

The InViWo Toolkit: Describing Autonomous Virtual Agents and Avatars

Nadine Richard¹, Philippe Codognet², and Alain Grumbach¹

¹ ENST
46 rue Barrault
75013 Paris, France
{Nadine.Richard, Alain.Grumbach}@enst.fr
² LIP6
8, rue du Capitaine Scott
75015 Paris, France
Philippe.Codognet@lip6.fr

Abstract. The INVIWO project aims at providing high-level intuitive tools to describe virtual worlds populated with “intelligent” creatures and avatars. For this purpose, we have defined the MARVIN language, which enables the high-level description of autonomous agent behaviours. In this paper, we present the underlying model we have designed, especially our agent and avatar architectures. We then present the main features of the MARVIN language and we introduce the use of constraints as powerful tools for describing and combining behaviours.

1 Introduction

1.1 Motivations

The INVIWO (*Intuitive Virtual Worlds*) project aims at providing high-level intuitive tools for describing inhabited virtual worlds, *i.e.* synthetic worlds where users can interact with objects and “intelligent” entities. A typical application area is entertainment worlds, such as the Diamond Park [35], a virtual leisure park where various virtual agents could either entertain users (*e.g.* circus agents) or assist them (*e.g.* bus drivers). A key point in the design of virtual worlds is the possibility to define autonomous entities that will “populate” the world and make it more attractive to the user.

Our objective is eventually to provide high-level tools, based on a specific language for agent behaviour description. This language, called MARVIN, enables the advanced user to define basic tasks such as goal-directed navigation or obstacle avoidance, by combining agent behaviours, triggering conditions and high-level goals in a simple way. MARVIN is essentially based on reactive rules, temporal control structures and constraints, and on the primitive actions an agent can perform.

The underlying agent model has been designed as a synthesis and a simplification of existing agent architectures, in order to present a minimal set of components that could be easily manipulated by non-computer scientists. Multi-agent systems are currently widely used to simulate or control complex systems, as they take advantage of the robustness and the flexibility of collaborating, adaptive and autonomous entities. Our agent architecture is fully decentralized and homogeneous.

Figure 1 shows an example of INVIWO worlds: the inhabitants are rabbits moving on a chess board. Each rabbit has to reach several predefined target points. When a rabbit detects a collision risk with an obstacle, for instance when a static cube or another rabbit appears into specified boundaries, it will try to avoid the obstacle while trying not to go too far from its goal trajectory. On the screenshot, the sensibility boundaries are represented as transparent spheres around the characters.

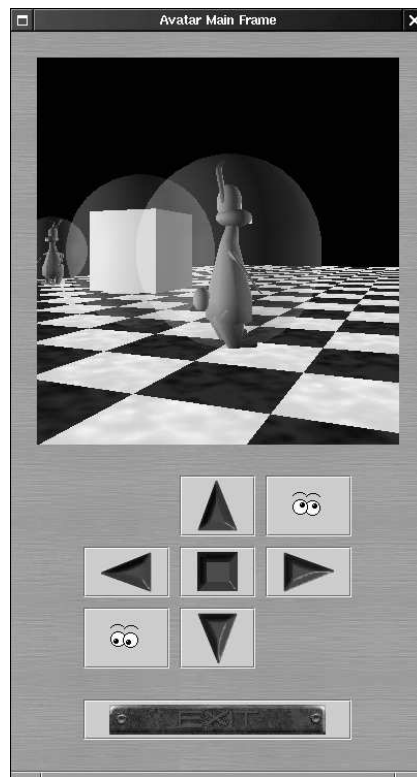


Fig. 1. The autonomous rabbits, seen from the avatar point of view.

1.2 Organization of the paper

This paper is organized as follows: first, we recall some background material on virtual agents and reactive architectures in section 2. Then section 3 details the model we have designed, *i.e.* the overall architecture we have chosen for inhabited virtual worlds, and the models we have defined for autonomous virtual agents and avatars. In section 4, we propose our approach to ease the description of agent behaviours: we describe our behaviour-based architecture for action selection, we present the MARVIN language and we introduce the use of constraints in our model. Before concluding this paper, we shortly describe the INVIWO toolkit in section 5.

