12. Intelligent Virtual Agent Systems for Interactive Stories

Version 1.0 – March 2005

Interactive Storytelling for Creative People

Deliverable Number: 3.1.1	**Workpackage:** 4
Contractual Date of Delivery: 03/2005	**Actual Date of Delivery:** 03/2005
Nature: Report	**Authors:** Luc Chustres
Public Deliverable	

Information Society
Technologies

Contents

12.1 INTRODUCTION 279
12.2 DEFINITIONS AND CONCEPTS 280
12.2.1 Agent........ 280
12.2.2 Goal........ 281
12.2.3 Autonomy........ 281
12.2.4 Behaviour 282
12.2.5 Communication 285
12.2.6 Social Organisation........ 288
12.3 MAKING VIRTUAL AGENTS INTELLIGENT 289
12.3.1 Fundamental AI Technologies........ 289
12.3.2 Existing Behaviour Based Control Architectures........ 291
12.3.3 Social Organisation........ 300
12.4 BUILDING AN EFFICIENT AGENT SYSTEM 301
12.4.1 Scalability........ 302
12.4.2 Existing Solutions........ 307
12.4.3 New Paradigms for Distributed Simulations........ 309
12.5 CONCLUSION........ 315
12.6 REFERENCES 317

Table of Illustrations

Figure 1 : Schematic view of an agent 281
Figure 2 : Behaviour based control architecture 284
Figure 3 : An autonomous agent as described by Monzani [Monzani-ABAVH02] 285
Figure 4 : The publish-subscribe paradigm........ 286
Figure 5 : Organisation of agents based on groups and roles........ 288
Figure 6 : Graphical representation of a virtual dog finite state machine 289
Figure 7 : Blumberg 5-layered architecture [Blumberg-MLDACRTVE95]........ 292
Figure 8 : The behaviour system layer in the Blumberg architecture [Blumberg-MLDACRTVE95]........ 293
Figure 9 : The run-time architecture of the Improv system [Perlin-ISSIAVW95]........ 293
Figure 10 : Actor communication through a shared blackboard in the Improv system........ 294
Figure 11 : Layered structure of the subsumption architecture with levels of competence 295
Figure 12 : An example PaT-Net for the hide and seek game [Badler-TPAARPB97] 296
Figure 13 : Syntax of the HPTS language........ 298
Figure 14 : An HTN representing methods for travelling and a plan based on this HTN for travelling from University of Maryland to MIT........ 300
Figure 15 : The organisation model of Chevaillier [Chevaillier-HAAIVEFFT01] in UML........ 301
Figure 16 : Differences between TCP and UDP 303
Figure 17 : The different modes of communication........ 303
Figure 18 : Each dot represent an agent and circle represents Area Of Interest (AOI) of a particular agent 306
Figure 19 : The region size selection problem. Left: AOI is smaller than region, Right: AOI is larger 306
Figure 20 : Technical components of HLA........ 310
Figure 21 : Ambassador paradigm in HLA........ 312
Figure 22 : OMG reference model architecture 313
Figure 23 : CORBA ORB architecture 314

12.1 Introduction

INSCAPE aims at enabling ordinary people to use and master the latest Information Society Technologies for interactively conceiving, authoring, publishing and experiencing **interactive stories** whatever their form, be it theatre, movie, cartoon, puppet show, video-games, interactive manuals, training simulators, etc. All of these applications raise the question of embedding a consistent and evolutionary (**non-linear** or **dynamic**) scenario into the system, in order to more precisely control the semantic that the users will interpret while interacting with it. As a consequence, the need of creating autonomous entities (commonly called **intelligent virtual agents** or **IVA**), capable of taking decisions based on their perceptions and motivations grows [Wooldridge-MASMADAI99]. The stake is then the conception of agent architectures modelling physical mechanisms [Chavaillier-VRMASMSIP00], living organisms [Drogoul-SMARCP93], or human beings [Magnenat-CMASA91]. Authoring such architectures take one's inspiration from artificial intelligence (**AI**), artificial life (**AL**), multi-agent systems (**MAS**) or robotics. Like most aspects of virtual reality (**VR**), this field is highly interdisciplinary.

A particular attention has been given to virtual characters or actors, as they are usually a predominant part of human stories, training simulators, and populate virtual worlds [Tisseau-RVAV01]. Since complete autonomy is still difficult to reach, especially for such complex creatures as virtual humans, a **plan**-based approach, which relies on some pre-defined behaviours, is usually favoured rather than AL techniques. However, the balance between automation and control is not a trivial task [Zicheng-KMORST95]. The key is to make the jointly use of a behavioural engine that generates high semantic level orders and a dedicated system that computes the final low level motion of the character [Menou-RTCAMLSSO01]. Indeed, the decisions taken by the behavioural engine has to be converted to concrete actions. It is also necessary to include the user by representing it through a specific model of an **avatar** within the system. The user is thus placed at the same conceptual level as the digital models that make up the virtual world. In other words, he is also an agent whose decision-making abilities are delegated to the human operator that it represents.

As a general rule, **virtual interactive storytelling** lies between two limits, characterized by opposite **agency** [Perlin-BACGUPM04]: movies where the story takes the whole agency, and video-games where the player has the agency when controlling the hero. These two limit-cases lead to two different behaviour paradigms. A *bottom-up* approach, which tries to create **emerging** personality based on the basic actions that the agent can perform. On the opposite side, a *top-down* approach, which embeds a **pre-defined** personality into the agent that interacts accordingly with its environment. Clearly, the latest is more suitable for storytelling and training simulation. Indeed, it is difficult to predict agent's behaviour with a bottom-up approach. Autonomy helps to construct adaptive behaviours and non-linear stories, but the main goal should be known and the results more or less "predictable". More, complex behaviours are tedious to obtain from a bottom-up approach. Nevertheless, artificial life tools such as neuronal networks [Pina-OACAAFENN98] or learning classifier systems [Luga-CBAIDVRS98][Sanza-ECVECS01] produced some convincing results. Recent works also demonstrated that a **story** can emerge from distributed roles between individual actors that pursue their own goals [Cavazza-CSAAIVS01][Nijholt-VSSCIA02]. But in our case, the author should have the general idea of the **plot**, and the autonomous agents ensure to make the simulation dynamically evolve **around** this plot. In other words, we are interested in an hybrid approach between two diametrically opposed approaches with respect to the "amount" of autonomy of the agents. At the **implicit** or **agent-based** extreme, the storyline emerges from the autonomous actions by a set of intelligent agents. At **explicit** or **script-based** extreme, the agents have no autonomy and therefore no control over the plot at all.

Under these circumstances, we consider that interactive storytelling relies on the following hypothesis:

- the scenario is pre-defined with various branching conditions at certain points, so that the agents can adapt to various situations

- agents are able to perceive variations in their environment and to react to changes
- plans are specified prior to simulation and agent will not create new plans at run-time

It seems essential to us that autonomy helps the agent reacting to unexpected events, but it still more or less follows a pre-defined plot. A virtual agent in interactive storytelling is somewhere in between a complete autonomous agent and an avatar. Most of its behaviours, emotion, and responses to the changing environment are described in story input.

Building intelligent agents is the first step needed by an authoring tool such as INSCAPE. But the underlying architecture of the whole system may also be robust and efficient. It has to deal with **large** data sets (geometrical representations), **thousands** of intelligent virtual agents which own complex behaviours (for example training simulations), and **high** synchronisation frequencies to ensure a satisfactory level of interaction and consistence. More, it will possibly require the collaboration of multiple users in order to achieve the construction of complex scenarios. The content of the virtual world will depend on the capability of this world. On the other hand, communications infrastructure is improving all the time, as telecommunications companies move to fibre optic links, and cable companies bring copper or fibre optic cable into many homes. Virtual worlds will be just one of the services or applications to use this infrastructure, though perhaps the most significant.

In this context, **networked virtual environments** (**NVEs**) have appeared to propose a solution to the realisation of complex applications. INSCAPE, as other VR systems, will really benefit from, and may be require, a distributed realisation especially to handle multi-users and multi-agents. NVEs provide computational power/memory sharing and efficient communication schemes. They aim at achieving sufficient complexity for a given application based on a **smart** management of the set of limited resources available for the application. More, they attempt to offer **scalable** models and architectures in order to accommodate a large number of simultaneous agents or users and to make their dynamic addition to the system light and easy.

The structure of the document is as follows. First, we precisely define the terminology and the concepts used along the rest of the document. Secondly, we survey work on making virtual agents “intelligent”. We explore basic AI technologies and also relevant existing solutions. Then we focus on how to build a scalable distributed system. In the first part of this study we introduce the scalability problem and technical solutions proposed. In the second part of the study, the different currently available systems for **distributed virtual reality** (**DVR**) are examined. Finally, we provide the conclusions as some keys to build an efficient MAS for interactive stories.

12.2 Definitions and Concepts

12.2.1 Agent

There is no actual consensus on what an agent is, but several key concepts are important to this emerging paradigm [Ferber-MASIDAI98]. Therefore, we choose to define an **agent**, in the context of MAS (where many agents can coexist in the same environment), as a virtual entity able to :

- perceive its environment (**awareness**)
- act inside its environment (**reactivity**)
- achieve individual or collective objectives (**goal-directed**)
- communicate with other agents (**language**)
- provide services through its own skills (**collaborative**)
- have a behaviour based on its internal states (**autonomous**)

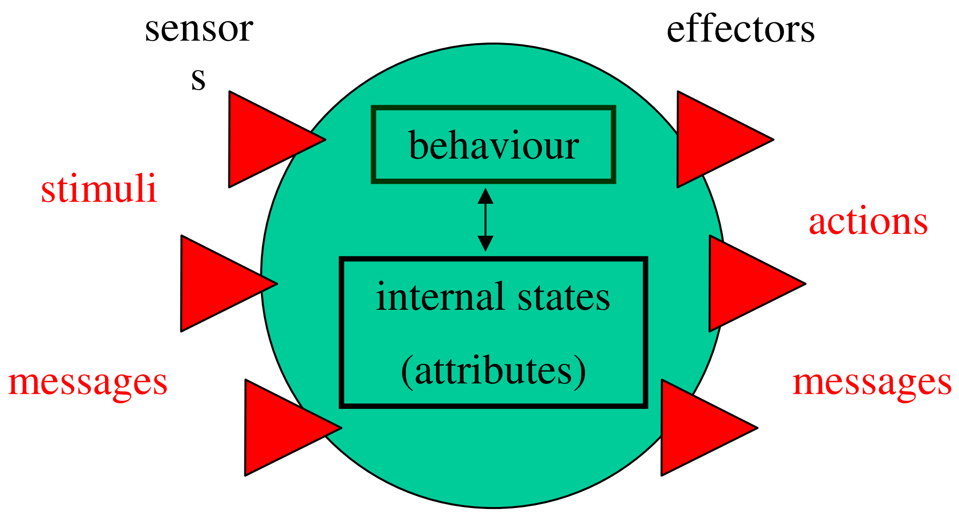


Figure 1 : Schematic view of an agent

An agent receives perceptible **stimuli** (shapes, sounds, messages, …) through **sensors** and acts on its environment through **effectors** or **actuators** (that eventually produce new stimuli). An agent owns a set of **behaviours**, which modifies its **internal states** (emotions, resources, …) and commands **actions** to its effectors, in order to achieve a specific **goal** or **task**. This paradigm defines a generic architecture that can either be used for virtual objects, virtual characters, avatars, physical mechanism, applications, processes, etc.

12.2.2 Goal

Without a goal the agent is lost, aimless since it has nothing to do, no plan will be invoked. Agents have one **main goal** and one or several **sub-goals**. The main goal is the objective that the agent is trying to achieves at a certain moment. During this process, the agent has to deal with intermediate sub-goals on which the **outcome** of the larger one relies on. For example, if the goal of the agent is to take the train, it should go to the desk, buy a ticket, and get into the train. The main goal is actually described as a **stack** of intermediate goals or tasks to perform. Only the task on the top of the stack is executed at a specific time, then deleted and popped when completed. Once the stack is empty the main goal is achieved.

Actually, **plans** constitute the most generic description of an agent's behaviour. They are often **hierarchical** plans that relate agent's intentions, goals, or high-level tasks to **primitive** actions that can be mapped onto low-level animation sequences for instance.

12.2.3 Autonomy

We can consider **autonomy** as the quality of being **self-governing** or **self-controlled**, without requiring the continual intervention of a user. As said earlier it relies on the coordination of agent's perceptions and actions [Meyer-SABARP91]. Going further, we can distinguish autonomy by **essence**, by **necessity** or by **ignorance** [Tisseau-RVAV01].

Autonomy by essence characterizes all living organisms and concerns adaptive behaviour for survival in changing environments. The goal is to stimulate the **genesis** of such behaviours in virtual environment. **Animats**, whose behaviour is based on real animals behaviour, are an example [Wilson-KGAA85]. Research in this field is highly related to AL and theory of learning (epigenesis) [Barto-LLIAS81], development (ontogeny) [Kodjabachian-EDNCLGFOAAI98] and evolution (phylogenesis) [Cliff-EER93] of control architectures [Meyer-FYAR94][Guillot-WNA00]. Introducing autonomy by essence allows us to understand autonomy observed in an organism.

Autonomy by necessity involves the recognition of changes in the environment, possibly due to the activity of other agents. The objective is to allow the agent to react to unplanned situations that come up during execution.

Autonomy by ignorance reveals our inability to analytically explain the behaviour of complex systems composed of heterogeneous entities. The idea is to distribute the control over the system's components by autonomize them. The evolution of these components enables a better understanding of the behaviour of the entire overall system.

These three different autonomy concepts play a part in virtual environments. The first concept presented is principally studied by AL. It focuses on **creating** behaviour from **scratch**. As explained in the introduction, it may result in unpredictable behaviours and is restricted to simple behaviour emergence such as locomotion [Sims-EMBC94]. The third concept is the central problem addressed by MAS. It relies on the assumption that an intelligent behaviour can emerge from interactions between agents that are more reactive placed in an environment that is itself active [Brooks-IWR91]. The second concept, which is somewhere between the two precedents, is probably the most well-suited and commonly used to experience interactive stories. The intelligence of the simulation comes from the behaviour of the autonomous agents but also from the autonomous agents interactions. In a sense, autonomy only means adaptation to environment variations for us.

12.2.4 Behaviour

The goal of **behavioural models** is to simulate the behaviour of different kinds of "living" (or nearly-living) things from plants [Reffye-PMFBSD88][Prusinkiewicz-APD93] to living beings like animals and persons [Badler-SHCGAC93][Badler-MMMCAAF91]. They define the internal and external behaviour of the entity, and also its actions and reactions. We can distinguish the three main components of such models:

- the **perception** module, responsible of **sensing** the world
- the **decision** module, responsible of **analysing** and **reasoning**
- the **action** module, responsible of **responding** to the sensory stimuli

The development of an autonomous agent requires automated sensing or perception, reasoning or planning and control or action. The data flow inside the agent is thus organized according to **sense-decide-act** (**SDA**) cycles.

Perception

Perception is the entry point of any behavioural model, which consists of creating an information flow from the environment to the agent in order to give him knowledge of its surrounding environment. A simple solution is to give the agent access to the information through the system that manages the environment. For example, if the environment is considered as made of 3D geometric shapes, the agent can have access to exact position of each objects and agents in the complete database of the synthetic world. Clearly, this solution becomes impracticable when the number of objects or agents increases. More, it does not correspond to reality where people do not have knowledge about the complete environment but only a limited part. A simple idea is to restrict the database to the portion of the world surrounding the agent (a sphere, a view cone, etc.).

However, to be more realistic, the agent should be **situated** and perceive the environment through visual, tactile, haptic, and auditory sensors [Thalmann-VSKTALVA95]. These **virtual sensors** are actually filters applied to the full environment information flow. This approach is also appropriate to include physical mechanisms such as cameras, detectors or sensors (IR, ultrasound, etc.) particularly useful for training simulations. However, in many scenarios, sensing the world by analysing and extracting valuable information from raw data is a time-consuming process. As a consequence, agent's perception in virtual simulations is often tragically simplified and/or specialized.

Virtual Vision

As **vision** is the most important sense of most living beings and is essential for navigation, it has been deeper studied than other senses, particularly for intelligent mobile robots [Horswill-SCRVNS93][Tsuji-MRRS93]. Renault et al. [Renault-VABA90] first introduced vision as a main information channel between the environment and the agent. The author point out that designing synthetic vision of a synthetic actor is a less complex task because it avoid, by *essence*, the problems of pattern recognition and distance detection. Indeed, computer graphics rendering methods generate direct knowledge of the object projected in each pixel and numerical information giving the distance. Typically, the agent perceives his environment from a small window in which the environment is rendered from his point of view. Several authors [Reynolds-EVBMCGM93][Tu-AFPLPB93][Blumberg-MLDACRTVE95] have adopted the same approach for simulating group behaviour or extended it to include visual memory [Noser-NDASVML90].

Virtual Audition

In real life, sounds are less important than vision by themselves, but they are essential because they indicate where to point eyes out. In opposition to vision, where light speed can be assumed infinite, sound renderers have to take care of propagation speed for convincing and realistic results. Noser et al. developed a framework for modelling a 3D acoustic environment with sound sources and microphones [Noser-SVADA95] but research in this area is still marginal. Much more attention has been given to speech recognition because a considerable part of human communication is based on speech. In this specific case, the sounds produced by the user are usually not simulated into the virtual world but directly translated to the internal agents' language.

Virtual Tactile and Haptic

Simulating the **hapting** system corresponds roughly to a collision detection process. Usually, a set of sensor points are attached to an agent and collide with the environment (objects and other agents) as they move around in space. As an example, Huang et al. [Huang-MSAGI95] use spherical multi-sensors, adapted from the use of proximity sensors in robotics [Espiau-CARRPS85], attached to their articulated figures.

