25] F. Multon, L. France, M. Cani-Gascuel, and G. Debunne. Computer animation of human walking : a survey. *Journal of Visualization and Computer Animation*, 10:39–54, 1999.
- [26] S. R. Musse. *Human Crowd Modelling with Various Levels of Behaviour Control*. PhD thesis, EPFL, Lausanne, Suisse, Jan. 2000.
- [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] P. Palamidese. Computer-aided staging. *The Journal of Visualization and Computer Animation*, 10:3–14, 1999.
- [30] B. J. Rhodes. *PHISH-Nets : Planning Heuristically In Situated Hybrid Networks*. PhD thesis, Massachusetts Institute of Technology, 1996.
- [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] T. Widyanto, A. Marriott, and M. West. Applying a visual perception system to a behavioral animation system. In *Eurographics Workshop on Animation and Simulation*, pages 89–98, Vienna, Austria, 1991.