2 Related work

2.1 Behaviour-based architectures

Classical Artificial Intelligence considers cognitive agents, which have symbolic representations of their environment and reasoning capabilities, in order to plan long-term actions. R. Brooks opposes the so-called “new Artificial Intelligence” to the cognitive approach, because of the complexity of the external environment to be symbolically represented [7]. This bottom-up, reactive approach is strongly inspired by ethology and biology; it has been successfully applied to virtual creatures called animats [5, 27, 30, 33] and to behaviour-based robotics [1, 7, 22]. As purely reactive agents lack of reasoning abilities and high-level, task-oriented behaviours, many authors have been advocating for hybrid agents that combine reactive and cognitive skills and that are thus able to have both reflex and deliberative reactions [1, 12, 20].

In behaviour-based architectures, sensing, action selection and acting are distributed among independent basic behavioural modules, that interact for achieving particular agent tasks (eating, mating, avoiding predators, *etc.*). The global behaviour of the agent emerges from the interaction of the concurrent modules. Behaviour-based architectures have been first investigated by R. Brooks and his well-known *subsumption architecture*, which consists in hierarchical layers of concurrent behaviours combined with inhibition mechanisms [6]. This architecture has been used for controlling robots that should quickly respond to dynamic and unpredictable environments.

2.2 Virtual agents

The reactive approach has been extensively used to animate virtual creatures, especially for navigation, obstacle avoidance, schooling and pursuit simulation [23, 24]. A classical example is presented in [34]; the authors have produced realistic animations of artificial fishes by adding biomechanical models to a behavioural architecture. Behaviour-based animations are usually only aimed for spectators that cannot interact with the virtual entities. Yet, there is a strong need for highly-interactive agents, for example in MUD-like worlds, which are designed

for gaming and chatting. In such environments, the user becomes a “spectator” interacting with assistants like hosts and security guards [21], chatterbots like JULIA [17], artificial pets as in the ALIVE system [18] or virtual humans [3, 31, 35].

2.3 Describing the behaviour of hybrid agents

Finite-state automata are traditionally used to describe reactive behaviours. Game characters [15] or vehicle simulations [9, 19] make use of hierarchical parallel automata, which reduce complexity when designing and executing concurrent behaviours. In particular, the PAT-NETS (*Parallel Transition Networks*) have been introduced for simulating virtual humans [2]. The main drawback of finite-state automata lies in their poor maintainability: a minor modification of the behaviour specification can lead to the complete redefinition of the automata.

Specific languages have been designed for high-level behaviour scripting [3, 16], in particular for reviving the cognitive approach in the field of computer animation [11]. The PAT-NETS have been recently extended with an abstract action model to provide high-level description and parametrization of generic actions [3]. Some efforts have also been made to propose intuitive graphical programming, for example in the VIRTUAL FISHTANK project [32].

3 The InViWo model

3.1 InViWo agents and worlds

We focus on decentralized virtual worlds, where no entity is in charge of managing the world. Unlike in [14], where the authors distinguish virtual humans and smart objects, we have chosen an homogeneous approach: we consider any object as an agent because each object can possibly autonomously react to incoming events. An INVIWO world is therefore a uniform multi-agent system, and the environment of an agent is just the set of the other agents. An INVIWO world is basically a set of agents, with no specific organization; such a world evolves depending on the agent interactions, creations and destructions. INVIWO agents are fully autonomous: the internal state and the behaviour of an agent cannot be directly manipulated by an external entity (a scene manager or another agent). Simpler agents are reactive, but agents that have a structured memory can use it to build their own representation of the environment, in order to make temporal and spatial reasoning.

Our agents communicate with each other *via* what we call *stimuli*. Stimuli are typed messages, which can contain structured data. Interaction with the environment is thus achieved only through message-based communication, according to three modes: unicast, multicast and broadcast communication. The INVIWO world designer can define multi-level communication within the same virtual world by describing appropriate interaction protocols between agents.