Decision

The analysis or reasoning core is what people often think about AI systems, but it is the part that actually uses the sensory data to analyse the current situation and make a decision.

Two opposite trends in AI (**classical** and **new** AI) naturally lead to two kind of agents:

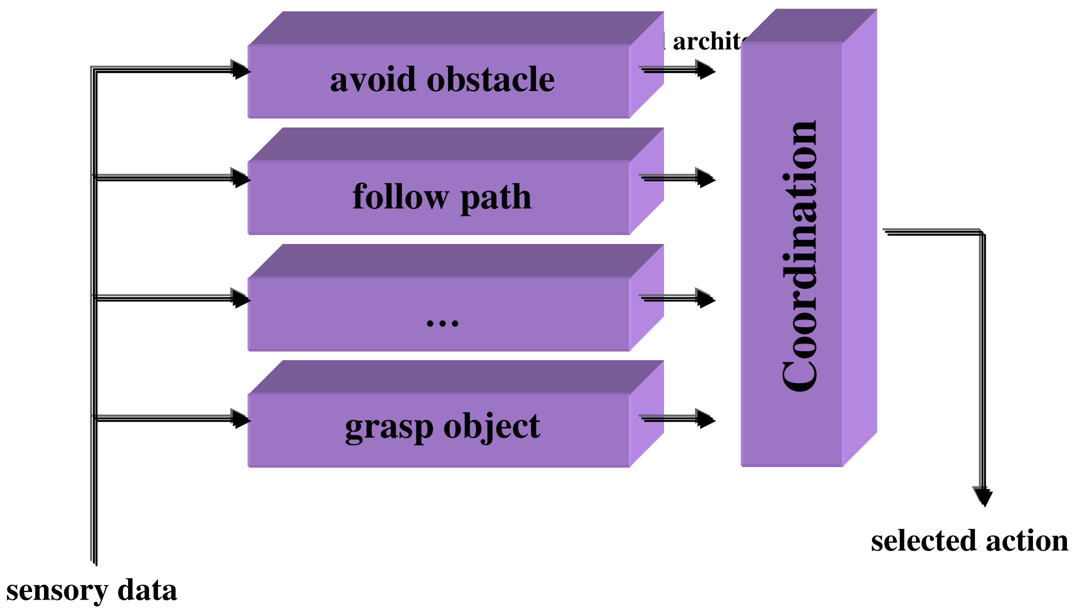
- **cognitive** agents, which perception and reasoning are described by symbolic representations [Ingrand-ARTRSC91]
- **reactive** agents, which link the sensors to the effectors by a **function** without memory [Maes-DAA91]

Cognitive or **deliberative** approach, which comes from classical AI, is often based on **BDI** architectures defining agent's behaviour in term of **belief**, **desires** and **intentions** [Wooldridge-RRA00], also known as **knowledge-based systems** (**KBS**) [Anastassakis-VASIAS01]. They allow the definition of inference rules which manage the execution of agent's actions. Cognitive models go beyond behavioural models presented later, in that they govern what a character knows, how that knowledge is acquired, and how it can be used to plan actions. The main shortcoming are the symbolic definition of the agent's perception requiring significant expertise, and the complexity and the slowness of the inference mechanisms that make them impracticable for real-time simulations including thousands of agents. Unlike cognitive agents, the reaction of reactive agents to a modification of their environment is fast. However, their behaviour remains rather reactive than intelligent or complex. It seems that an hybrid approach should give promising results for interactive stories.

Following this idea, **behavioural based control/animation** aims at distributing entity control among a set of behaviours. Each behaviour is responsible for one specific aspect of control. Brooks [Brooks-IWR91] introduced these **multiple layers architectures**, where each layer is an isolated computational unit that implements the whole cognitive process, i.e. the sense-decide-act cycle. Because the different behaviour modules are simple, specialized, and can be run in parallel, their response is fast and well-suited for real-time applications. Thus the major issue in the design of behavioural based control systems remains the formulation of effective mechanisms for coordination of the behaviours into strategies for rational and coherent behaviour. This is known as the **action selection problem** (**ASP**). Numerous **action selection mechanisms** (**ASM**) has been proposed over the last decade [Pirjanian-BCMSA99]. They are divided into two main groups: **arbitration** or **competitive** ASM, that can handle one behaviour at time, and **command fusion** or **cooperative** ASM, that can handle multiple behavioural constraints at time.

Action

Once the action has been selected by the reasoning core, it has to be translated into **concrete events**. Concrete events include modifying the representation of the agent in the virtual world, modifying the immediate environment of the agent, modifying the internal states of the agent (resources, emotions, etc.) and communicate decision results to other agents (send orders for instance). The applicability or usefulness of each action is a function of the current state of the environment. When an action is selected and performed, its invocation alters the environment, thus influencing the selection of future actions. In this way, a sequence of actions (a **plan**) emerges. Actually, as defined by Tu [Tu-AFPLPB93]: actions are the **atomic** components of behaviours.



In virtual worlds, each behaviour is also **physically** implemented by an animation, or a sequence of animations. Therefore, the main tasks consist in describing the low-level motions needed by the system to perform actions and translating the high-level actions into low-level motions [Menou-RTCAMLSSO01]. Monzani [Monzani-ABAVH02] presented an elegant solution to this problem. He separated the module responsible for handling the animation of the virtual body and the interaction with the environment: **embodied agent** (**EA**), and the module that takes decisions and behaves: **intelligent virtual agent** (**IVA**). The EA acts as a library of motions triggered by the IVA but also as the part responsible for sensing the environment. However, this technique is unwieldy because it adds a new communication layer between the IVA and the EA, which is an additional time-consuming process in already complex simulations with many agents.

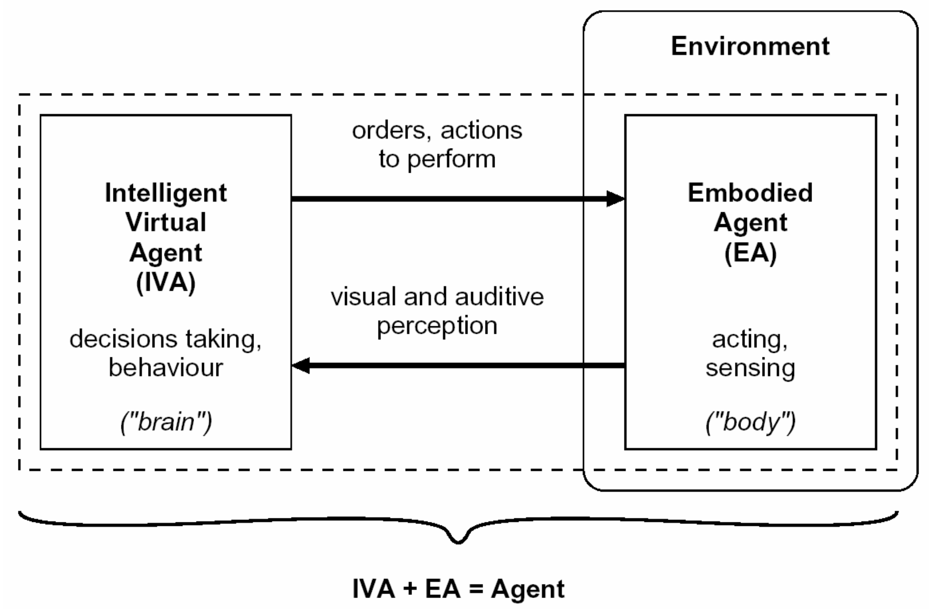

Environment
Intelligent Virtual Agent (IVA)
decisions taking, behaviour
("brain")
orders, actions to perform
visual and auditive perception
Embodied Agent (EA)
acting, sensing
("body")
IVA + EA = Agent

Figure 3 : An autonomous agent as described by Monzani [Monzani-ABAVH02]

12.2.5 Communication

When conceiving a MAS, the author mainly focuses on the **local** behaviour of the different agents and not on the behaviour of the **global** system. Then, in order to obtain a consistent global behaviour, he has to describe the interactions between entities themselves and with their environment. Most non-trivial simulations model the interaction of multiple entities, and coordination between those entities is often the most important part of the model.

Basics

The prevalent interaction inside a MAS is the exchange of **messages** (**direct** communication between agents) or **events/updates** (reflecting environment changes), which types are usually known in advance by the agents of the system. These messages can be broadcasted or send to a particular agent. This type of communication is the minimal ability of an agent, that makes the difference from an object. The reception of a message by an agent may be viewed as the call of a particular method on an object. The agent process the message and decide to (not) invocate the method according to its internal states.

The language of **speech acts** is the most evolved form of communication. It includes requesting, promising, offering, proposing, etc. KQML was one of the first protocol defining such a language [Finin-KACL94]. This **performative** (the speaker is asserting that s/he is doing a request, order or whatever) based language has been the foundation of the FIPA protocol used for communication between heterogeneous systems [OBrien-TSSA88]. Although such performatives can characterize message types, efficient language to express message content that allow agents to understand each other have not been demonstrated effectively. The ontology problem, that is, how agents can share meaning, is still open in the MAS community [Gruber-TAPO93]. More, using such languages implies coupling with evolved cognitive systems, able to reasoning on ambiguous messages, which are too much complex as we said earlier for our interests.

What is called **indirect** communication consists in using the environment to send signals or modifying it (objects creation, destruction or modification). No direct contact between the different agents is possible in the simulation. This type of communication can be unintentional and participate to the emergence of a reactive cooperation when the agents are not aware of other agents [Ballet-MASDCS97]. On the opposite side, when the agents are aware of other agents, they need to infer the actions of other agents. In this case, agents need a model of other agents such as the one proposed by Parenthoen using **fuzzy cognitive maps** [Parenthoen-APCVWFCMW01].

Methods

Theoretically, in direct communication, every pair of agents has a communication link between them. However, this simple model lead to a complex architecture and network overload. It seems obvious that enhanced models are required. Two types of efficient communication method – **message-passing** and **blackboards** – have been proposed.

The **message-passing system** uses a straightforward subscription-based addressing scheme. This requirement has been filled by various middleware products that are characterized as **messaging**, **message oriented middleware** (**MOM**), **message queuing**, or **publish-subscribe** [Mathur-PSPSGCS95]. Systems in which communication is done through a publish and subscribe paradigm require the sending agents (**publishers**) to publish messages without explicitly specifying recipients or having knowledge of intended recipients. Similarly, receiving agents (**subscribers**) must receive only those messages that the subscriber has registered an interest in. This decoupling between senders and recipients is usually accomplished by an intervening entity *between* the publisher and the subscriber, which serves as a level of indirection. This intervening entity is a *queue* which is used to represent a **subject** or **channel** of communication. A subscriber subscribes to a queue by expressing interest in messages enqueued to that queue and by using a subject- or content-based **rule** as a filter. Agents can also maintain individual message queues, which they can poll on demand. An agent may subscribe to any number of pre-existing channels and may also create and destroy channels at will. Messages can be sent to any channels.

Message-passing is intended as a lightweight mechanism for synchronizing behaviour between multiple agents as well as for representing hierarchical command structures among agents. More, it is well-suited for a multicast implementation which is a very efficient and often hardware-supported communication mode (see 0). The message-passing communication model is summarized Figure 4.

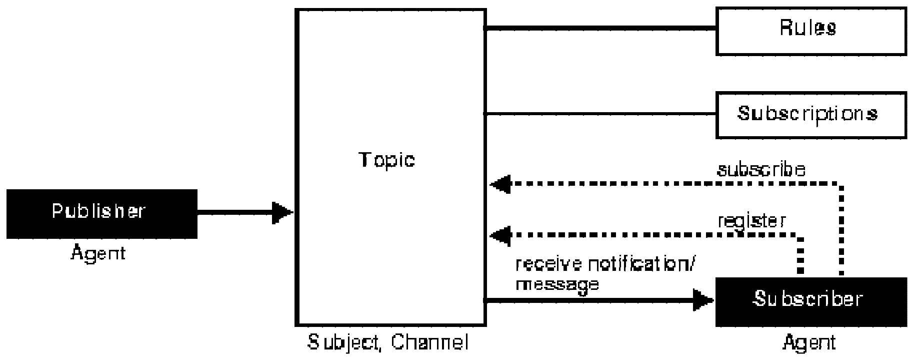


Figure 4 : The publish-subscribe paradigm

The **blackboard system** [Harrison-IMDAKTRTES96], by contrast, provides a **shared memory** framework that agents can use to store and exchange knowledge. In this architecture agents, called **contributors**, communicate by updating the blackboard. It is analogous to a team of experts who communicate their ideas by writing them on a blackboard [Hopgood-ISES02]. A **moderator** object determines the order in which contributors perform these updates. Agents can also create and destroy blackboards as desired. A blackboard is essentially a named collection of key-value pairs that has global visibility. Agents can post values to and read values from any key on any blackboard.

Typically, blackboards would be useful in simulations where groups of agents need to share a common situational picture, or where agents need to publish information without knowing the identity of the recipients (similarly to the message-passing model).

The shared memory model is efficient when a small number of agents are connected together, but a bottleneck occurs with a large number of agents due to access contention [Uhr-MAAIFRPS87]. This problem is avoided in the message-passing model as each agent may has its own local memory, although there is a communication overhead when passing messages between agents [Uhr-MAAIFRPS87]. In general, the message-passing model scales better than the shared memory model [Wong-MCDPS00].

Synchronisation

Communication raise the problem of **synchronisation** between agents. Indeed, in the simulated world, there might be delays that depend on quantities having **nothing** to do with the simulation model, e.g. delays encountered by messages as they travel through a network. This can lead to anomalies such as the cause appearing to happen **after** the effect.

A message passing system provides **primitives** for sending and receiving messages. These primitives may by either **synchronous** or **asynchronous** (or both). A synchronous send will not complete (will not allow the sender to proceed) until the receiving agent has received the message. An asynchronous send simply queues the message for transmission without waiting for it to be received. A synchronous receive primitive will wait until there is a message to read whereas an asynchronous receive will return immediately, either with a message or to say that no message has arrived. In synchronous message passing, send and receive are said to be **blocking** operations, while in asynchronous message passing, they are said to be **non-blocking**.

Synchronous architectures ensure causality order but are more likely subject to **deadlocks**. More, because of the autonomy of agents, MAS generally supports the non-synchronous model. Thus, consistency of the simulation is the main question about such an architecture. The FIFO queues only ensure **receive-order** (**RO**) for the messages but not **causal-order** (**CO**). Temporal anomalies can be eliminated by assigning a **time stamp** to each event or messages and ensuring that events are delivered or arranged in **time-stamp order** (**TSO**). The cause will always be assigned a smaller time stamp than any effect. This guarantees that causal relationships will be correctly reproduced by the simulation model. Following the same idea, a **priority level** can be assigned to each message in order to ensure a specific synchronisation.

Although shared memory communication is easy and efficient, synchronization is a more significant problem in this case. Indeed, it is easy to predict execution order within an agent, but not when the instructions are executed by different agents. Thus, shared memory communication requires some type of synchronization mechanism, for example to force an agent wait until another agent's change is made. Actually, it is the job of the moderator in the blackboard architecture to choose the read/write order.

12.2.6 Social Organisation

Agents can have **social** abilities that allow them to cooperate or to be competitive. Agents cooperate in order to achieve a collective goal that cannot be achieved by a single agent. A **shared mental model** or **memory** is the key to effective **teamwork** [Blickensderfer-TBTSFSMM97]. The **organisation** of the MAS can be then viewed as the **topology** of a group that allows to specialize the agents according to their abilities.

Agent models allowing collaboration are commonly based on the notions of **role**, which describes agent abilities, and **group**, **team** or **hierarchy**, which describes the organisation of the roles. A very interesting approach seems to be the **agent-group-role** (**AGR**) model of Gutknecht and Ferber [Gutknecht-MAPA00]. As presented in Figure 5, the organisation model is based on three concepts:

- **agent**: an entity that plays a role in an organisation
- **group**: a set of agents, an agent can be the member of one or more groups
- **role**: the representation of a **function** or a **service** in the group, an agent can play one ore more roles and a role may be played by one or more agents (an agent must provides the right abilities to play a specific role)

This model provides a template for conception that has been included in the MadKit MAS framework [Gutknecht-MAPA00]. The main problem in such model is resources allocation, abilities attribution and action coordination.

Negotiation is another form of collaboration used to constraint agents with opposite goals to achieve a “median” goal [Faratin-NDFAA98]. In this kind of application, agents explicitly communicate through speech acts and are highly cognitive, therefore not well-suited for INSCAPE.

Hierarchy is another example of organisation that have been explored in the MAS literature. The authority for decision making and control is concentrated in a specialized group at each level of the hierarchy. Interaction is through **vertical** communication from superior to sub-ordinate agent, and vice versa. Superior agents exercise control over resources and decision making.

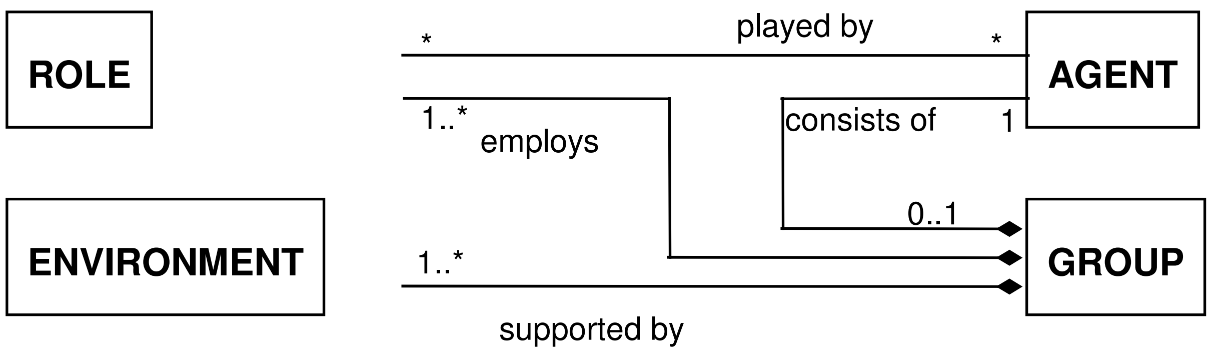


Figure 5 : Organisation of agents based on groups and roles

12.3 Making Virtual Agents Intelligent

In the first part of this section we describe the basic AI tools used to simulate behaviour. Then, in the second part, we survey the different existing strategies of behavioural based control, which seems to be the best method for designing intelligent agent in interactive stories. At last, we explorer ideas to introduce some simple organisation into MAS.

12.3.1 Fundamental AI Technologies

Finite State Machines

Essentially, a **finite state machine** (**FSM**), also called **deterministic finite automata** (**DFA**), depicts the agent's brain as a set of possible actions (**states**) and ways to change from one action to the other (**transition**). FSMs consist of a set of states (including an initial state), a set of inputs, a set of outputs, and a state transition function. The state transition function takes the input and the current state and returns a *single* new state and a set of outputs. Transition activation depends on internal agent states, but also stimuli received by its sensors.

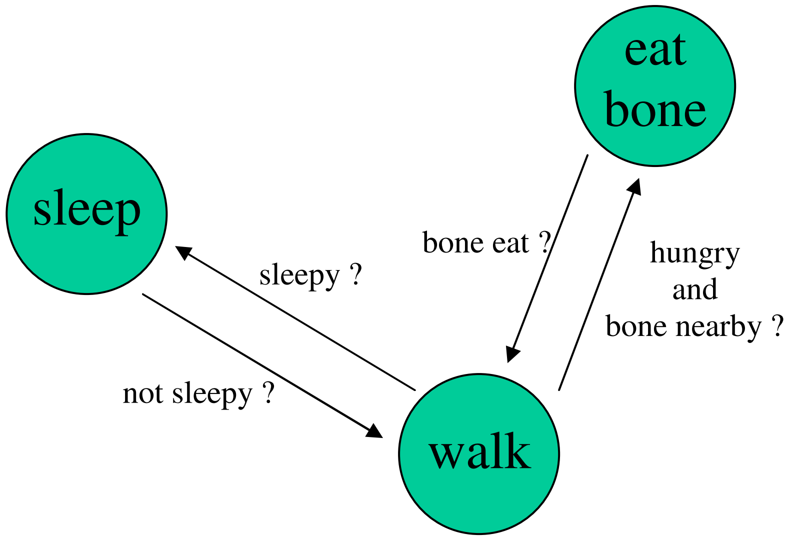


Figure 6 : Graphical representation of a virtual dog finite state machine

FSMs are intuitive to understand, easy to code, perform well, and can represent a broad range of behaviours. More, they have a simple graphical layout easier to master than complex written descriptions. For all of these reasons, they are the most popular technique used to create game AI.

However, when the modelled behaviour is complex, a classic FSM grows quickly and becomes unmanageable. Using **hierarchical FSM** (**HFSM**) or **parallel automata** is one of the most popular approach to allow greater control over complex AI systems. As the name implies, an HFSM is simply a hierarchy of FSMs. That is, each node of an HFSM may itself be an HFSM. Just like functions and procedures in a regular programming language, this provides a convenient way to make the design of an FSM more modular. An HFSM can be used as a **brick** to construct a more complex HFSM, leading to a **layered** behaviour. HFSMs have proven their capacity to produce concurrent behaviour for videos games [Koga-IDA98] or virtual humans simulation [Badler-SHCGAC93].

Rule Systems

FSMs are well suited for behaviours that are **local** (only a few outcomes are possible from a certain state) and **sequential** (tasks are carried out after other tasks depending on certain condition) in nature. **Rule systems** (**RS**) are more adapted to describe global and prioritised behaviours. A rule has the simple form: **condition → action**. The left-hand-side (**LHS**) of the rule specify the circumstances that activate the rule, while the right-hand-side (**RHS**) of the rule specify which actions to carry out if the rule is active. As in the case of FSMs, condition validity depends on internal agent states, but also stimuli received by its sensors. A rule system is made of a set of rule defining the global behaviour of the agent. Logical formalism can be used to represent complex conditions. For instance the rule system of a virtual dog could be:

(hungry) and (bone nearby) → eat it
(hungry) and (no bone nearby) → wander
(not hungry) and (sleepy) → sleep
(not hungry) and (not sleepy) → bark and walk

The execution of a RS is really straightforward. Rule conditions are tested in order and the action of the first rule that is activated is executed. This way implies a sense of priority (from top to bottom). RS are very easy to implement using **decision trees** that are direct mapping of the rule set, in priority order, to an "if-then" tree.

Introducing Randomness

Classic FSMs or RSs are deterministic in a mathematical sense of the word. This means that we can predict which of the outgoing transitions or actions will be executed if any. In practical terms, the behaviour of the AI system is totally predictable. This predictability gives tight control to the developer but is not desirable for nearly autonomous agents. A limited degree of "virtual freedom" can be given to the agent by introducing **randomness** in transitions and actions. The idea is to make action or transition selection **probabilistic**. A weight, which represent the probability of selection, is associated with each action or transition. This simple strategy has been successfully used in the **Improv** system for creating real-time behaviour-based animated actors [Perlin-ISSIAVW95].

Introducing Synchronisation

A way of combining simple behaviours into complex systems is to make use of AI **synchronisation**. Implementing such technique is just a matter of using a shared memory pool (blackboard system, see 12.2.5), which is visible to all agents and can be read and written by the different AIs. Then, the RS or FSM must be enhanced to take advantage of this shared memory.

Scripting

Scripting addresses the general problem of AI systems flexibility. Indeed, the internals of the AI are built into the application's source, making add-ons and changes tiring and troublesome. It would be better to **externalise** the AI so it could be run from separate modules written in a specific language called a **script**. Then, the AI modules could be coded by different people rather than those coding the main engine. This is a fundamental property for interactive storytelling, letting creativity of authors expresses itself.

A scripting language can symbolically represent the rules of a RS or the states and transitions of a FSM. Thus, AI system structure is **parsed** from external files, and executed in real-time **on the fly**. As an example, we can imagine the following script for our virtual dog:

```
(define rule
        (resource-found bone)
        ( hungry)
=>
        (eat bone)
)
```

Scripting also allows to somehow implement the concept of memory or state in the otherwise stateless world of rules. Indeed, a rule can set a script variable value and another rule can test whether the variable has a certain value or not. At last, as for HFSMs, **hierarchical** or **layered scripting** is possible. Through layering, an author can create complex behaviours (or scripts) from simpler behaviours (or scripts). Take the following example:

```
(define script “greeting”
        (“walk” center)
        (wait 1)
        ( “turn” camera)
        (wait 3)
        (“bow”)
)
```

In this example describing a greeting behaviour, the virtual actor first activate the “walk” script, which instructs the actor to reach the room centre. The “walk” and “greeting” scripts are actually running in parallel. Then, the actor waits one second before executing the “turn” script to look in front of the camera. Finally, the actor waits three seconds more before activating the “bow” action during which time the previous action has ended.

Even if building a scripting language from scratch was used to be common some years ago, nowadays **embedded** language is exactly the tool we need. An embedded language is designed specifically to be called from a host application, much like the way plug-ins work. They provide both the internal of a classical programming language (avoiding the task to define the syntax of the language, write a parser and an execution engine) and an API to communicate back and forth with the host application. This way on can start with a full-featured language instead of having to create one.

Many embedded languages exist, such as Python or Lua. Even regular programming languages can be embedded by using special tools such as Java and JNI. We recommend the use of Lua [Ierusalimschy-LRM03] for different reasons. First, it offers a small memory footprint and very good performances. Secondly, it has a low learning curve. At last, it is an interpreted language with dynamic typing and scripts running in a safe environment. Unsurprisingly it has been used in many video games: *Baldur’s Gate* (one of the games with the largest AI in history), *Impossible Creatures*, *Escape from Monkey Island*, *Grim Fandango*, etc.

12.3.2 Existing Behaviour Based Control Architectures

ALIVE

The first architecture presented has been developed by Blumberg [Blumberg-MLDACRTVE95] and used in the ALIVE project [Maes-ASFIAA95]. The model is inspired from ethology [Blumberg-BDELLH96] and clearly distinguish **behaviours** from **motor skills**. It consists of a 5-layered architecture for autonomous animated creature (Figure 7). The **geometry** layer provides the shapes and transforms manipulated over time for animation. The **motor skills** provide atomic motion elements which manipulate the geometry in order to produce coordinated motion. It has no knowledge of the environment or state of the agent other than that needed to execute the skill. “Walking”, “Running” are examples of motor skills. At the top rests the **behaviour system** responsible for deciding what to do given goals and sensory input. It triggers the correct motor skills to achieve the current task.

Each layer control the next one, and there are two important abstraction barrier provided by the architecture:

- one between the behaviour system and the motor skills
- one between the motor skills and geometry

Motor System

These layers of insulation, the **controller** and the **degrees of freedom** (**DOFs**), are important to making the architecture generic and extensible.

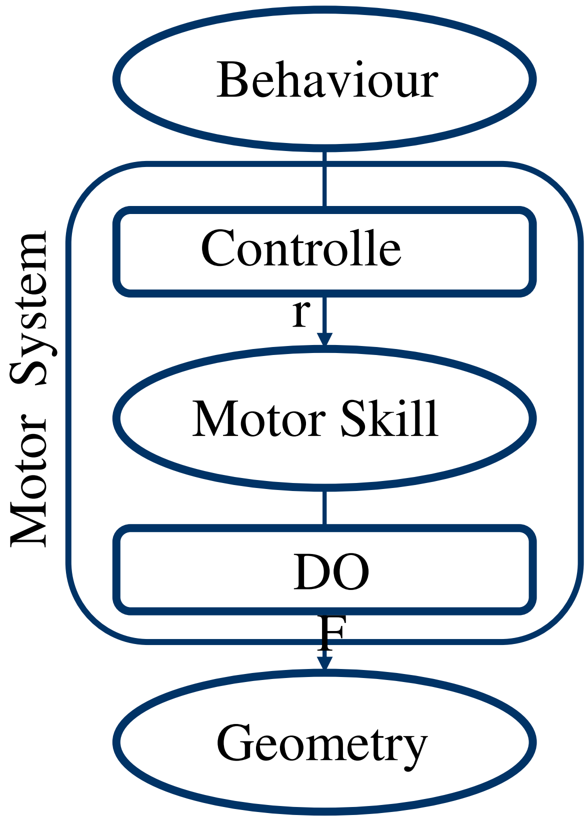


Figure 7 : Blumberg 5-layered architecture [Blumberg-MLDACRTVE95]

The controller provides the common interface to the motor skills by mapping a high-level command such as "forward" into the correct motor skill and parameters for a given agent. In this way, the same behaviour may be used by more than one type of agent.

The DOFs are "knobs" that can be used to modify the underlying geometry. They provide interpolation over the time and also resource management. In fact, each DOF can be locked by a motor skill, restricting it until unlocked. Coherent concurrent motion is then possible. As long as motor skills do not conflict for DOFs, they are free to run concurrently. Else the motor skills have to control the availability of DOF's resource and eventually wait until the desired DOFs are released.

Behaviour

The behaviour system is organized into groups of mutually inhibiting behaviour modules, which structure is presented Figure 8. A module purpose is to evaluate its appropriateness given external and internal motivations, and to issue motor command if appropriate. The **releasing mechanisms** (**RMs**) are simply filters (or detectors) which identify relevant objects or events from sensory input. By varying the allowed maximum for a given RM, a behaviour can be made more or less sensitive to the presence of a given input. Motivations and goals are simple internal variables which represents the strength of the motivation, with associated **damping** and **growth** rates. A behaviour combines the values of the RMs and internal variables and scale them by its **level of interest** used to model boredom. RMs and internal variables are shared among behaviour modules. At last, behaviours must compete with other behaviours for control of the agent. This task relies on the phenomena known as the **avalanche effect** [Minsky-SM88] that insures only one behaviour will have a non-zero value.

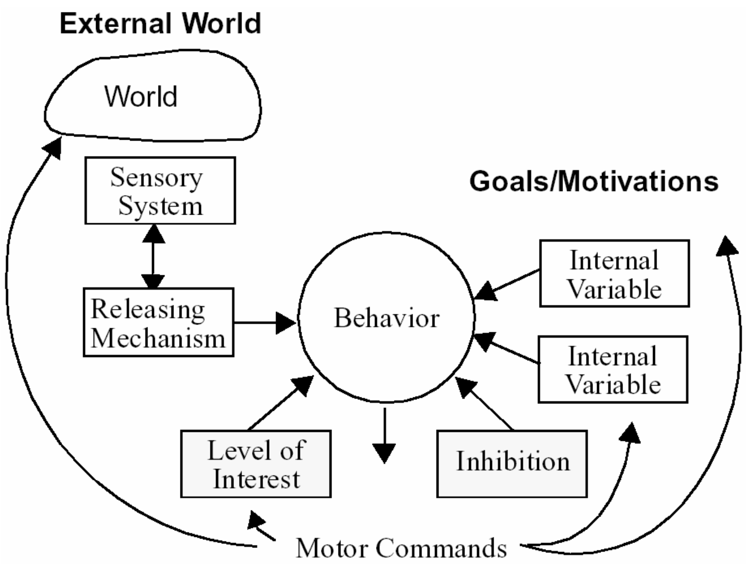


Figure 8 : The behaviour system layer in the Blumberg architecture [Blumberg-MLDACRTVE95]

IMPROV

Improv is a system for the creation of real-time behaviour-based animated actors [Perlin-ISSIAVW95]. It consists of two subsystems. The first one is an **animation engine** that uses procedural techniques to create layered, continuous, non-repetitive motions and smooth transition between them. The second one is a behaviour engine that enables to create sophisticated rules governing how actors communicate, change and make decisions. The behaviour model is similar to the one presented previously as illustrated Figure 9. In addition, it maintains the internal model of the agent, representing various aspects of an actor's moods, goals and personality.

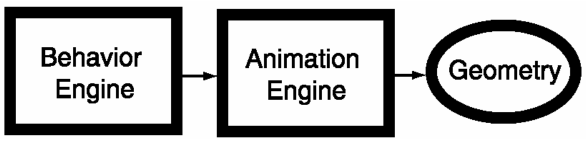


Figure 9 : The run-time architecture of the Improv system [Perlin-ISSIAVW95]

Actions

The author defines an action simply as a list of DOFs together with a range and a time varying expression. This continuous and procedural variation is done via combination of sine, cosine and coherent noise [Perlin-IS85] that allows authors to give the impression of naturalistic motions without needing to incorporate complex simulation models.

The agent can be doing many things at once, and these simultaneous activities can interact in different ways. Actually, the author can place actions in different **groups** organized into a "back-to-font" order. Actions in the same group compete. At any time, each action possesses some weight or **opacity**. When an action is selected its weight transitions smoothly from zero to one. Meanwhile, the weights of all other actions in the same group transition smoothly down to zero. Actions in groups which are further forward obscure those in groups which are further back.

In order to apply actions to the geometrical model, the run time system compute at each animation frame a weighted sum taken over the contribution of each action to each DOF within each group. The values for all DOFs in every group are then composited, proceeding from back to front. The result is a single value for each DOF, which is used to move the model.

Behaviour

The most basic tool for guiding agents' behavioural choices is a simple **parallel scripting system**. At any moment an agent executes a number of script in parallel. Like actions, scripts are organized into **groups**. Unlike actions, when a script within a group is selected, any other script that was running in the same group stops. Groups represent alternatives modes that an agent can be in some level of abstraction. For example the group of activities that an agent performs might be: resting, working, dining and conversing. The author first specifies those groups of scripts that control longer term goals (they tend to change slowly over time), then those that are most physical (they tend to choose actual actions in response to environment or internal states changes).

A script consists of a sequence of **clauses**. The two primary functions of a clause are to trigger other actions or scripts (leading to script layering) and to check, create or modify agent's properties. The choice of actions or scripts can be randomly performed, adding the more non-deterministic behaviour required for interactive non-linear applications.

At last, Improv provides a simple coordination system of multiple agents. Agents are allowed to modify each other's properties with the same freedom with which an agent can modify its own properties. The inter-agent communication occurs through the use of a shared **blackboard** (Figure 10). This way, agents are coordinate in the same manner even when running on a single processor, multiple processors or across network.

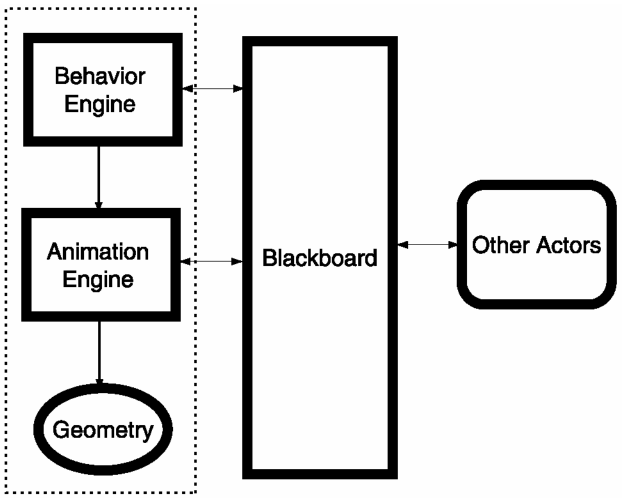


Figure 10 : Actor communication through a shared blackboard in the Improv system

Subsumption Architecture

In the subsumption architecture introduced by Brooks [Brooks-RLCSMR90], a collection of individual behavioural (reactive) modules implement the overall behaviour of the agent. Each module is an asynchronous FSM and owns a set of inputs/outputs to interact with the other components of the system (sensors, effectors, and other behaviours). Messages between components travel on pre-defined connexions.