3.2 Architecture of the INViWO agents

An autonomous agent is a computational entity that is able to perceive its environment and act on this environment. It is autonomous since it decides what to do depending on the received stimuli, its own resources and its goals. The behaviour of an agent consists of a set of decision rules which lead to action selection. As shown in figure 2, an INViWO agent is made of a set of attributes representing the characteristics of the agent (*e.g.* its shape, position, or mood) and its knowledge (*e.g.* a personal map of the world), sensors to perceive internal and external stimuli, effectors to perform internal and external primitive actions, and a decision process.

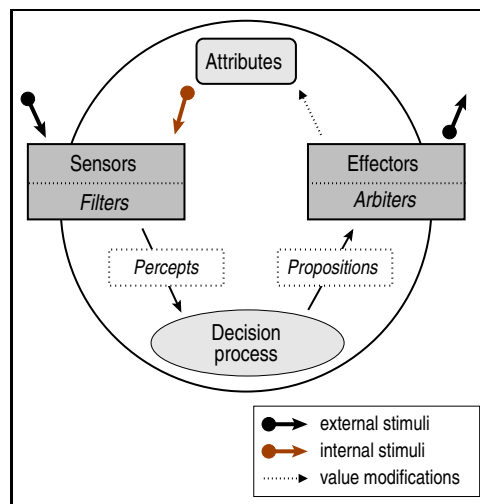


Fig. 2. Basic architecture of an INViWO agent.

Effectors encapsulate abstract actions to be realized through combinations of primitive actions. Three primitive actions are available:

- Send an external stimulus (according to a specified communication mode).
- Modify the value of an attribute.
- Modify the agent structure by adding or removing a component. This kind of actions enables evolutive agents.

An internal state modification, either a structure or an attribute change, causes the appropriate internal stimulus to be generated. Sensors can receive both external stimuli from the environment and internal stimuli from inside the agent. Figure 2 only represents the attribute modification and reaction, for clarity purposes.

Sensors translate the received stimuli into percepts and send those percepts to the decision process, after a filtering step. Activation bounds can be specified

in the sensors as facilities for external stimulus filtering. The decision process is made of independent decision-making units called behavioural modules, which run concurrently to select actions to be performed. A behavioural module is basically a set of rules activated on the received percepts, and that lead to action propositions made to the effectors (see section 4.1 for details). The arbiter associated with an effector should solve the conflicts between the requests from the concurrent behavioural modules when needed: the decision process of an agent is thus shared between the behavioural modules and the arbiters, as in the DAMN architecture [28]. The effectors finally perform the chosen actions.

3.3 The InViWo avatars

Within a virtual world, the human operator (or user) is usually represented by an avatar, which is in charge of two separate tasks: it represents the user towards the other users, and it includes navigation and basic perception capabilities. In most cases, the object representing the user can move within the world and a camera gives him/her a subjective view of the 3D scene. The simplest way to integrate users into INVIWO worlds is to consider the avatar as an agent, which both represents the user towards the other agents, and gives him a personal view of the environment with the capability to act on it: an INVIWO avatar is a specific agent, which can be interactively controlled by a human operator. We thus prefer to refer to the INVIWO avatar as a mediator of the user, rather than to his/her representation, since it is a bidirectional translator:

- It receives external stimuli, and it informs its user about its internal state and about the environment.
- It executes user commands by modifying its attributes and structure and by sending external stimuli into the environment.

As shown in figure 3, the interaction between the user and his/her avatar is realized by adding specific stimuli called *user stimuli*. The user interface should be able to send and receive user stimuli in order to communicate with the avatar. This model makes the user interface completely independent from the avatar; it also enables two original uses of avatars:

- A user can control several avatars in the same INVIWO world, and thus be represented by different characters and perceive the world through different points of view, at the same time.
- An avatar can be controlled by several users, and thus become a “multi-user avatar”.