Brooks has defined **levels of competence**, which are informal specifications of a desired class of behaviours at different abstraction levels connecting perception to action. For instance: explore the world by visiting places which look reachable, wander aimlessly around, avoid contact with objects, etc. An illustration of such architecture is given in Figure 11.

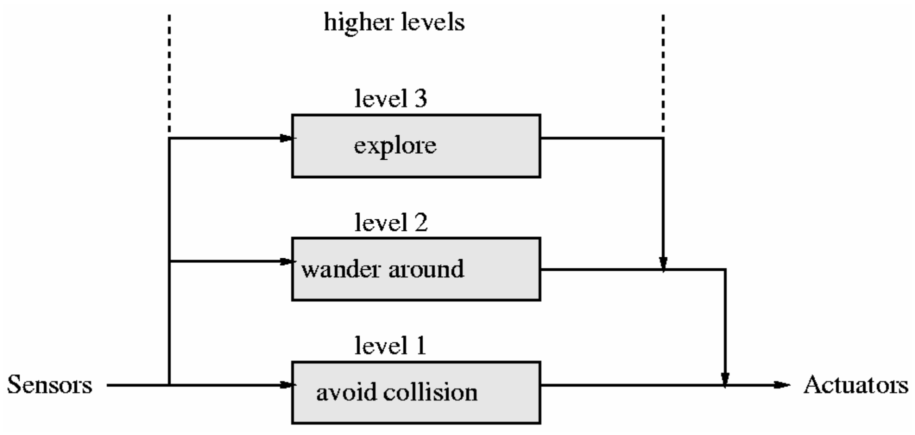


Figure 11 : Layered structure of the subsumption architecture with levels of competence

Control is **layered** with higher level layers **subsuming** the roles of lower level layers when they want to take control. Each level is able to examine data from the level below and is also allowed to send data to the level below, suppressing the normal data flow. **Inhibition** consists in blocking message diffusion on a lower behaviour output, and **suppression** consists in blocking message arrival on a lower behaviour input and sending the messages of the higher behaviour in place. The action selection relies on the hierarchical relations between the different behaviours.

The subsumption architecture has been widely and successfully used in robotics [Brooks-EPC90]. However, it is difficult to create and maintain a coherent behaviour with this approach [Arkin-BBR98]. Competence levels should be independent, but they are not in practise. Indeed, the coordination is based on operations applied to lower layers inputs/outputs. At last, it is not always possible to determine priorities between the different behaviours.

JACK

The research team of N. Badler has been working on developing autonomous virtual humans for years [Badler-TPAARPB97][Badler-RTVH99], and has build a dedicated system for this task: **JACK** (www.ugs.com/products/efactory/jack/) [Badler-SHCGAC93].

Agent architecture in JACK is a two-level structure where the motoring skills, which manipulate the geometrical representation of the agent, and the behaviour modules are clearly separated. Indeed, Badler considered **low-level** capabilities of an agent, such as being able to locomote [to], reach [for], look [at], etc. He concentrated primarily on the walking behaviour influenced by the local structure of the environment, the presence of sensed obstacles, etc. To produce such **locally-adaptive** (reactive) behaviour, reactive **sense-control-act** (**SCA**) loops are used. On the other hand, **high-level** patterns of activity and deliberation are captured in the JACK framework through **parallel state-machines** called parallel transition networks (**PaT-Nets**). PaT-Nets can sequence actions based on the current state of the environments, of the gal, or the system itself, and represent the tasks in progress, conditions to be monitored, resources used, and temporal synchronisation. An agent instantiates PaT-Nets to accomplish goals, while low-level control is mediated through direct sensing and action couplings in the SCA loop.

SCA Loops

The behavioural loop of JACK is a **continuous** stream of floating point numbers from the simulated environment. Simulated sensors map these data to the abstract results of perception and route them through control processes attempting to solve a minimization problem. This behavioural loop is actually modelled as a network of interacting SCA processes connected by arcs across which only floating point messages travel. A path from sensors to effectors is referred to as a **behavioural net**. The components of an SCA loop are:

- **sensory nodes** modelling the abstract, geometric results of object perception.
- **control nodes** modelling the lowest level influences on behaviour
- **action nodes** connecting to and executing routines defined on the underlying human body model

Sensory nodes continuously generate signals describing the relative (to the agent) polar coordinates of a particular object, or of all objects of a certain type, within a specified distance and field of view. Control nodes receive input signals from sensory nodes and send outputs to action nodes. They are formulated as explicit minimizations using outputs to drive inputs to a desired value (similar to Wilhelms' [Wilhems-NIBAC90] use of Braitenberg's behaviours [Braitenberg-VESP84]). Actions nodes arbitrate among inputs, either by selecting one set of incoming signals or averaging all incoming signals.

PaT-Nets

PaT-Nets are finite state-machines with message passing and semaphore capabilities [Becket-JLA94]. Each node is associated with processes that can invoke executable behaviours, other PaT-Nets, or specialized planners. Invocation occurs when a node is entered and transition between nodes may check a local condition evaluated within the PaT-Net or a global condition evaluated in an external environment. In order to reach more complex behaviours, arcs are prioritised and nodes also support probabilistic transitions. PaT-Nets are defined in an object-oriented structure, so running networks are created by making an instance of the PaT-Net class. More, new nets can be defined that override, blend, or extend the functionality of existing nets.

Badler showed how behavioural patterns, which much of everyday human activity falls into, are easily supported in PaT-Nets. For example, a PaT-Net representing the behavioural pattern for the well-known hide and seek game is presented Figure 12.

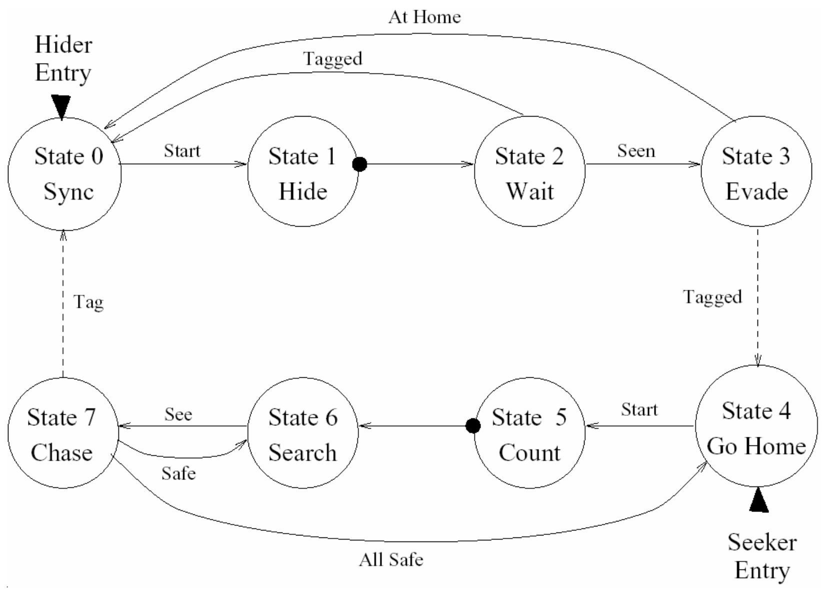


Figure 12 : An example PaT-Net for the hide and seek game [Badler-TPAARPB97]

All running PaT-Nets are embedded in a LISP operating system that **time-slices** them into the overall simulation. While running, PaT-Nets can spawn new nets, communicate with each other, kill other nets, and/or wait until a condition is met.

PAR

The main problem with the JACK system is that programming and maintaining automata is an unwieldy and complicated task. More, a little modification in the specifications of the behaviour may result in a complete reformulation of the automat. At last, even if PaT-Nets are effective programming tools, they do not represent exactly the way people conceptualise a particular situation. Thus, Badler introduced a higher-level representation to capture additional information, parameters, and aspects of human action by incorporating **natural-language** (**NL**) semantics into a **parameterised action representation** (**PAR**) [Badler-ACRTVH99]. The PAR aims at bridging the gap between natural language and animations. Indeed, natural languages often describe actions at a high level, leaving out many of the details that have to be specified for animation [Naryanam-TTWW97].

A PAR gives the description of an action specifying any relevant objects and information about the path, location, manner, and purpose. This information can be conveyed with constraints in a language: agents and objects tend to be verb arguments, path are often prepositional phrases, and manners and purposes might be in additional clauses [Palmer-CMVGST98]. In the system, a parser and translator map the components of an instruction into the parameters or variables of the PAR, which is then linked directly to PaT-Nets executing the specified movement generators in real-time. A sampling of the different parameters of a PAR is:

- **objects**: the objects used in the action. Each object knows the actions that can be performed on it and what state changes they cause (**smart object** [Kallman-BISAOIRT99]).
- **agent**: the agent executes the action. Agent are treated as special objects associated with a process, which controls its actions based on the personality and capabilities of the agent.
- **applicability conditions**: specify what needs to be true in the world in order to carry out the action.
- **preparatory conditions**: conditions to be satisfied before the action can proceed. It is actually a list of actions to be performed before the current one.
- **executing steps**: the details of executing the action after all the conditions have been satisfied. A PAR can describe either a primitive (the underlying Pat-Net is directly invoked) or complex action (a list a number of sub-actions are executed in sequence, parallel, or a combination of both).
- **manner**: describes the way which the agent carries out the action. At low-level it may result in animation modifications (slow down, speed up, etc.).
- **termination conditions**: conditions which when satisfied indicate the completion of the action.
- **post assertions**: list of statements that are executed after the termination of the action. Usually these assertions update the world database to reflect changes in the environment.

A PAR takes on two different forms: **uninstantiated** (**UPAR**) and **instantiated** (**IPAR**). A UPAR contains default applicability conditions, preparatory specifications, and execution steps, but not information about the actual agent or physical objects involved. An IPAR is a UPAR instantiated with specific information on agent, physical object, manner, termination conditions, and other bound parameters.

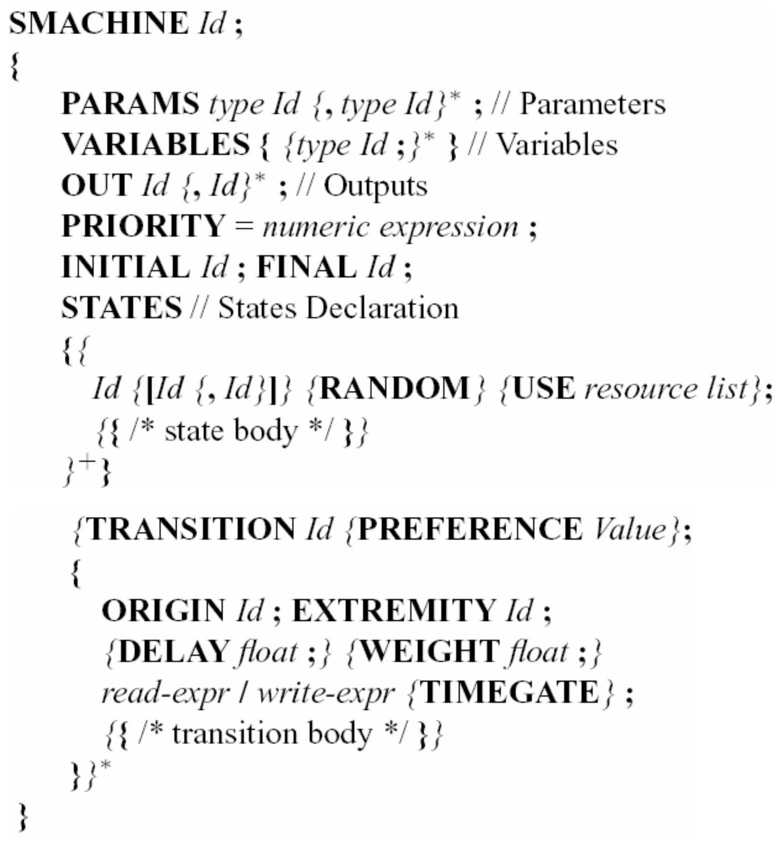
HPTS

HPTS, which stands for **hierarchical parallel transition systems**, concerns the modelling of the behavioural part of an agent [Moreau-PRTIRBS98] but is also used as an intermediate level for scenario authoring [Donikian-KSLAS99]. HPTS, like **HCSM** [Ahmad-HCSMBMSC94], is based on a **hierarchy** of **concurrent** state machines and offer a set of programming paradigms, which permit to address hierarchical concurrent behaviours. HPTS offer also the ability to manage time information, such as state frequency, delay, minimal and maximal durations.

Behaviour Description

In HPTS each agent is assimilated to a corresponding state machine. Each state machine of the system is either an atomic state machine, or a composite (hierarchical) state machine. In other words, each agent consist of sub-agents, which can be viewed as a multi-agent system in which agents are organized as a hierarchy of state machines. Hierarchical structuring of the behaviour provides the possibility of pre-empting sub-behaviours and allows to manage parallel/concurrent behaviours. HPTS also provides time and frequency handling for execution of sub-behaviours in order to model reaction times in perception activities. 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.

Donikian et al. decided to build a language for the behaviour/HFSM description. Figure 13 presents the syntax of the behavioural programming language which fully implements the HPTS formalism. The behavioural description language is not described in details here. For a complete description of the model refer to [Donikian-HBMLAA01].



```
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 13 : Syntax of the HPTS language

The body of a declaration contains a list of states and a list of transitions between these states. A state is defined by its activity with regard to data-flows. It accepts an optional duration parameter which stands for the minimum and maximum amount of time spent in the state. A state machine can be parameterised by a set of parameters used to characterize it at creation. Variables are local to a state machine and only variables that has been declared as outputs can be viewed by the meta (parent) state machine. A 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.

Starting from a state machine description, C++ code for the simulation platform GASP [Donikian-GMPDE98] is generated. Concrete state machines are instantiated from an abstract state machine class which provides pure virtual methods for running the state machines. Description can either be **compiled** or **interpreted**, which allows to modify state machines during the execution phase with an increase of only ten percent of the execution time.

Behaviour Coordination

In order to combine different behaviours, the notions of **resources** and **priority** have been included in the model [Lamarche-AOBMRP02]. A priority and a list of resources is attached to each state machine in HPTS.

Each state of a state machine can use a set of resources, which can be considered as semaphores and are used for mutual exclusion. Entering a node implies that resources are marked as taken and exiting implies that those resources are released. Using this mechanism it becomes possible to synchronize behaviours according to resources needed by them. However, as HPTS is a hierarchical model, each state machine can wait for sub-state machines ending; this synchronization creates dependencies between state machines. Thus, there are possibilities of **dead locks** if a state machine uses common resources with its sub-state machines while waiting for their ending. Thus, another constraint has been added: resources used by a state machine have to be different than resources used by its descendants.

A **priority function**, which value can be interpreted as a coefficient of *adequacy* between context and behaviour, is also associated to each state machine. Depending on its sign, this function has different meanings:

- **> 0**: the behaviour is adapted to the current context and has to be executed
- **< 0**: the behaviour is inadequate and has to be inhibited

This function consisting in a numeric expression, which allow the priority to evolve during the simulation, can be used to control the behaviour during the running phase. As it is user defined, it can be correlated with the internal state of the character (psychological parameters, intentions) or with external stimuli. It provides an easy way of control on the behaviour realization.

HTN

Cavazza described in [Cavazza-IVCIS02] a strongly character-centred interactive storytelling approach supporting anytime intervention by the spectator. His system can be viewed as a situation whereby spectators try to influence the story by "shouting" advice at the on-screen characters. Cavazza discussed the central role of artificial actors in interactive storytelling and how real-time generated behaviours participates in the creation of a dynamic storyline. He followed previous work describing behaviour through AI planning formalisms and modelled the set of possible roles for an actor as a **hierarchical task network** (**HTN**) [Nau-CSHPTP98].

Behaviour Description

HTN are networks representing (generally ordered) tasks **decomposition** (Figure 14). The top-level task of the network is the main goal of the agent and each task can be associated a set of **methods** that decompose it into **sub-tasks**. Each method includes a prescription for how to decompose the task, with various **restrictions** that must be satisfied in order for the method to be applicable. Tasks are recursively decomposed into smaller subtasks until **primitive** tasks are found that can be performed directly (such as playing an animation sequence or changing an agent state). Cavazza represents HTNs as AND/OR **graphs** and adopts total ordering of subtasks in order to preclude the possibility of interleaving subtasks from different primitive tasks, thus eliminating the classical problem of interaction to a large extend.

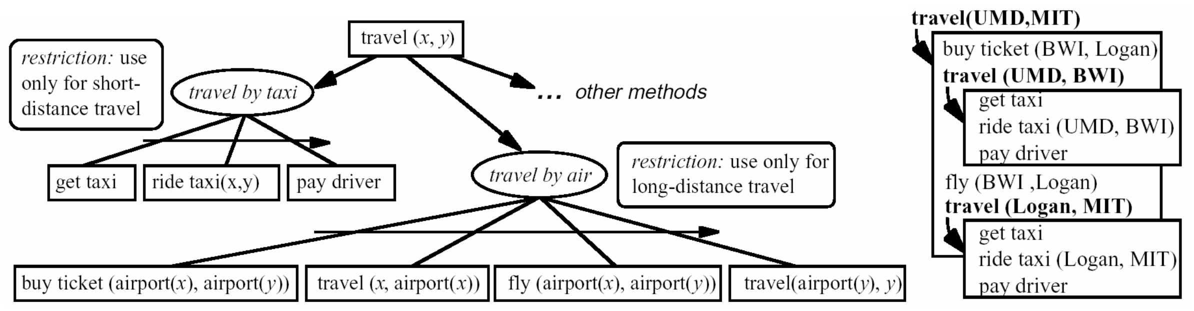


Figure 14 : An HTN representing methods for travelling and a plan based on this HTN for travelling from University of Maryland to MIT

HTN planning consists in producing a suitable plan at real-time from the description of character's roles as HTNs. The algorithm of Cavazza, as classical ones [Smith-CBBWAP98], searches the HTN **depth-first left-to-right** and attempts to execute in the virtual world any primitive action encountered. But the environment of the synthetic characters is by nature a dynamic one; it might constantly change under the influence of other agents or due to user intervention. This would call for an approach interleaving planning **and** execution, so that the action taken are constantly adapted to the current situation. Thus, **backtracking** is allowed to try an alternative decomposition if the action fail, e.g. because of the intervention of other agents or the user. Indeed, when planning and execution are interleaved, **re-planning** takes place through direct backtracking in the HTN. Actually, the interventions of the user or other agents often interfere with the executability conditions of terminal actions and the search process must be resumed to produce an alternative solution for the current node. Which makes it possible to perform reasoning about it.

In addition, **heuristic values** are attached to the various subtasks so search can make use of these values for selecting a decomposition and to **bias** search through the action space [Weyhrauch-GID97]. In the system, dynamic alteration of mood values impact on the heuristic evaluation for the nodes yet to be explored in the HTN and favour goals and activities in agreement with emotional state of the agent.

12.3.3 Social Organisation

Social organisation models are not commonly used in virtual reality systems. They have been more deeply studied in the scope of general purpose MAS [Hannoun-OMMAS00][Corkill-UMCCDPSN83] or AL [Adami-SMEPEAC98]. However, following the idea of Gutknecht and Ferber [Gutknecht-MAPA00], Chevaillier presented a simple but effective model used fore fire-fighting training [Chevaillier-HAAIVEFFT01].

The model is a more complete generic organisation model than the one of Gutknecht, and is based on UML [Booch-UMLUG99] as shown in Figure 15. In the model, the aim of the **organisation** is to **structure** the interactions between agents. It enables each agent to know its **partners** and the role he is playing in the collaboration. The concept of **role** represents the responsibilities played by the agent in the organisation and is concretely realized by a behavioural module. Agent have then an **organisational behaviour** that permits them to play or abandon a role in an organisation. This behaviour also enables agents to take into account the existence of the other members of the organisation. The model is a generic model in the sense that all classes are abstract. It is then derived to implement concrete organisations such as physical and social environments that have to be simulated in the virtual environment for training.

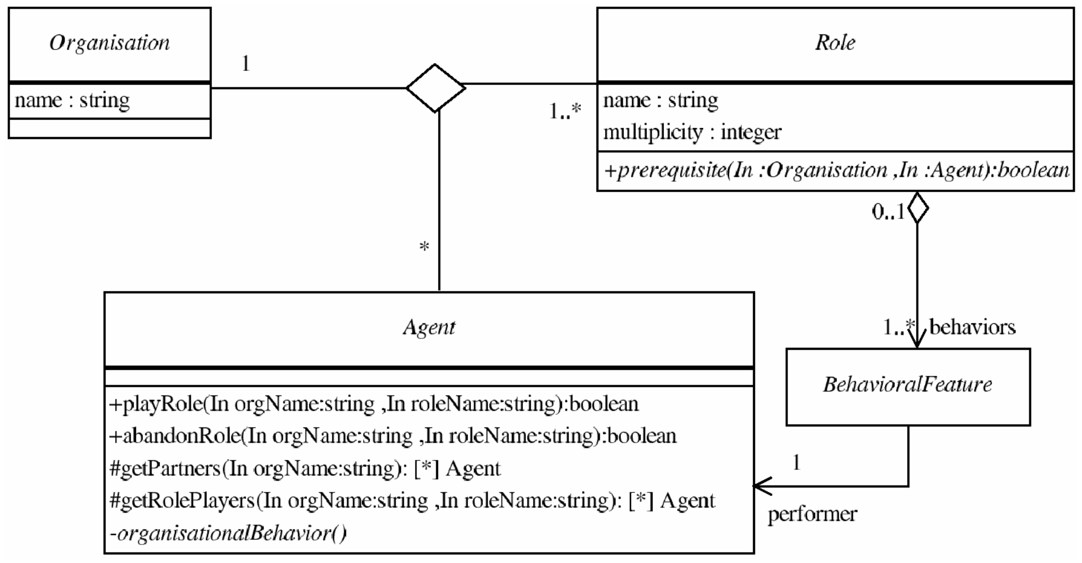

Organisation
name : string
1
1..*
Role
name : string
multiplicity : integer
+prerequisite(In :Organisation ,In :Agent):boolean
0..1
*
Agent
+playRole(In orgName:string ,In roleName:string):boolean
+abandonRole(In orgName:string ,In roleName:string):boolean
#getPartners(In orgName:string): [*] Agent
#getRolePlayers(In orgName:string ,In roleName:string): [*] Agent
-organisationalBehavior()
1..* behaviors
BehavioralFeature
1
performer

Figure 15 : The organisation model of Chevaillier [Chevaillier-HAAIVEFFT01] in UML

12.4 Building an Efficient Agent System

Interactive simulations can manage very large sets of data (3D vertices for example) and deal with thousands of entities which own complex behaviours and need to communicate. Thus, the framework architecture of the INSCAPE project should be designed to accommodate these problems. **Distributed virtual reality** (**DVR**) or **networked virtual environments** (**NVE**) have appeared recently as a way to solve complex virtual environments problems. Indeed they provide a way to **share** and a way to **communicate** based on internet network technologies.

Dimensions of complexity in distributed virtual reality mainly include:

- number of users
- number and size of worlds
- number and complexity of objects and agents in a world
- richness of agent behaviour
- richness of user interaction with the world
- richness of interaction between users or agents

The main limits on achievable complexity, against which the system competes, are:

- technological and physical limits
- financial cost

- memory
- computation
- communication

The goal of the virtual reality system is to use the resources represented by these limitations efficiently, in order to provide "sufficient" complexity in the above areas. Exactly what is considered "sufficient" may vary in different application domains, but some approaches may be expected to be better than others in most or all domains.

This section of the document is structured as follows. First we present the basic ideas to make a **scalable** system that can handle a large number of agents. Then we survey the different historical systems for DVR. Finally, we discuss new available generic paradigms to create distributed simulations. In the rest of the section, and as explained when we have defined this generic concept (12.2.1), we will often use the word "agent" either to mean user, IVA, object, or even process.

12.4.1 Scalability

Scalability generally concerns with whether the system can accommodate a large number of simultaneous agents. Scalable systems own two main characteristics:

- **joinability**: agents may be added to the system
- **maintainability**: system remains functional after various agents enter or leave it

Scalability is a key aspect to consider for real-time interaction and is a new trend of NVE elaboration as illustrated by the recent ATLAS project [Lee-ASNFDVE02]. Various approaches have been taken to create scalable systems, they generally fall into either the "increase resource" or the "reduce consumption" categories that will be presented next. But first of all, we have to make some incursion into basic network technologies to ensure a correct understanding of the underlying technical problems.

Communicating Through Networks

Communication Protocols

Nowadays, the internet is omnipresent and most communication technologies are based on the internet **protocol**. Internet is a **packet-switched**, **fault-tolerant** network. It means that information is broken down into small packets and sent from start to end point by traversing a weblike structure. The packets are sent using paths that adapt to network circumstances, errors, server malfunctions, and so on. Clearly, there are two tasks taking place at very high speeds: **data fragmentation/reassembling** and **routing**. Two protocols working in parallel perform these tasks: the **transmission control protocol** (**TCP**) and the **internet protocol** (**IP**), known as **TCP/IP**.

TCP is said to be a **connection-oriented** protocol because it keeps a permanent connection open between two or more **peers**. More, it ensure that all data sent from one end to the TCP stream will reach its destination, and in the right order (FIFO operation). However, there is a downside: TCP is slow. Another lightweight protocol exists, which sacrifices some of the "slower" features for the sake of speed, and that can be used to replace TCP: **user datagram protocol** (**UDP**). The differences between TCP and UDP are summarized Figure 16.

TCP	UDP
keeps connection	does not keep connection
variable-size packets	fixed-size packets
guarantees reception	does not guarantee reception
FIFO	Not necessarily FIFO
Slow	fast

Figure 16 : Differences between TCP and UDP

Obviously, UDP is more relevant in the case of highly dynamic simulations. This is the reason why most DVR systems have developed proprietary protocols based on UDP. Actually, TCP should be used for important or with priority transmissions that **must** be handled by the system (for example user interactions), while UDP should be used for update information that needs high refresh rate but requires less reliability.

Communication Modes

When different applications communicate through the network using either TCP or UDP, they can choose between different **modes** (Figure 17):

- **point to point**: a packet is sent to a specific computer using its address
- **broadcasting**: a packet is sent to all the computers connected to the network
- **multicasting**: a packet is sent to a **group** of computers

Point to point communication is the simplest and the most commonly used communication mode at the present time. It is available either through TCP or UDP.

Broadcasting reduces bandwidth consumption when sending the same information to all computers on the network (with regard to a point to point mode that requires to establish many connections). However, most of the computers often receive an information which is clearly not relevant for them. Thus, broadcasting was used by early DVR systems but is now superseded by the two other communication modes. Obviously, it is not globally available on the internet to avoid network overhead.

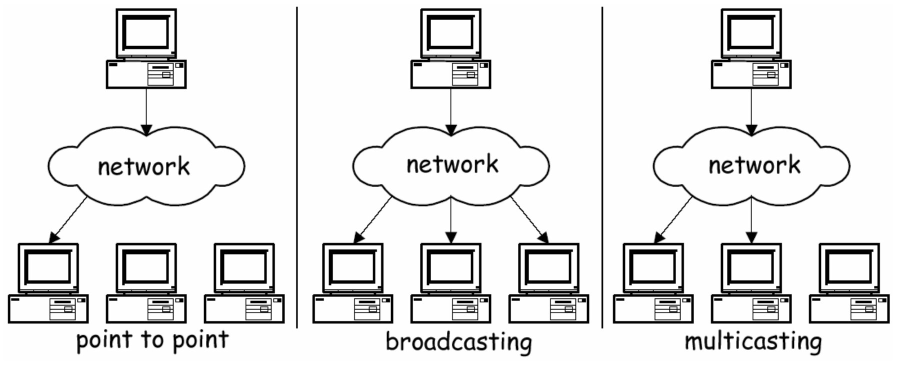


Figure 17 : The different modes of communication

Multicasting is the interesting form of communication whereby a block of information can be sent in a single operation to a set of destinations. This is contrasted with **unicast** communication (e.g. as in point-to-point connections) in which a single send operation causes the block of information to reach at most one destination (exactly one destination, if the communication is reliable and neither party fails). Multicasting can be a useful programming abstraction, but it has the additional benefit that bus-based physical communication systems can support multicasting at the **hardware** level in an inherently parallel manner. Modern routers support multicasting and the cost to the sender of multicasting can be virtually the same as a single unicast send. Indeed, the responsibility for copying and multiple forwarding of the data is assumed by the network. For this reason, multicast is currently the privileged strategy to build efficient DVR systems.

Basic Communication Architectures

Here, we present the different basic distributed architectures used by DVR systems among years. However, these days new architectures (presented in the next paragraph) have emerged to support greater scalability.

In the **client-server** architecture, the world or the simulation takes place in a **single** server. Clients send orders to the central server and receive relevant information for the user back. Only one user can act as a given time. One can see this architecture is not extensible and cannot handle a large number of users.

The **distributed client-server** architecture consists in distributing the virtual world on the different clients. Thus, only communications are managed by the central server. This way, it can be used to provide more complex management such as message filtering (see 0). However, when the world comprises many agents, the server will quickly become the **bottleneck**. In order to solve this problem, more servers can be added to the architecture. Each server is then generally specialized in handling a specific kind of messages or a specific type of agents.

In opposition to the client-server model, **peer-to-peer** architectures give the same role to each computer. The virtual world is duplicated on each machine and local modifications are communicated to other computers. Efficiency of such architectures main strongly depends on the communication mode chosen. Point to point communications are not well-suited for real-time architectures because the number of exchanges exponentially growths according to the number of computers. Therefore, multicasting is the strategy selected by most of the systems.

Enhanced Communication Architectures

Recent research has explored ways to **combine** communication architectures in order to enable more efficient information dissemination [Singhal-NVEDI99]. In other words, optimising by changing the **logical** structure of the network. Two basic structures have been particularly investigated in order to support greater scalability:

- **client-server**: enhanced into **federation** or **cluster** of servers
- **peer-to-peer**: enhanced into **peer-server** architecture

The server clusters architectures generally consists in **partitioning** clients across multiple servers. Each client send messages to its server, that server forwards the messages to its interested clients as well as other servers having clients interested in the information. The other servers then forward the information to its interested clients. This method requires that the servers themselves communicate using peer-to-peer protocols. The disadvantages include greater latency due to the exchange of information through multiple servers and greater amount of processing required due to the exchange of composite information. A more evolved strategy uses **server hierarchy**. In this case, the servers themselves act as client in a client-server relationship with higher-level servers.

The hybrid peer-server technique merges the best characteristics of the traditional peer-to-peer and client-server systems characteristics. It relies on the use of two types of servers:

- **forwarding server**: subscribes to multicast groups for agents of interest, performs aggregation and filtering functions, and forwards messages to the destination hosts
- **monitoring directory server**: collects information about the environment and dynamically determines which hosts should receive transmissions for each agent in the environment

Reachability testing determines whether a source host can communicate with a destination host.

Increase Resource