In many existing systems, the user can have multiple views and perform actions in the virtual world, even if the avatar does not have the corresponding sensors and effectors. We think that a user should not be able to do more than his/her avatar can do: the user partly controls his INVIWO avatar, but he/she can be constrained by its abilities. When autonomous enough, the avatar may assist the user by executing repetitive or high-precision tasks, like walking on an

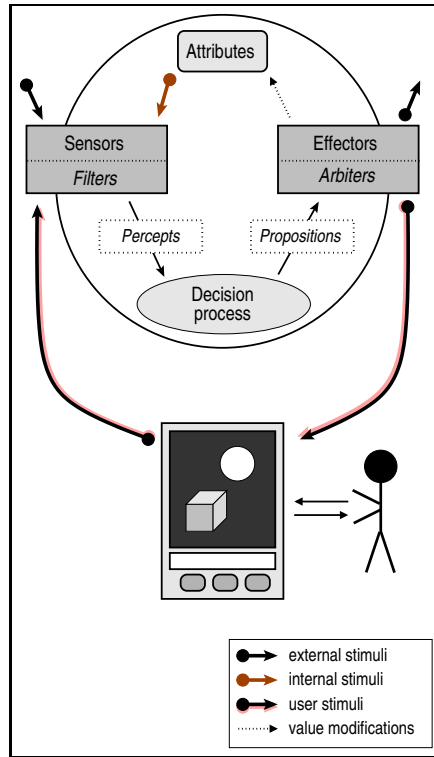


Fig. 3. The INViWO avatar, the user interface and the user.

irregular ground or grasping a glass of water. Moreover, the avatar can have its own personality, and thus constrain the user in a role, in order to get a believable behaviour: for example, a user should not be able to control the lurching behaviour of his drunk avatar.

As an INViWO avatar is an agent, the avatar autonomy can be achieved with little effort: in the example of figure 1, the avatar is based exactly on the same behavioural modules as the rabbits. We just added the ability to take user inputs into account by adding avatar-specific behavioural modules and the corresponding user stimuli: the user can request to go forward or backward, to turn right or left and to stop walking. Without having to program the corresponding behavioural modules twice, we obtain a semi-autonomous avatar, which avoids static and dynamic obstacles by itself. Conflicts between the autonomous part and the user-controlled part of the agent are solved by the arbiters in an homogeneous way.

4 Describing virtual agent behaviours

4.1 The behaviour-based architecture

Our behaviour-based architecture for action selection is strongly inspired from the principles of reactive synchronous systems, in particular from the ESTEREL language [4]. The INVIWO behavioural modules are reactive, concurrent units that should respond *instantaneously* to input signals by emitting output signals; it means that output signals should be produced in the same instant (timestep) as input signals. Signals are typed messages, which can contain structured values. The outputs of a module can be routed to the inputs of other modules, by creating specific channels; this connection is possible only if the linked ports have the same type. As shown on figure 4, the percepts built by the sensors are actually signals, as well as the action propositions received by the effectors.

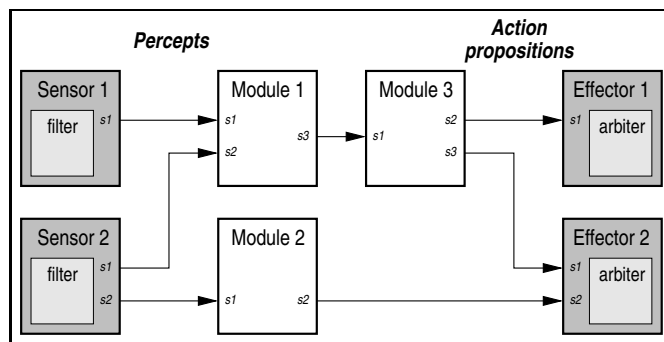


Fig. 4. Example of a behavioural network.

A behavioural module is basically made of a decision process, and possibly of internal variables and encapsulated subbehaviours. The decision process is roughly a set of decision rules, triggered when specific signals are received. The activation of a rule leads to the emission of output signals and possibly to the modification of internal variables. Sensors and effectors are similar to behavioural modules, with some differences:

- A sensor has no input signals, as it receives stimuli; its filter is able to translate the received stimuli into signals.
- An effector has no output signals, as it performs internal and external actions; its arbiter is able to perform primitive actions, such as sending an external stimulus or modifying the value of an attribute.

We have described a decomposition of the decision process into independent, concurrent behavioural modules. This decomposition enables modular programming (*e.g.* in MARVIN) as well as visual component-based description of the behaviour of an INVIWO agent.

4.2 The Marvin language

The MARVIN language is clearly inspired from ESTEREL, with the addition of object-oriented and agent-oriented features. Agents and agent components are specific objects. Behavioural modules described in MARVIN are composed of an interface and a body: input and output signals are declared in the interface part, and the body part contains a composition of imperative and reactive instructions. A behavioural module can only modify its own state or emit output signals. Sensors and effectors are syntactically similar to behavioural modules, to keep an homogeneous description language.

The following example of a MARVIN description shows how we can describe the management of the agent energy level. This level decreases at each timestep of the agent or when the `Loose` signal is received. We assume that the attributes `energy` and `energy_step` do exist; those attributes can be read by using the `owner` keyword, which refers to the agent:

```
behaviour loose_energy
  in Loose (int) ;
  out Reduce (int) combine with + ;

  body
    loop
      wait
        Tick -> emit Reduce (owner.energy_step)
        | Loose (a) with [ a > 0 ] -> emit Reduce (a)
      end wait
    end loop
  end body
end behaviour
```

The `Tick` signal, used in the first reactive rule, is a pure predefined signal corresponding to the internal clock; it doesn't need to be declared in the module interface. The second rule makes use of a guard concerning the received signal parameter; this guard ensures that the reaction will only lead to a decreasing of the energy level. Such a guard can also be applied to the agent attributes or to the internal variables of the module.

The two rules can be activated simultaneously if both signals are received at the same instant; those rules can thus emit simultaneous different values for the `Reduce` signal. To avoid inconsistencies, we have to define a combination method for this output signal; when both rules are triggered, the two emitted values are added (+) in order to get only one signal value for the current instant. The module we have defined should be linked with the appropriate sensors or other behavioural modules so it can be activated; it should also be routed to the appropriate effectors so it can have an effect on the `energy` attribute.

The next example describes a sensor, which supervises the energy level in order to warn the other behavioural components when its value falls below a specified threshold; we assume that this threshold is an agent attribute. This

behaviour is realized by waiting for the internal stimulus `_energy`, with the appropriate guard:

```
sensor energy_sensor
  out Dead ;

  filter
    waitstim _energy
    with [ owner.energy < owner.min_energy ] -> emit Dead
  end filter
end sensor
```

In the following example, we describe an effector, which accepts two kinds of action propositions: adding or subtracting a given quantity to the `energy` attribute. The `set` primitive affects the given value to the attribute (known by its identifier) and then generates the associated internal stimulus:

```
effector energy_effector
  in Add (int), Subtract (int) ;

  method sum (l)
    q <- 0 ;
    foreach p in l do
      q <- q + p
    end foreach ;
    return q
  end method

  arbiter
    loop
      add <- 0 ; sub <- 0 ;
      wait
        all Add as propositions ->
          add <- sum (propositions.values())
        | all Subtract as propositions ->
          sub <- sum (propositions.values())
        end wait ;
      set (owner.energy.id, (add - sub))
    end loop
  end arbiter
end effector
```

The “all S as *var*” construction means that all the S signals received simultaneously will be placed in a dictionary called *var*. This construction enables the arbiter to process the multiple propositions made by the behavioural modules for the current instant. As an effector is an object, we can define a method `sum` for factorizing code.

4.3 Using constraints

Constraint solvers for arbitration A constraint is simply a logical relation between several variables, which restricts the degrees of freedom of the variables, that is the possible values the variables can take in their specific domain. A constraint thus represents some partial information relating the objects of interest. The whole idea of constraint solving is to start reasoning and computing with partial information, ensuring the overall consistency and reducing as much as possible the domains of the variables in order to prune the search space. Constraint Programming combines the declarativity of high-level languages with the efficiency of specialized algorithms for constraint solving, sometimes borrowing techniques from Operations Research and Numerical Analysis [29]. It has proved to be very successful for Problem Solving and Combinatorial Optimization applications. While constraint solvers can be enhanced by using agent-based approaches, multi-agent systems can take advantage of the constraint computation paradigm [10].