A key technique shared by all of the DRV systems is that of **distributing** computation over a number of computers or processors. This allows greater performance through parallel execution of code and more peripherals to be used. It may also make more memory available (being associated with the additional processors) except for **fully replicated** systems where the whole world is cloned on each processor. The obvious cost of distribution is in communications because networks are much slower than computer buses.

Nowadays, using **multiple servers** for multiple worlds or using server-cluster to maintain a single world has become a popular approach, especially for commercial NVEs such as Buttefly (www.butterfly.com) or Zona (www.zona.net). For example, commercial **massively multiplayer** (**MMP**) **online games** (**MMOG**) are set up with multiple servers for the same game, each serving a pre-determined number of users. When a server is full, it simply denies additional connections. However, users may not interact between servers.

Server-clusters [Funkhouser-RCSSMUVE95], on the other hand, divides the world into **zones**, and supports what appears to users as a single coherent world. This technique whereby the total space is divided up into subspaces, which are generally non-overlapping and adjacent, are also known as **adaptive spatial subdivision**. Each subspace may be handled by a separate server, or the division may be purely internal. The main problem is the appearance of “hotspot” regions to which many agents converge for some plot reason. Indeed, when lots of agents converge to a specific region, the server array must reconfigure itself automatically, subdividing the affected regions further so the overall number of agents per server stays under control. A more simple solution is to have **hot-swappable** servers that are waiting for spikes. Then, any overloaded server can pass part of the agents to the hot-swappable server. The benefits of the subdivision technique are principally an increase in computation (by parallel handling of subspaces), but also each server should be able to localise communications and limit memory requirements. The overhead is in coordinating the parallel subspaces, especially where objects in different subspaces interact.

Nevertheless, the increase resource strategy is a brute force method that relies on adding hardware to support the charge. More intelligent and algorithmic solutions, presented in the next paragraph, have been developed to really improve distributed systems **design** and **conception**.

Decrease Consumption

Area Of Interest

Messages and events are generated by agent actions or environment changes and exchanged to maintain consistency. However, if messages are sent to all other agents, the amount of transmission and processing grows at O(n^2), which is clearly not scalable. Different techniques to economize bandwidth exist, such as packet compression or aggregation [Singhal-NVEDI99], but we consider **interest management** more relevant. Indeed, real-world observation tells us that each individual only has a localized interest [Morse-IMLSDS96], resulting in a limited visibility or **sphere of interaction**. A commonly used concept is **area of interest** (**AOI**) [Funkhouser-RCSSMUVE95], which usually describes a circle, sphere or rectangular box centred on the agent. Only messages (position update, interaction messages, etc.) generated within the AOI are relevant to the user.

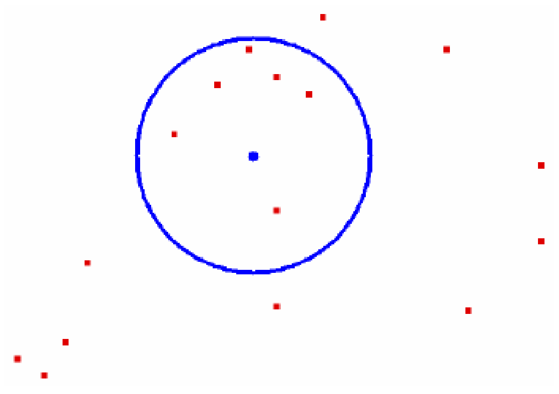


Figure 18 : Each dot represent an agent and circle represents Area Of Interest (AOI) of a particular agent

Another technique is to divide the world into **regions** [Barrus-LSLMVE96]. Each agent only receives messages from relevant regions. The challenge then is to determine the best region size. Actually, if it is larger than the real AOI of the agent, irrelevant messages are still received; while if it is smaller than AOI, it becomes inefficient to maintain. Ideally, regions would dynamically adjust size and shape based on current agent location.

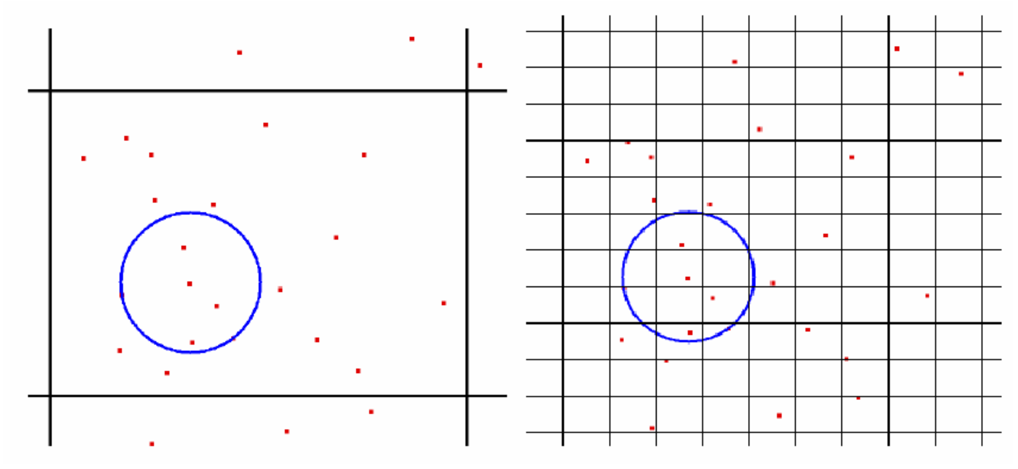


Figure 19 : The region size selection problem. Left: AOI is smaller than region, Right: AOI is larger

Interest management therefore deals with relevant information filtering to decrease unnecessary resource consumption. The best way to handle this strategy is to adopt the client-server model, where clients send messages to the server, which acts as interest manager and send back filtered messages. Network-support such as multicast can also be used to achieve this task [Macedonia-ERMG95]. More, interest management can be based on various criteria: geography (distance-based), object types or attributes (class-based), or some combination [Morse-IMLSDS96].

Explicit Interest

Following the previous idea, a different approach is to use **static** (instead of dynamic) interest. That is all agents explicitly declare their interest in other agents and events or messages. Firstly, in a distributed database, replication and distribution of database items may be minimised; this avoids the need to fully replicate the database. Secondly, in a message-passing or event-driven system explicit declaration of interest can be used to limit distribution of messages and events to only those agents which will use them.

This technique should result in reduced communications overheads, lower memory requirements and subsequently reduced computation at the receiving agents. However, the sending objects and intermediate system components will consume some additional memory and processing to support registration of interest and filtering of messages.

Level Of Detail

Level of detail (**LOD**) is a well-known technique in computer graphics that has been successfully applied to speed-up 3D rendering [Luebke-LODG02]. It relies on the fact that **perceptual limitations** are inherent to a human, and more generally, to any reactive entity (due to the physical limits of its sensors). For instance, humans cannot discern intricate details about appearance or location if an object is distant. Thus, transmitting high-resolution information to distant agents imposes unnecessary bandwidth burdens on the network and processing burdens on the receiving agents.

The idea is to exploit the LOD perception by providing at **multiple** levels of details and at **different** update rates [Singhal-NVEDI99]. Only the agents who are located near the entity of interest need to receive high-detail information. Distant agents can tolerate less detail and less information (about structure, position, orientation, etc.). Actually, each agent transmits on multiple independent data **channels**, each with a different LOD and frequency. For example, the low resolution channel can provide updates once every twenty seconds and contain only position, while the high resolution channel can provide updates every three seconds and include also information about orientation, appearance, etc. Agents **subscribe** to the appropriate channel depending on their distance from the source of information. The main problem that remains is then the choice of the most suitable number of channels depending on the application requirements.

Dead-Reckoning

The **dead-reckoning** technique involves the use of historical information about entities' attributes (for instance position) to predict future attribute values. This can be used to compensate for different frame rates in cooperating systems (e.g. smoothing the apparent movement of a remote object executing on a slow processor) - this is not strictly an aid to supporting complexity. A simple extension to this is to require the use of attribute extrapolation, and to **drop** update messages which would be predicted anyway (within some working tolerance). This technique can reduce the number of messages or events but at the expense of additional computation: the extrapolation calculations and error compensation. Following this idea, a remote prediction to behaviour in general is also possible but harder to implement.

12.4.2 Existing Solutions

Here we present a quick overview of the most known DVR systems in an historical point of view. That is we expose the capabilities of the different systems as they were presented when the systems were developed. However, some systems are still working and continue to evolve (for example MASSIVE, which has reached the third version, or NPSNET, which has reached the fifth version). We encourage the reader to consult the work of Greenhalgh for a more detailed description of each system and their advantages and drawbacks [Greenhalgh-ADVRS96][Greenhalgh-SCDVRS96].

DIVE

DIVE [Anderson-DIVE] (Distributed Interactive Virtual Environment) is a distributed multi-user VR system, developed at the Swedish Institute of Computer Science. Use of the DIVE system is covered by a license which is currently free to academic users.

DIVE is based around a fully replicated database of objects. Objects have a standard form which comprises a simple geometry plus an optional event-driven finite state machine for behaviour. More complex behaviour is achieved by manipulating the objects using specifically written C code. There can be many worlds, each identified by a simple text name. When a program joins a world it receives a complete copy of the current world state (all of the objects in the world) via TCP/IP. Changes to the world are propagated by update messages, which are reliably multicast to all processes in the world. There is also a fixed set of events which includes collision, input and interaction events, plus a limited number of simple user events. These events are all broadcast to the entire world, and each process interprets them independently. Programs can join and leave worlds independently, and can move between worlds.

dVS

dVS [Division-DUG] was a commercial virtual reality system produced by DIVISION Ltd. in the U.K.

dVS is based on a partially replicated database. There is a single world and each process which joins that world may express interest in individual database items, or in all the items of a given type in the database. The principal communication is via database operations: creating, deleting and updating database items. The database is maintained by one agent per machine, and the agents communicate to distribute database items as required. The processes cooperating in the virtual world (actors) include the renderer, I/O handler and a light-weight object context called dVISE. All objects (those in dVISE and application-specific objects) are realised using a standard set of types in the database. dVISE objects can have simple event-response behaviours.

MR Toolkit

The **MR Toolkit** [Green-MRTPM] has been developed at the University of Alberta.

MR Toolkit is a toolkit for creating virtual reality style user interfaces. Communication between processes is via shared data structures, the values of which may be copied between cooperating processes. All connections are explicit, and transfers are asynchronous and uncoordinated. The majority of the MR Toolkit is aimed at fixed configurations of cooperating processes with mutual knowledge of each others existence. Generally a single master process creates all of the other processes. The peer package provides some low-level communications facilities between such groups of processors. There is no object model, or notion of a database or world. These things would have to be implemented independently using the communications facilities provided by MR Toolkit.

AVIARY

The **AVIARY** [West-AGVRIRAVRS93] architecture and prototype implementation have been developed at the University of Manchester.

AVIARY is based on a general-purpose distributed object system with message passing for communication. Messages may be sent to single objects, lists of objects, all objects in a world or all objects in the system. In many situations, explicit interest must be expressed by one object before another will send it messages. Objects may be light-weight, executing within the context of an object server process, or heavy-weight, having their own heavyweight UNIX process. Light weight object types must currently be compiled into the object servers before execution. Object instances may be dynamically created and destroyed and can move between object servers. AVIARY supports multiple worlds, identified by a simple name. Objects can move between any worlds on the same distributed object system. Light weight objects are implemented using C in an object-oriented style, which includes inheritance. Heavy weight objects are also written in C. AVIARY includes a collision detector (with adaptive space subdivision) for each world. This is used by the renderer to limit the objects considered for rendering. The renderer has a cache volume, somewhat larger than the view cone. The renderer expresses interest in collisions between objects and this caching volume, and uses these collision events to identify objects to be rendered and to request state updates on those objects.

WAVES

The **WAVES** architecture [Kazman-OMMAS00] have been developed by Rick Kazman at the University of Waterloo and Carnegie Mellon University.

WAVES is a distributed object system, with an emphasis on the explicit representation of behaviour in order to allow object behaviour to be predicted remotely. It is also another architecture based on message passing objects. The system comprises a message manager and a number of hosts which provide execution contexts for objects (called objoids). Message passing can theoretically be limited by each host specifying a message profile, which characterises the messages it wishes to receive. This may be according to the objoid's location, or semantic constraints (e.g. particular kinds of objects). As well as the objoids being executed on a host, there is a cache of remotely located objoids which are of interest to the host. These "clones" model the behaviour of their master object, with the aim of reducing the number of coordinating messages required and enabling predictions of behaviour to compensate for network latency.

NPSNET

NPSNET [Zyda-NDFAA98], developed at the Naval Postgraduate School, was designed for military simulation with medium to large numbers of users (100s or 1000s), using standard high-end graphics workstations. It has been called a low-cost version of **SIMNET** [Blau-NVE92], being based wholly on standard Silicon Graphics workstations connected by Ethernet.

The world comprises a largely unchanging terrain over which vehicles may move. Each of these vehicles multicasts its behaviour and appearance to all other participants in the world, using the DIS protocol [ICS-MRTPM95]. Position extrapolation methods such as dead reckoning are used to calculate the expected position of other vehicles and to minimise the number of position updates to be sent (by dropping position updates which fall within some error bound of that predicted by the extrapolation method in use). The constrained behaviour available in NPSNET, together with the use of position extrapolation and multicast messages allow the system to potentially support hundreds of users on ethernet-based networks.

MASSIVE

MASSIVE [Greenhalg-EISM] (Model, Architecture and System for Spatial Interaction in Virtual Environments) has been developed at Nottingham University to investigate and assess the spatial model of interaction which is being developed by Nottingham, SICS and others [Benford-SMILVE93].

MASSIVE is based on point to point communication via connected interfaces. Interfaces include actions (RPCs), streams and attributes. An object is characterised by an **aura** interface and a peer interface. The aura interface is connected to a specific aura manager (depending on the object's world and medium) and detects collisions between object auras. Upon collision, the objects begin to communicate via their peer interfaces, which may be used to exchange medium-specific information about appearance or sound, or to pass messages. This peer-to-peer communication is controlled by mutual awareness levels, which are calculated using **focus** and **nimbus**, two other components of the spatial model. Objects may be hand-coded in C, or passive objects (e.g. scenery) can be maintained by an object server which reads object descriptions from a file. There can be many worlds, each identified by a simple name. Objects can move between worlds associated with a single master aura manager.

12.4.3 New Paradigms for Distributed Simulations

One can see many DVR systems have appeared among years. Each system was often designed for a specific use, platform, network architecture, and uses its own communication layer. The time has come where the different approaches will be unified by more generic design schemes. **HLA** and

CORBA, presented next, aims at enabling such standards. Indeed they are more suitable than any other emerging standards for distribution such as **Windows COM+** [Eddon-ICOM99], which is a component model limited to desktop application and that does not address heterogeneous distributed computing, or **SOAP** [W3C-SOAP], which is a XML-based protocol to exchange structured and typed information using HTTP/MIME with a considerable time/space overhead.

HLA

The **high level architecture** (**HLA**) [DoD-HLAIS] was developed by the Defense Modeling and Simulation Office (**DMSO**) of the Department of Defense (**DoD**) to meet the needs of defense-related projects. It is primarily developed for battlefield simulations and war simulations. But it is now increasingly being used in other application areas [Perdigau-DVSSFT03] and has also become a non-military standard through IEEE [IEEE-HLAFR].

The HLA is a standard **framework** that supports simulation composed of different simulation components. It supersedes several earlier standards such as DIS [ICS-MRTPM95] and ALSP [Weatherly-ALSP91]. Simulation systems can be entirely computer-based or involve real people. One type of simulation is called a **virtual** simulation where a real person operates simulated equipment (**human-in-the-loop**), for example a flight simulator. Another type of simulations is a **constructive** simulation where simulated people in a computer operate simulated equipment (**closed-form simulation**), for example in computer-generated forces. Yet another kind of simulation is a **live** simulation where real people operate real equipment (**hardware-in-the-loop**), for example soldiers during a military exercise connected to other simulation systems using radio equipment.

Simulation Components

HLA simulations are made up of a number of HLA **federates** and are called **federation**. There can be multiple instances of a particular type of federate. In other words, simulations that use the HLA are modular allowing federates to join and resign from the federation as the simulation executes. Federations can include more than simulations. They can also include interfaces to human operators, to real hardware and to general software performing functions such as data collection, data analysis, data display.