In the INVIWO framework, constraint solvers can be used in the arbiters, in order to choose the actions to perform according to the propositions given by the behavioural modules. Classical constraint solvers can combine non-contradictory actions to get emergent behaviours. For example, a left move combined with a right move will lead to go straight. To handle possibly contradictory actions, we need to use *soft constraints* solvers, as detailed in [13]. Soft constraints are no more simple boolean logical relations, but are valued in a semi-ring structure. For instance, fuzzy constraints are valued in the set $[0..1]$ equipped with *max* and *min* as additive and multiplicative operations.

Goal constraints and adaptive search Constraints can also be used to express high-level goals in a declarative way: we call *goal constraint* a relation that the agent should try to achieve whenever it is not satisfied. We cannot consider classical Constraint Programming to handle dynamic and unpredictable environments: we therefore propose a new solving method called *adaptive search* to iteratively select actions that will eventually lead to the satisfaction of the goal constraints [8].

At each timestep, the agent should select the best action, in order to reduce the discrepancy between its current state and the overall satisfaction of the set of goal constraints. It is thus possible to reactively adapt the agent behaviour to a changing environment. It is worth noticing that behaviours are stated in an implicit way by giving a set of constraints and not in an explicit way, for example by giving a precise trajectory. Our framework can be used as a motivation architecture for virtual creatures, by considering constrained variables for denoting internal states (*e.g.* energy level or thirst), and goal constraints for defining internal needs (*e.g.* the energy level should stay above a given value), routine behaviours (*e.g.* if the energy level is too low, go for food) or external desired properties (*e.g.* stay away from predators).

We need the ability to handle internal variables, parametrized inputs and dynamic representations of goals to be achieved. Constraints are used to state

goals, or more precisely partial goals, that the agent has to achieve. The primitive spatial constraints for autonomous navigation are:

- *in(region)*: stay within the zone defined by *region*
- *out(region)*: stay outside the zone defined by *region*
- *go(object)*: move towards the location of *object*
- *away(object)*: move away from the location of *object*
- *attraction(stimulus)*: move towards the source of *stimulus*
- *repulsion(stimulus)*: move away from the source of *stimulus*

These declarative constraints will reduce to some arithmetic constraints. For a circle *region*, the *in(region)* constraint will for instance reduce to:

$$agent.position - region.center < region.radius$$

It is clear that a combination of such goal constraints could produce quite complex behaviours: a following behaviour can be simply obtained by combining a *go* constraint and an *out* constraint, in order to move towards the followed object but not too close. The limited set of primitive goal constraints has been chosen because efficient methods can be designed to solve them. Indeed, one can define a *repair* mechanism for each constraint: this mechanism will propose an action that could reduce the degree of violation of the constraint whenever the constraint is not satisfied. For example, the repair action for a *go* constraint could perform the following navigation step in the direction of the target *object*:

$$agent.position = agent.speed \times \|object.position - agent.position\|$$

Let us now detail how to solve a combination of goal constraints by choosing the most appropriate repair action. Local search methods, such as simulated annealing or genetic algorithms, are working by iterative improvement over an initial state and are thus well-suited to reactive environments. The basic algorithm consists in starting from a random configuration, explore the neighbourhood and then move to the best candidate; this process will continue until a satisfactory solution is found.

In our framework, we select the adequate repair action whenever goal constraints are not fully satisfied. The input of the method is a set of variables V , a set of constraints C over V , and a cost function F to be minimized (for example, the number of violated constraints). For each constraint, we need also an *error function* giving an indication on how much the constraint is violated. Adaptive search seeks to reduce the error on the worse variable. It also include an adaptive memory module to prevent being trapped by local minima, like in the Tabu search method: each variable leading to a local minimum is marked and cannot be chosen for new iterations. The algorithm starts from a random assignment of the variables in V , and then repeats the following steps until a solution is found or the maximal number of iterations is reached:

1. **Compute scores** of constraints in C and combine scores on each variable.
2. **Select variable** X with highest score and evaluate cost of all possible moves from X .