The **run-time infrastructure** (**RTI**) lets the participating simulation systems (federates) connect to each other and exchange information. They can communicate what objects they have and what the attribute values are, as well as exchanging interactions. The complete HLA framework is illustrated Figure 20.

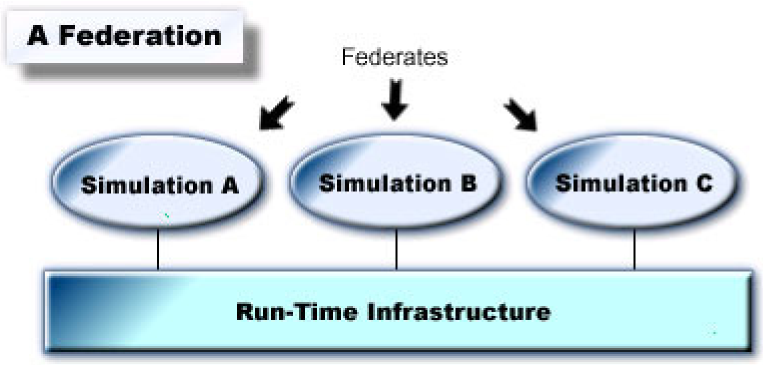


Figure 20 : Technical components of HLA

HLA distinguishes between two notions of time. The **wallclock** time is the true **global** time, typically derived from a hardware clock. Advances in wallclock time cannot be controlled by a federate. The **logical** time, what is commonly referred as **simulation time**, is the federate controlled time value (**local** time). The RTI can synchronize time within the federation. Different types of simulations handle time in various ways, so HLA supports several time management methods. Some examples are **real-time**, **scaled real-time**, **event-based** and **as-fast-as-possible**. There are also two types of simulation models that HLA is designed to handle: **continuous** (**time-stepped**) and **discrete** (**event-driven**). Actually, the HLA specifies the following four combinations of event or message transport and ordering:

- reliable/receive-ordered
- reliable/time-stamp-ordered
- best-effort/receive-ordered
- best-effort/time-stamp-ordered

Simulation API

HLA consists of a set of ten rules which must be obeyed if a federate or federation is to be regarded as HLA compliant. HLA also requires that inter-federate interactions use a standard API and defines the standard services to be used by the federates. These interfaces are arranged into six basic RTI service groups:

- **federation management** (**FM**): control an exercise (create, destroy, join, resign, save, load, pause, resume a federation)
- **declaration management** (**DM**): negotiate data exchange (**publish/subscribe** paradigm for object attributes and object interactions)
- **object management** (**OM**): communicate entity existence and characteristics (create, update, and delete objects, query or make updates, send or receive interactions)
- **ownership management** (**OSM**): share attribute ownership (distribute the right to update an object attribute among federates)
- **data distribution management** (**DDM**): route information (message filtering either based on region or interaction/class of interest)
- **time management** (**TM**): coordinate the advancement of logical time and its relationship to wallclock time during the federation execution

Discussion of all of these services is beyond the scope of this report and the interested reader is referred to [Kuhl-CCSS99] for more information on all of the HLA services.

HLA defines a two-part interface which federates are required to use for communicating with the RTI. This interface is based on the **ambassador** paradigm. A federate communicates with the RTI using its RTI ambassador. Conversely, the RTI communicates with a federate via the federate's ambassador. From the federate programmer's point of view these ambassadors are objects and the communication between the participants is performed by calling methods of these objects (Figure 21).

At last, **object model templates** (**OMTs**) provide a common method for specifying information: objects, attributes, and relationship among them. The **simulation object model** (**SOM**) identifies the object used to model real-world entities in the simulation and specifies the public attributes whose ownership may be transferred or imported. The **federation object model** (**FOM**) describes the set of objects, attributes and interactions shared across a federation. It is specified in a file read by each federate at start up.

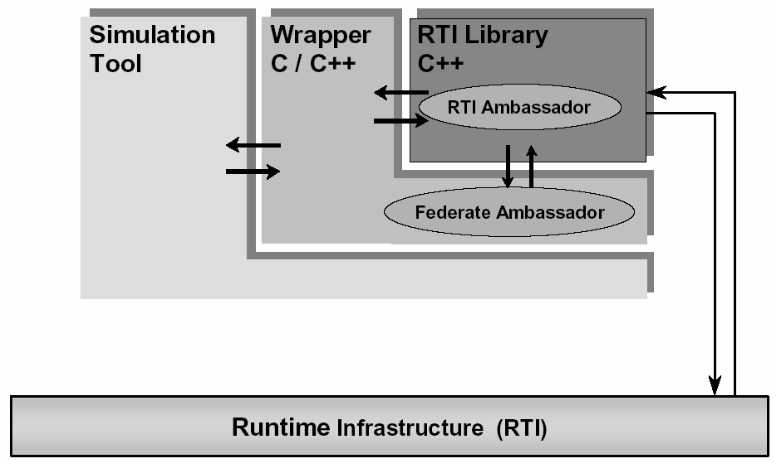


Figure 21 : Ambassador paradigm in HLA

Implementations

A reference RTI, which is now under commercialisation (**RTI-NG**), was originally freely available through DMSO. A variety of RTIs are either freely or commercially available in HLA 1.3 and IEEE 1516 versions. Some of these RTIs are fully asynchronous while others are partially asynchronous (requiring periodic calling of a "tick" method to allow the RTI to perform operations). The most powerful commercial RTIs are certainly the **MÄK High Performance RTI** (www.mak.com) and the **pRTI** (www.pitch.se). An interesting free RTI has also been developed by the ONERA but it is not yet fully implemented [Siron-DIHRPO98]. Some other RTIs are build on top of Real-Time CORBA (see next paragraph).

CORBA

As a MAS is fundamentally a **distributed object systems** (**DOS**), in which agents are objects, it seems intuitive that a generic architecture of this domain can be directly usable. We present **Common Object Request Broker Architecture** (**CORBA**) [OMG-CORBAS00], which is currently the most popular distributed object architecture. It aims at creating a **middleware framework** that allows clients to invoke operations on distributed objects without concern for object location, programming language, OS platform, communication protocols and interconnects, and hardware.

Components

The Figure 22 illustrates the primary components of the architecture as standardized by the **Object Management Group** (**OMG**) :

- **Object services**: domain-independent interfaces that are used by distributed object programs. Two examples of object services that fulfil this role are the **naming service**, which allows to find objects based on names, and the **trading service**, which allows to find objects based on their properties.
- **Common facilities**: interfaces oriented towards end-user applications. An example of such facility is the distributed document component facility (DDCF) that allows the presentation and interchange of objects based on a document model.
- **Domain interfaces**: interfaces oriented towards specific application domains. For example, the product data management (PDM) issued in the manufacturing domain.

- **Application interfaces**: interfaces developed specifically for a given application. Because the OMG does not develop applications (only specifications), these interfaces are not standardized yet.
- **Object request broker (ORB)**: provides a mechanism for transparently communicating client requests to target object implementations. When a client invokes an operation, the ORB is responsible for finding the object implementation, transparently activating it if necessary, delivering the request to the object, and returning any response to the caller.

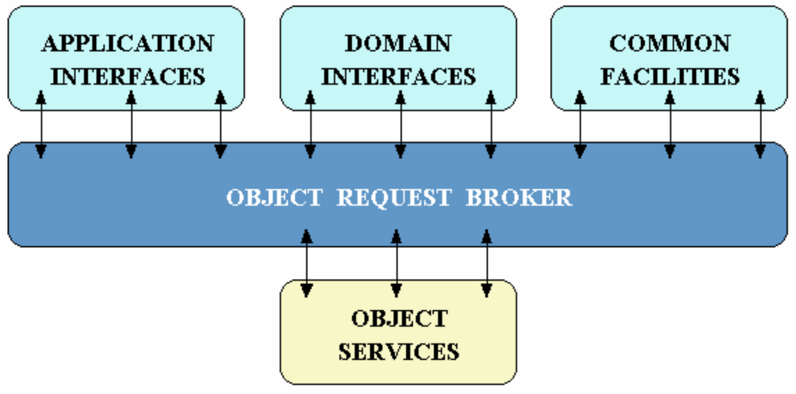


Figure 22 : OMG reference model architecture

CORBA ORB

The Figure 23 illustrates the primary components in the CORBA ORB architecture, which is the central node of the CORBA overall architecture. But first of all, we define some basic concepts of CORBA:

- **Object**: the CORBA programming entity that consists of an **identity**, an **interface** and an **implementation**, which is known as a **servant**.
- **Object reference**: a strongly-typed opaque handle that identifies an object's location
- **Object interface**: the object abstract type that defines its methods and attributes
- **Servant**: an implementation programming language entity (C, C++, Java, Smalltalk, Ada, etc.) that defines the operations supported by a CORBA object interface.
- **Client**: the program entity that invokes an operation on an object implementation.
- **Stub**: a proxy that converts method calls into messages
- **Skeleton**: an adapter that converts messages back into method calls

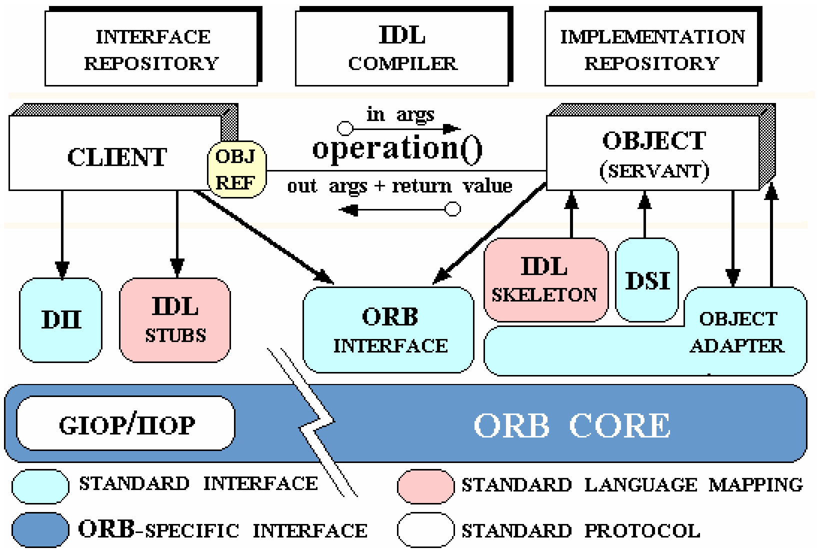


Figure 23 : CORBA ORB architecture

A client of an object accesses its reference and invokes operations on the object. A client knows only the logical structure of the object according to its interface and experiences the behaviour of the object through invocations. Definitions of the interfaces to objects can be done in two ways. Interfaces can be defined *statically* in an **interface definition language**, called the **OMG Interface Definition Language** (**OMG IDL**). This language defines the types of objects according to the operations that may be performed on them and the parameters to those operations. Alternatively, interfaces can be added to an **interface repository service** representing the components of an interface as objects and permitting run-time access to these components.

IDL is the means by which a particular object implementation tells its potential clients what operations are available and how they should be invoked. From the IDL definitions, it is possible to map CORBA objects into particular programming languages or object systems. IDL stubs and skeletons serve as the "glue" between the client and the object, respectively, and the ORB. The transformation between CORBA IDL definitions and the target programming language is automated by an **IDL compiler**. The use of a compiler reduces the potential for inconsistencies between client stubs and server skeletons and increases opportunities for automated compiler optimisations [Eide-FFOIC97].

The ORB is actually the message-passing infrastructure of CORBA. It is a logical entity that may be implemented in various ways. To decouple applications from implementation details, the CORBA specification defines an *abstract interface* for the ORB. This interface provides various helper functions such as converting object references to strings and vice versa, and creating argument lists for requests made through the **dynamic invocation interface** (**DII**) or **static invocation interface** (**SII**). In the static case, requests are controlled at compilation-time, while in the dynamic case they are generated at run-time. The server side's analogues to the client side's DII and SII are the **dynamic skeleton interface** (**DSI**) and **static skeleton interface** (**SSI**). The DSI allows the ORB to deliver requests to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. The client making the request has no idea whether the implementation is using the type-specific IDL skeletons or is using the dynamic skeletons.

Implementations

Similarly to HLA, CORBA is a specification and not an implementation. Therefore, the choice of an implementation is crucial depending on the application requirements.

The standard CORBA specifications does not require an ORB to support timed operation and uses a mapping over TCP/IP. More, conventional ORB incur significant throughput [Pyarali-DPOOFHPEMI96] and latency [Gokhale-MOCLSHSN98] overhead, as well as exhibiting many priority inversions and sources of non-determinism [Schmidt-OEASSRTA97]. As a result, it is hard to develop portable and efficient real-time applications with CORBA. To overcome these drawbacks, specific real-time CORBA specifications [OMG-RTCORBA01] and implementations have appeared. For example, TAO [Schmidt-DTRTORB98] (www.cs.wustl.edu/~schmidt/TAO.html) is a freely available, open-source, and standards-compliant real-time implementation of CORBA that provides efficient, predictable, and scalable quality of service. It seems to be the most suitable implementation for DVR systems.

12.5 Conclusion

This document has provided an overview of the different tools that need to be managed in order to create an efficient multi-agents system. First, AI techniques that make virtual agents intelligent and nearly autonomous, providing the desired level of non-linearity for INSCAPE. Secondly, network technologies and scalability considerations that make the system efficient. From this study we can try to provide a **roadmap** for the INSCAPE project and the goal we have to reach.

Because INSCAPE aims at enabling ordinary people to **interactively** author non-linear stories, it seems obvious that a simple and interactive scripting language may be used to customize or design agent's behaviours as well as story scenario **on the fly**. The language should allow the user to design behaviours according to a classical SDA cycle. In other words, from the values of the agent's attributes or its sensors' outputs, the behaviour chooses which action to perform. Thus, available attributes and actions for a given agent have to be **exported** to the scripting language whatever the mean. This language could be especially created for INSCAPE but using existing solutions is more suitable and will probably result in a more stable scripting engine, even if an effort has to be made to find the right compromise in expressiveness power. Scripting languages such as Perl, Python, or Lua provide increased flexibility and freedom in programming in the environments for which they have been created. They do this by being typeless and interpreted. We believe in that a scripting language bridges the gap between low-level behaviour programming and interactive scenario programming. Scripting is a powerful tool used to let authors designing realistic entity behaviours and has been successfully applied to many applications involving dynamic scenarios. For example, the programming environment designed to make 3D animation accessible to a large audience **Alice** provides its own scripting language based on the Python language to control and describe the movements of objects in the environment. Just as in Improv, actions in Alice are characterized by the way they control objects' degrees of freedom. Alice's scripting language provides users with constructs to create concurrent actions, as well as action sequences [Conway-AESN97]. **UnrealScript** is a scripting language based roughly on a cross between Java and C++ that allows users of the Unreal 3D game to add to the behaviours and actions experienced in the game. UnrealScript is noteworthy because it defines a distinct notion of state and can be used to build autonomous agents [Sweeney-USLR]. Thus, we recommend the use of an embedded scripting language for the INSCAPE project so is Lua.

For the underlying animation engine, **layering** seems to be the most appropriate solution. That is giving access from the behaviour to the agent at different levels in order to provide a flexible way of control. For instance, the user may control the whole body of an actor for high-level tasks such as walking and different parts of the body (arms, legs, …) for lower-level tasks such as grasping an object or simply moving the hand. Thus, actions available for a given agent must define which are their different levels. This approach is quite similar to the DOFs of the ALIVE system architecture [Blumberg-MLDACRTVE95], but with a coarser control. Indeed, we are convicted that pre-defined actions, even on low-level geometry, is a relevant abstraction for ordinary users.

We think that a simple organisation model (as the one presented in paragraph 12.3.3) helps to reduce the complexity of the MAS. Indeed, agents are less autonomous and confined to a limited set of roles, and therefore a limited set of behaviours, at a given time. In the same manner, introducing hierarchy may also be a good strategy in order to reduce complexity. More, it avoid resource attribution problems when achieving a collective goal, without the need of a time-consuming resolution. For example the superior takes all the resource it needs, then the first sub-ordinate takes all the resources it needs unless the resources previously taken by the superior, etc. Thus, we recommend to develop a **team behaviour** for the INSCAPE project based on the assignment of roles and resources to a particular group of agents. This raise the question of appropriateness between an agent and his role in a team. For instance, an adult might be used as a teacher in a classroom but not a child. We will have to investigate further in finding a mechanism to ensure behaviour consistence in a team.

As a general rule, distributed agent-based architectures can themselves be used as a means of interoperating with other simulations, it can be more useful to integrate them into a standard simulation interoperability architecture such as HLA, leveraging the benefits of both architectures. Although HLA seems to have some similarities with CORBA, HLA offers more than CORBA can do for simulations tools. HLA has integrated mechanisms for the synchronization of simulation tools regarding time and data exchange as well as intelligent data distribution mechanisms. HLA provides the main and **essential** solutions to scalability: regions of interest, routing spaces, dead-reckoning, explicit interest, etc. More, current implementations of CORBA are based on TCP, which is clearly not sufficient for some tasks of a DVR systems. But as HLA is "only" a specification, efficiency may strongly depends on the implementation chosen. Actually, the best choice is probably to implement our own subset of HLA, among the very large set of HLA services, especially dedicated for the INSCAPE simulation kernel. It should focus on managing non-linearity of the scenario and reflecting changes in the world due to dynamic behaviour of agents and story progress.

12.6 References

[Adami-SMEPEAC98] C. Adami, R. Belew, H. Kitano, and C. Taylor. Simulating multiple emergent phenomena - exemplified in an ant colony. In *Proceedings of Artificial Life VI*, 1998.

[Ahmad-HCSMBMSC94] 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*, 1994.

[Anastassakis-VASIAS01] G. Anastassakis, T. Ritchings, and T. Panayiotopoulos. Virtual agent socities with the mvital intellignet agent system. In *Proceedings of Intelligent Virtual Agents*, pages 112–125, 2001.

[Arkin-BBR98] R.C. Arkin. Behaviour-based robotics, 1998.

[Badler-ACRTVH99] N. Badler, M. Palmer, and R. Bindiganavale. Animation control for real-time virtual humans. *Communications of the ACM*, 42(8) :64–73, 1999.

[Badler-MMMCAAF91] N.I. Badler, B.L. Webber, J. Kalita, and J. Esakov. Make them move : Mechanics, control, and animation of articulated figures, 1991.

[Badler-RTVH99] N.I. Badler. Real-time virtual humans. In *International Conference on Digital Media Futures*, 1999.

[Badler-SHCGAC93] N.I. Badler, C.B. Phillips, and B.L. Webber. Simulating humans : Computer graphics animation and control, 1993.

[Badler-TPAARPB97] N.I. Badler, B.D. Reich, and B.L. Webber. Towards personalities for animated agents with reactive and planning behaviors, 1997.

[Ballet-MASDCS97] P. Ballet, V. Rodin, and J. Tisseau. A multiagent system for detecting concentric strias. In *SPIE's Optical Sciences, Engineering Instrumentation'97*, pages 659–666, 1997.

[Barrus-LSLMVE96] J. Barrus, R. Waters, and D. Anderson. Locales : Supporting large multiuser virtual environments. *IEEE Computer Graphics and Applications*, 16(6) :50–57, 1996.

[Barto-LLIAS81] A.G. Barto and R.S. Sutton. Landmark learning : an illustration of associative search. *Biological Cybernetics*, 42 :1–8, 1994.