3. **If** no better move **then** mark X tabu **else** select the best move and change the value of X accordingly.

This very simple method is surprisingly efficient to solve complex combinatorial problems such as the well-known “magic square” puzzle (see [8] for details). This framework also naturally copes with over-constrained problems. As defined above, this method does not perform any planning, as it only computes the next move out of all possible current moves. A simple extension would be to allow some limited planning by considering not only the immediate neighbours but all configurations on paths up to a given distance, and then choose to move to the neighbour in the direction of the most promising node. Therefore the method can plan for the best trajectory in a limited time-window. Goal constraints are expressed in MARVIN as specific entities, based on behavioural modules.

5 The InViWo toolkit

The current prototype of the INViWo toolkit is developed in Java, to be portable and to take advantage of simple GUI design. We use the Java3D API for displaying the 3D view of avatars.

The low-level library implements a programming interface and an execution platform. The platform is in charge of executing agent behaviours and of handling the communication between entities. The API provides the basic generic agent and avatar components we described in sections 3.2 and 3.3. Programmers can easily extend our model and develop new components. No interpreter for the high-level language is available at the moment: MARVIN descriptions are specifications that we manually translate into Java programs, which make use of the INViWo API.

6 Conclusion

We have proposed a multi-agent model for inhabited virtual worlds, *i.e.* worlds populated with autonomous agents and avatars. We have designed a synthetic agent architecture, in order to ease the description of virtual creatures. We have extended this architecture to integrate avatars while keeping an homogeneous model; moreover, avatars can be partly autonomous and thus have their own behaviour or personality.

The control architecture of our hybrid agents is a behaviour-based model, where concurrent behavioural modules propose actions and where arbiters select final actions to be performed among those propositions. Behavioural modules can be specified in the high-level language MARVIN, inspired from ESTEREL. The use of constraints extends the expressivity of our language, and makes it possible to add some planning capabilities to the INViWo agents.

References

1. R. ARKIN. *Behavior-based Robotics*. MIT Press, 1998.
2. N. BADLER, C. PHILIPS, B. WEBBER. *Simulating humans: computer graphics, animation, and control*. Oxford University Press, 1993.
3. N. BADLER ET AL. *Real Time Virtual Humans*. International Conference on Digital Media Futures, Bradford (UK), April 1999.
4. G. BERRY *The ESTEREL v5 language primer*. Technical Report, Centre de Mathématiques Appliquées (INRIA et cole des Mines de Paris),1999.
5. B. BLUMBERG, P. TODD, P. MAES. *No bad dogs: ethological lessons for learning in Hamsterdam*. From Animals to Animats 4, Cape Cod (USA), September 1996.
6. R. BROOKS. *A robust layered control system for a mobile robot*. IEEE Journal of Robotics and Automation, April 1986.
7. R. BROOKS. *Cambrian Intelligence: the early history of the new AI*. MIT Press, 1999.
8. P. CODOGNET. *Declarative behaviors for virtual creatures*. ICAT'99, 8th International Conference on Augmented Reality and Tele-existence, IOS Press 1999.
9. S. DONIKIAN, E. RUTTEN. *Reactivity, concurrency, data-flow and hierarchical preemption for behavioural animation*. Eurographics Workshop on Programming paradigms in Graphics, Maastricht (The Netherlands), September 1995.
10. P. EATON, E. FREUDER, R. WALLACE. *Constraints and agents*. American Association for Artificial Intelligence, Summer 1998.
11. J. FUNGE, X. TU, D. TERZOPOULOS. *Cognitive modeling: knowledge, reasoning and planning for intelligent characters*. SIGGRAPH'99, Los Angeles (USA), August 1999.
12. E. GAT. *Reliable goal-directed reactive control for real-world autonomous mobile robots*. Ph. D., Virginia Polytechnic Institute and State University, 1991.
13. Y. GEORGET AND P. CODOGNET. *Compiling Semiring-based Constraints with clp(FD,S)*. CP'98, 4th International Conference on Constraint Programming, Pisa (Italy), Lecture Notes in Computer Science, Springer Verlag 1998.
14. M. KALLMAN, D. THALMANN. *A behavioral interface to simulate agent-object interactions in real time*. Computer Animation'99, Geneva (Switzerland), May 1999.
15. Y. KOGA, C. BECKER, M. SVIHURA, D. ZHU. *On intelligent digital actors*. Imagina'98, Monte-Carlo (Monaco), 1998.
16. B. LOYALL, J. BATES. *Real-time control of animated broad agents*. 15th Annual Conference of the Cognitive Science Society, Boulder (USA), June 1993.
17. P. MAES. *Artificial Life meets entertainment: lifelike autonomous agents*. Communications of the ACM, Special Issue on New Horizons of Commercial and Industrial AI, 38(11), November 1995.
18. P. MAES ET AL. *The ALIVE system: wireless, full-body interaction with autonomous agents*. Technical Report of the MIT Media Laboratory Perceptual Computing, November 1995.
19. G. MOREAU, S. DONIKIAN. *From psychological and real-time interaction requirements to behavioural simulation*. Workshop on Computer Animation and Simulation'98, Lisbon (Portugal), September 1998.
20. J. MILLER. *Control architectures for autonomous and interacting agents: a survey*. Intelligent Agent Systems (Theoretical and Practical Issues), Springer-Verlag, 1997.
21. C. PAUL, R. PETERS, A. GRAEFF. *Agents in multi-user virtual environments*. Eurographics'98 short paper session, Lisbon (Portugal), September 1998.
22. R. PFEIFER AND C. SCHEIER. *Understanding Intelligence*. MIT Press, 1999.