[Becket-JLA94] W. Becket. The jack lisp api. Technical Report Technical Report MS-CIS-94-01, University of Pennsylvania, 1994.

[Benford-SMILVE93] S.D. Benford and L.E. Fahlén. A spatial model of interaction in large virtual environments. In *Third European Conference on CSCW (ECSCW'93)*, 1993.

[Blau-NVE92] B. Blau, C.E. Hugues, J.M. Moshell, and C. Lisle. Networked virtual environments. In *Computer Graphics 1992 Symposium on Interactive 3D Graphics*, 1992.

[Blickensderfer-TBTSFSMM97] E. Blickensderfer, J.A. Cannon-Bowers, and E. Salas. Theoretical bases for team self-correction : Fostering shared mental model, 1997.

[Blumberg-BDELLH96] B. Blumberg, P.M. Todd, and P. Maes. No bad dogs : Ethological lessons for learning in hamsterdam. In *Proceedings of the 4th International Conference on Simulation of Adaptive Behaviour (SAB-96)*, 1996.

[Blumberg-MLDACRTVE95] B.M. Blumberg and T.A. Galyen. Multi-level direction of autonomous creatures for real-time virtual environments. In *Computer Graphics, Proceedings SIGGRAPH'94=5*, pages 42–48, 1995.

[Booch-UMLUG99] G. Booch, J. Rumbaugh, and I. Jacobson. The unified modeling language user guide, 1999.

[Braitenberg-VESP84] V. Braitenberg. Vehicles : Experiments in synthetic psychology, 1984.

[Brooks-EPC90] R.A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6 :3–15, 1990.

[Brooks-IWR91] R.A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47 :139–159, 1991.

[Brooks-RLCSMR90] R.A. Brooks. A robust layered control system for a mobile robot. *Autonomous Mobile Robots*, pages 152–161, 1990.

[Cavazza-AIVS01] M. Cavazza, F. Charles, and S.J. Mead. Agents'interaction in virtual storytelling. In *3rd International Workshop on Intelligent Virtual Agents*, 2001.

[Cavazza-CSAAIVS01] M. Cavazza, F. Charles, and S.J. Mead. Characters in search of an author : Ai-based virtual storytelling. In *Virtual*

Storytelling : Using Virtual Reality Technologies for Storytelling, 2001.

[Cavazza-IVCIS02] M. Cavazza, F. Charles, and S.J. Mead. Interacting with virtual characters in interactive storytelling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002.

[Chankong-MODMTM83] V. Chankong and Y.Y. Haimes. Multiobjective decision making - theory and methodology, 1983.

[Chavaillier-VRMASMSIP00] P. Chavaillier, F. Harrouet, P. Reignier, and J. Tisseau. Virtual reality and multi-agent systems for manufacturing system interactive prototyping. *International Journal of Design and Innovation Research*, 2(1) :90–1001, 2000.

[Chevaillier-HAAIVEFFT01] P. Chevaillier, R. Querrec, and P. Reignier. Humans and autonomous agents interactions in a virtual environment for fire fighting training. In *Proceedings of Laval Virtual*, 2001.

[Cliff-EER93] P. Chavaillier, F. Harrouet, P. Reignier, and J. Tisseau. Explorations in evolutionary robotics. *Adaptive Behaviour*, 2(1) :73–110, 1993.

[Conway-AESN97] M.J. Conway. *Alice : Easy-to-Learn 3D Scripting for Novices*. PhD thesis, University of Virginia, 1997.

[Corkill-UMCCDPSN83] D.D. Corkill and V.R. Lesser. Use of metalevel control for coordination in a distributed problem-solving network. In *In Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, 1983.

[Division-DUG] DIVISION limited. dvise user guide.

[DoD-HLAIS] U.S. Departement of Defense. High level architecture interface specification.

[Donikian-BMLAA01] S. Donikian. Hpts : a behaviour modelling language for autonomous agent. In *Fith International Conference on Autonomous Agent*, 2001.

[Donikian-GMPDE98] S. Donikian, A. Chauffaut, R. Kulpa, and T. Duval. Gasp : from modular programming to distributed execution. In *Computer Animation'98*, 1998.

[Donikian-HBMLAA01] S. Donikian. Hpts : a behaviour modelling language for autonomous agents. In *Fifth International Conference on Autonomous Agents*, 2001.

[Donikian-KSLAS99] S. Donikian, F. Devillers, and G. Moreau. The kernel of a scenario language for animation and simulation. In *Eurographics Workshop on Animation and Simulation*, 1999.

[Drogoul-SMARCP93] A. Drogoul. *De la Simulation Multi-Agents à la Résolution Collective de Problèmes. Une étude de l'Emergence de Structures d'Organisation dans les Systèmes Multi-Agents.* PhD thesis, Université de Paris VI, 1993.

[Eddon-ICOM99] G. Eddon and H. Eddon. Inside com+, 1999.

[Eide-FFOIC97] E. Eide, K. Frei, B. Ford, J. Lepreau, and G. Lindstrom. Flick : A flexible, optimizing idl compiler. In *Proceedings of ACM SIGPLAN'97 Conference on Programming Language Design and Implementation (PLDI)*, 1997.

[Espiau-CARRPS85] B. Espiau and R. Boulic. Collision avoidance for redondants robots with proximity sensors. In *Proceedings of Thrid International Symposium of Robotics Research*, 1985.

[Faratin-NDFAA98] P. Faratin, C. Sierra, and N. Jennings. Negociation decision function for autonomous agents. *Robotics and Autonomous Systems*, 24 :159–182, 1998.

[Ferber-MASIDAI98] J. Ferber. Multi-agent systems : an introduction to distributed artificial intelligence, 1998.

[Ferber-VIC95] J. Ferber. Towards collective intelligence, 1995.

[Finin-KACL94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, 1994.

[Funkhouser-RCSSMUVE95] T.A. Funkhouser. Ring : a client-server system for multi-user virtual environments. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 85–92, 1995.

[Gokhale-MOCLSHSN98] A. Gokhale and D.C. Schmidt. Measuring and optimizing corba latency and scalability over high-speed networks. *Transactions on Computing*, 47(4), 1998.

[Green-MRTPM] M M. Green and L. White. Minimal reality toolkit version 1.3 programmer's manual.

[Greenhalg-EISM] C.M. Greenhalg. An experimental implementation of the spatial model.

[Greenhalgh-ADVRS96] C. Greenhalgh. Approaches to distributing virtual reality systems. Technical Report Technical Report NOTTCS-TR-96-5, Department of Computer Science, University of Nottingham, 1996.

[Greenhalgh-SCDVRS96] C. Greenhalgh. Supporting complexity in distributed virtual reality systems. Technical Report Technical Report NOTTCS-TR-96-6, Department of Computer Science, University of Nottingham, 1996.

[Gruber-TAPO93] T.R. Gruber. A translation approach to portable ontology. *Knowledge Acquisition*, 162(6) :199–220, 1993.

[Guillot-WNA00] A. Guillot and J.A. Meyer. From sab94 to sab2000 : What's new, animat ? In *Proceedings From Animals to Animats'00*, pages 1–10, 2000.

[Gutknecht-MAPA00] O. Gutknecht and J. Ferber. The madkit agent platform architecture. In *1st Workshop on Infrastructure for Scalable Multi-Agent Systems, Autonomous Agents 2000*, pages 48–55, 2000.

[Hannoun-OMMAS00] M. Hannoun, O. Boissier, J. Sichman, and C. Sayette. Moise : an organazitional model for multi-agent systems. In *Proceedings IBERAMIA-SBIA 2000*, pages 156–165, 2000.

[Harrison-IMDAKTRTES96] A. Harrison and P.G. Thomas. Integrating multiple and diverse abstract knowledge types in real-time embedded systems. *Knowledge-Based Systems*, 9(7) :417–434, 1996.

[Hopgood-ISES02] A.A. Hopgood. Intelligent systems for engineers and scientists, 2002.

[Horswill-SCRVNS93] I. Horswill. A simple, cheap, and robust visual navigation system. In *Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour*, pages 129–136, 1993.

[Huang-MSAGI95] Z. Huang, R. Boulic, M.N. Thalmann, and D. Thalmann. A multi-sensor approach for grasping and 3d interaction. In *Proceedings CGI'95*, 1995.

[ICS-MRTPM95] IEEE Computer Society. Standard for distributed interactive simulation communication services and profiles, 1995.

[IEEE-HLAFR] IEEE. High level architecture framework and rules.

[Ierusalimschy-LRM03] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes. Lua 5.0 reference manual. Technical Report Technical Report MCC-14/03, PUC-Rio, 2003.

[Ingrand-ARTRSC91] M. Ingrand, M. Georgeff, and A. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, pages 34–44, 1991.

[Kallman-BISAOIRT99] M. Kallmann and D. Thalmann. A behavioral interface to simulate agent-object interactions in real-time. In *Proceedings of Computer Animation*, 1999.

[Kazman-OMMAS00] R. Kazman. Making waves : On the design of architectures for low-end distributed virtual environments. In *IEEE Virtual Reality International Symposium (VRAIS'93)*, 1993.

[Kodjabachian-EDNCLGFOAAI98] J. Kodjabachian and J.A. Meyer. Evolution and development of neural controllers for locomotion, gradient-following, and obstacle-avoidance in artificial insects. *IEEE Transactions on Neural Networks*, 9 :796–812, 1998.

[Koga-IDA98] Y. Koga. On intelligent digital actors. In *Imagina'98*, 1998.

[Kuhl-CCSS99] F. Kuhl, R. Weatherly, and J. Dahmann. Creating computer simulation systems, 1999.

[Lamarche-AOBMRP02] F. Lamarche and S. Donikian. Automatic orchestration of behaviours through the management of resources and priority levels. In *Proceedings of Autonomous Agents and Multiagent Systems (AAMAS'02)*, 2002.

[Lee-ASNFDVE02] D. Lee, M. Lim, and S. Han. Atlas - a scalable network framework for distributed virtual environments. In *ACM Collaborative Virtual Environments (CVE2002)*, pages 47–54, 2002.

[Luebke-LODG02] D. Luebke, M. Reddy, J.D. Cohen, A. Varshney, B. Watson, and R. Huebner. Level of detail for 3d graphics, 2002.

[Luga-CBAIDVRS98] H. Luga, C. Panatier, P. Torguet, and Y. Duthen. Collective behaviour and adaptive interaction on a distributed virtual reality system. In *Proceedings of ROMAN'98, IEEE International Workshop on Robot and Human Communication*, 1998.

[Macedonia-ERMG95] M.R. Macedonia, M.J. Zyda, D.R. Pratt, D.P. Brutzmann, and P.T. Barham. Exploiting reality with multicast groups. *IEEE Computer Graphics and Applications*, 15(5) :38–45, 1995.

[Maes-ASFIAA95] P. Maes, T. Darell, and B. Blumberg. The alive system : Full-body interaction with autonomous agents. In *Proceedings of Computer Animation'95 Conference*, pages 11–18, 1995.

[Maes-DAA91] P. Maes. Designing autonomous agents, 1991.

[Magnenat-CMASA91] N. Magnenat-Thalmann and D. Thalmann. Complex models for animating synthetic actors. *IEEE Computers Graphics and Applications*, 11(5) :32–44, 1991.

[Mathur-PSPSGCS95] A. Mathur, R. Hall, F. Jahanian, A. Prakash, and C. Rasmussen. The publish/subscribe paradigm for scalable group collaboration systems. Technical Report Technical Report CSE-TR-270-95, University of Michigan, EECS Department, 1995.

[Menou-RTCAMLSSO01] E. Menou, V. Bonnafous, J.P. Jessel, and R. Caubet. Real-time character animation using multi-layered scripts and spacetime optimization. In *Virtual Storytelling, using Virtual Reality Technologies for Storytelling*, 2001.

[Menou-VPAVC03] E. Menou, L. Philippon, S. Sanchez, J. Duchon, and O. Balet. The v-man project : toward autonomous virtual characters. In *Virtual Storytelling, using Virtual Reality Technologies for Storytelling*, 2003.

[Meyer-FYAR94] A. Guillot and J.A. Meyer. From sab90 to sab94 : Four years of animat reseach. In *Proceedings From Animals to Animats'94*, pages 2–11, 1994.

[Meyer-SABARP91] J.A. Meyer and A. Guillot. Simulation of adaptive behavior in animats : Review and prospect. In *Proceedings from Animals to Animats'91*, pages 2–14, 1991.

[Minsky-SM88] M. Minsky. The society of mind, 1988.

[Monzani-ABAVH02] J.S. Monzani. *An Architecture for the Behavioural Animation of Virtual Humans*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2002.

[Moreau-PRTIRBS98] G. Moreau and S. Donikian. From psychological and real-time interaction requirements to behavioural simulation. In *Eurographics Workshop on Animation and Simulation*, 1998.

[Morse-IMLSDS96] K.L. Morse. Interest management in large-scale distributed simulations. Technical Report Technical Report ICS-TR-96-27, UC Irvine, 1996.

[Naryanam-TTWW97] S. Naryanam. Talking the talk is like walking the walk. In *Proceedings of the 19th Annual International Conference of the Cognitive Science Society*, 1997.

[Nau-CSHPTP98] D.S. Nau, S.J.J. Smith, and K. Erol. Control strategies in htn planning : Theory versus practise. In *Proceedings of AAAI 98*, pages 1127–1133, 1998.

[Nijholt-VSSCIA02] A. Nijholt, M. Theune, S. Faas, and D. Heylen. The virtual storyteller : Story creation by intelligent agent. 2002.

[Noser-NDASVML90] H. Noser, N. M. Thalmann, and D. Thalmann. Navigation for digital actors based on synthetic vision, memory and learning. *Computer and Graphics*, 19(1) :7–19, 1990.

[Noser-SVADA95] H. Noser and D. Thalmann. Synthetic vision and audition for digital actors. In *Proceedings EuroGraphics'95*, pages 325–336, 1995.

[OBrien-TSSA88] P. O'Brien and R. Nicol. Towards a standard for software agents. *BT Technology Journal*, 16(3) :51–59, 1988.

[OMG-CORBAS00] Object Management Group. The common object request broker : Architecture and specification, 2000.

[OMG-RTCORBA01] Object Management Group. Real-time corba, 2001.

[Palmer-CMVGST98] M. Plamer, J. Rosenzweig, and W. Schuler. Capturing motion verb generalizations with synchronous tag, 1998.

[Parenthoen-APCVWFCMW01] M. Parenthoen, J. Tisseau, P. Reignier, and F. Dory. Agent's perception and charactors in virtual worlds : put fuzzy cognitive maps to work. In *Proceedings of VIRC'01*, pages 11–18, 2001.

[Perdigau-DVSSFT03] E. Perdigau, P. Torguet, C. Sanza, and J.P. Jessel. A distributed virtual storytelling system for firefighters training. In *Virtual Storytelling, using Virtual Reality Technologies for Storytelling*, 2003.

[Perlin-BACGUPM04] Ken Perlin. Better acting in computer games : the use of procedural methods. *Computers and Graphics*, 26(1), 2002.

[Perlin-IS85] K. Perlin. An image synthetizer. In *Computer Graphics, Proceedings SIGGRAPH'85*, pages 287–293, 1985.

[Perlin-ISSIAVW95] K. Perlin and A. Goldberg. Improv : a system for scripting interactive actors in virtual worlds. In *Computer Graphics, Proceedings SIGGRAPH'96*, pages 205–216, 1996.

[Pina-OACAAFENN98] A. Pina and F. Seron. Obtaining autonomous computer animated actors using fuzzy experts systems and neural networks. In *Proceedings of the 3rd International Conference on Computer Graphics and Artificial Intelligence*, 1998.

[Pirjanian-BCMSA99] Paolo Pirjanian. Behavior coordination mechanisms - state-of-the-art. Technical Report Tech-report IRIS-99-375, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, 1999.

[Pirjanian-MOBBC99] Paolo Pirjanian. Multiple objective behaviour-based control. *Journal of Robotics and Autonomous Systems*, 1999.

[Prusinkiewicz-APD93] P. Prusinkiewicz, M.S. Hammel, and E. Mjolsness. Animation of plant development. In *Computer Graphics (SIGGRAPH'93 Proceedings)*, 1993.

[Pyarali-DPOOFHPEMI96] I. Pyarali, T. H. Harrison, and D. C. Schmidt. Design and performance of an object-oriented framework for high-performance electronic medical imaging. *USENIX Computing Systems*, 9, 1996.

[Reffye-PMFBSD88] P. Reffye, C. Edelin, J. Francon, M. Jaeger, and C. Puech. Real-time character animation using multi-layered scripts and spacetime optimization. In *Computer Graphics (SIGGRAPH'88 Proceedings)*, 1988.

[Renault-VABA90] O. Renault, N. M. Thalmann, and D. Thalmann. A vision-based approach to behavioural animation. *Journal of Visualization and Computer Animation*, 1(1) :18–21, 1990.

[Reynolds-EVBMCGM93] C.W. Reynolds. An evolved, vision-based behavioral model of coordinate group motion. In *From animals to Animats, Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior*, pages 384–392, 1993.

[Sanza-ECVECS01] C. Sanza, O. Heguy, and Y. Duthen. Evolution and co-operation of virtual entities with classifier systems. In *CAS'2001, Eurographic Workshop on Computer Animation and Simulation*, 2001.

[Schmidt-DTRTORB98] D.C. Schmidt, D.L. Levine, and S. Mungee. The design of the tao real-time object request broker. *Computer Communications*, 21(4), 1998.

[Schmidt-OEASSRTA97] D.C. Schmidt, R. Bector, D.L. Levine, S. Mungee, and G. Parulkar. An orb endsystem architecture for statically scheduled real-time applications. In *Proceedings of the Workshop on Middleware for Real-Time Systems and Services*, 1997.

[Sims-EMBC94] K. Sims. Evolving 3d morphology and behaviour by competition. *Artificial Life*, 4 :28–39, 1994.

[Singhal-NVEDI99] S. Singhal and M. Zyda. Networked virtual environments : Design and implementation, 1999.

[Siron-DIHRPO98] P. Siron. Design and implementation of a hla rti prototype at onera. In *Simulation Interoperability Workshop*, 1998.

[Smith-CBBWAP98] S.J.J. Smith, D.S. Nau, and T. Throop. Computer bridge : a big win for ai planning. *AI Magazine*, 19(2) :93–105, 1998.

[Sweeney-USLR] T. Sweeney. Unrealscript language reference.

[Thalmann-NGSAIPA96] D. Thalmann. A new generation of synthetic actors : the interactive perceptive actors. In *Proceedings Pacific Graphics'96*, pages 200–219, 1996.

[Thalmann-VSKTALVA95] D. Thalmann. Virtual sensors : a key tool for the artificial life of virtual actors. In *Proceedings Pacific Graphics'95*, pages 22–40, 1995.

[Tisseau-RVAV01] J. Tisseau. *Virtual Reality : in Virtuo Autonomy*. PhD thesis, Université de Rennes I, 2001.

[Tsuji-MRRS93] S. Tsuji and S. Li. Memorizing and representing route scenes. In *From Animals to Animats, Proceedings of the 2nd International Conference on Simulation of Adaptive Behaviour*, pages 225–232, 1993.

[Tu-AFPLPB93] X. Tu and D. Terzopoulos. Artificial fishes : Physics, locomotion, perception, behaviour. In *Computer Graphics, Proceedings SIGGRAPH'94*, pages 42–48, 1994.

[Tyrrell-CMAS96] T. Tyrrell. *Computational Mechanisms for Action Selection.* PhD thesis, University of Toronto, 1996.

[Uhr-MAAIFRPS87] L. Uhr. Multi-computer architectures for artificial intelligence : towards fast, robust, parallel systems, 1987.

[W3C-SOAP] W3C. Soap version 1.2.

[Weatherly-ALSP91] R. Weatherly, D. Seidel, and J. Weissman. Aggregate level simulation protocol. In *Proceedings of the 1991 Summer Computer Simulation Conference*, 1991.

[West-AGVRIRAVRS93] A.J. West, T.L.J. Howard, R.J. Hubbold, A.D. Murta, D.N. Snowdon, and D.A. Butler. Aviary - a generic virtual reality interface for real applications, in virtual reality systems, 1993.

[Weyhrauch-GID97] P. Weyhrauch. *Guiding Interactive Drama.* PhD thesis, Carnegie Mellon University, 1997.

[Wilhems-NIBAC90] J. Wilhems and R.A. Skinner. A notion for interactive behavioural animation control. *IEEE Computer Graphics and Applicaiton*, 10(3) :14–22, 1990.

[Wilson-KGAA85] S.W. Wilson. Knowledge growth in an artificial animal. In *Proceedings Genetic Algorithms and their Applicaitons'85*, pages 16–23, 1985.

[Wong-MCDPS00] K.L. Wong. *A Message Controller for Distributed Processing Systems.* PhD thesis, Nottingham Trent University, 2000.

[Wooldridge-MASMADAI99] M. Wooldridge. Multiagent systems : a modern approach to distributed artificial intelligence, 1999.

[Wooldridge-RRA00] M. Wooldridge. Reasoning about rational agents, 2000.

[Zicheng-KMORST95] L. Zicheng and M. F. Cohen. Keyframe motion optimization by relaxing speed and timing, 1995.

[Zyda-NDFAA98] M.J. Zyda, D.R. Pratt, J.G. Monahan, and K.P. Wilson. Npsnet : Constructing a 3d virtual world. *Computer Graphics*, 3, 1992.