23. C. REYNOLDS. *Flocks, herds, and schools: a distributed behavioral model*. SIGGRAPH'87, Anaheim (USA), July 1987.
24. C. REYNOLDS. *Steering behaviors for autonomous characters*. Game Developers Conference, San Jose (USA), March 1999.
25. N. RICHARD, P. CODOGNET, A. GRUMBACH. *Constraints to describe high-level behaviours in virtual worlds*. Eurographics'98 short-paper session, Lisbon (Portugal), September 1998.
26. N. RICHARD, P. CODOGNET, A. GRUMBACH. *The INVIWo virtual agents*. Eurographics'99 short-paper session, Milano (Italy), September 1999.
27. H. ROITBLAT AND J-A. MEYER (Eds.). *Comparative approaches to cognitive science*. MIT Press, 1995.
28. J. K. ROSENBLATT. *DAMN: a Distributed Architecture for Mobile Navigation*. Ph. D. thesis, Robotics Institute, Carnegie Mellon University, 1997.
29. V. SARASWAT, P. VAN HENTENRYCK, P. CODOGNET ET AL. *Constraint Programming*. ACM Computing Surveys, 28(4), December 1996.
30. N. SCHMAJUCK (Ed.). *Special issue on biologically-inspired models of navigation*. Adaptive Behavior, 6(3/4), Winter/Spring 1998.
31. D. THALMANN, H. NOSER. *Towards autonomous, perceptive, and intelligent virtual actors*. Artificial Intelligence Today, Lecture Notes in Artificial Intelligence, 1600, Springer, 1999.
32. M. TRAVERS. *Behave! A visual behavior language*. <http://xenia.media.mit.edu/~mt/behave/behave.html>.
33. O. TRULLIER AND J-A. MEYER. *Biomimetic navigation models and strategies in animats*. *AI Communications*, vol. 10, no. 2, 1997.
34. X. TU, D. TERZOPOULOS. *Artificial fishes: physics, locomotion, perception, behavior*. SIGGRAPH'94, Orlando (USA), July 1994.
35. R. WATERS ET AL. *Diamond Park and Spline: a social Virtual Reality system with 3D animation, spoken interaction, and runtime modifiability*. Technical report of the Mitsubishi Electric Research Laboratory, November 1996.