An Animated Metaphor for Agent Oriented Programming

Sonia V. Rueda
Departamento de Ciencias de la Computación
UNIVERSIDAD NACIONAL DEL SUR
Bahía Blanca, ARGENTINA
e-mail: srueda@criba.edu.ar

Abstract

The term **Animated Systems** has been introduced in the bibliography in reference to interactive dynamic worlds simulations, composed of interacting independent objects [Tra96]. Simulation is a powerful tool because it allows the construction of virtual worlds that model a part of the real world. The laws of physics, the animal behavior patterns, are no longer abstract theories, and they transform into tangible realities. Through the creation, the observation and the modification of the virtual world it is possible to obtain an enhanced comprehension of the world that is being modeled.

The most flexible way to create a simulation is by programming it [Cyp95]. The environments and languages of conventional programming allow the development of virtual worlds, but they are not adequate for this task. The conception of a program as a sequence of instructions, on what the procedural model is based, requires a considerable capacity for mental contortion. Even object oriented programming, based on message passing, demands a strong level of abstraction. In particular, they are too complex for novice users.

We cannot eliminate the inherent complexity of the problem of building a virtual world, but we can search for tools that are expressive enough so the task is not complicated any further. So, the construction of dynamic worlds requires paradigms, environments and programming languages that provide a new way of thinking about programs [Cyp94].

This article proposes agent based programming as a metaphor for building worlds of interactive autonomous objects. This alternative is attractive because it is natural to build animated systems on the base of a metaphor that takes elements of live agents of the real world to build a virtual world.

1 Introduction

The computer is a universal medium that can be used to create interactive dynamic models of any mathematical, physical, biological, etc. process. The laws of physics, the animal behavior patterns can be comprehended on the base of concrete and visible simulations. The computational environments that allow this type of activity are particularly powerful tools for education because they can change the way children think and learn.

The activity of expressing dynamic ideas in an interactive medium is known as programming. Papert: "When a child learns how to program, it is the learning process that is transformed, it becomes an active and auto directed part." [Pap80]

Conventional programming requires the learning of a language. A programming language demands the use of a very rigorous and inflexible syntax that constantly distracts the child from his main task. Learning a new language just to communicate with the computer is not attractive to the vast majority of them. They love to manipulate it, have control over it, but it is not natural to them the way they have to express their wishes to it.

The working environment should favor the creative process and not burden it. Some much more simple and natural, but at the same time rich and expressive, alternative should then be

used. The graphic interfaces have resulted in this sense very attractive because they are easy to use and very intuitive.

The programming language is not the only aspect that burdens this activity. Writing a program requires a process of abstraction whose complexity depends on the problem. One alternative is to totally modify the way of thinking about programs using a metaphor that takes elements of the real world to build symbolic simulations [Rue98].

The term *metaphor* is usually used in reference to a linguistic resource destined to use the language in a figured sense. In this article when we speak of metaphor we do not refer to the literary sense but to a more ample notion developed in the Contemporary Theory of the metaphor,

Agent based programming is a programming model strongly based on the animated metaphor. The most important technique in the construction of agents consists in associating explicit goals to them. The goals make agents have a completely anthropomorphic appearance: an agent has a job to do, reacts upon the events that affect its work and is capable of detecting conflicts with the goals of other agents.

2 Agent-Based Programming

Agent based programming can be thought as a particular way of organizing the fragments that compose a program. In object oriented programming the procedures are methods that describe the behavior of the instances of a class. The procedures are grouped then by the recognition of classes. In agent based programming the procedures allow the agents to carry out actions that allow them to reach their goals. The procedures are then grouped by their goals. Both models provide a great level of modularity and encapsulation through the separation between the interface and the internal implementation.

In agent based programming the animated metaphor has a lead role. The anthropomorphic characteristics are present in the language as in the interface and the user is encouraged to think in terms of the agent's activity. These agents are not just mere "instruction followers" but are characterized by purpose, autonomy and reactivity. Finally they are animated entities.

2.1 Contemporary Theory of the Metaphor

In the classic language theory, the metaphor is associated to the literary language, in particular to poetry. In this conception a metaphor is a linguistic expression in which one or more words are used in a figured sense. That is, an expression is used to denote a "similar" concept to which it is reffering literally. So, in classic theory, metaphors are language resources and are not tied to thought.

Contemporary theory tries to find a rigorous generalization to govern metaphors. All metaphors establish a mapping between domains. Under this conception a metaphor is no longer a literary resource, it has to do with thought: a metaphor allows to tie a mental domain with another.

In contemporary theory metaphors are no longer a mechanism exclusive to literature, they are resources of common use. In its place, the term "metaphoric expression" refers to a phrase that is the concrete realization of a metaphor.

Metaphors are a central topic for the study of semantics in a natural language, but they are not part of the language but of the underlying conceptual system [Lak93]. The Contemporary Theory of Metaphors has discovered a metaphorical system that structures our current conceptual system.

When a concept is used in reference to another, it is generally not a poetic expression used as a rethoric resource, but a way of reasoning about a domain starting from a more familiar one. The understanding of a domain and a strongly structured correspondence allow to reason about a less familiar one. Many physical experiences give place to literals, while less abstract concepts like emotions, causes and purposes are expressed through metaphors.

The metaphor concept is attached to the analogy concept but they encase different meanings. They both allow to establish mappings between domains, but while in an analogy an explicit mapping is built between precisely specified domains, in a metaphor the mapping and even the domains remain implicit. An expression like: "my life is like a drifting ship" establishes an analogy, while "my life is a drifting ship" is a metaphor.

Many metaphors are so rooted to our thinking habits and our language that it is difficult to see them as metaphors. A **dead** or **transparent** metaphor is a phrase that is so conventional that it is not possible to distinguish the mapping between domains. Some authors do not consider these phrases as metaphors.

Metaphors play a key role, although a controverted one, in the discourse of science because they act as tools for building new concepts and terminology. Notwithstanding some computer science formalists oppose to the use of metaphors because they consider them a not rigorous way of thinking.

2.2 Metaphors and Computing

Our science is probably the one that most exploits the use of metaphors. Metaphors are so natural in computing that many times it is difficult to note their presence. Their use has been a permanent resource to explain new concepts in terms of others already known and to explore new domains and describe their characteristics.

The use of the term "computing" to refer to the activity carried out by a complex electronic device, is itself a metaphor. As any other, this metaphor privileges certain aspects of the domain over others, in this case the operations carried out internally.

Many terms with metaphoric roots, like "tree" or "trash collection" have taken a technical meaning so strong that the original domain is no longer in sight. This phenomenon, in which metaphorical terms are transformed into literals, exists in many fields of science, but in particular in computing.

Metaphors are a frequent resource in the formation of programmers. In the majority of the initial programming courses a program is defined as a recipe. This metaphor allows the students to comprehend the behavior of the computer in terms of a domain that is familiar to them. Another concrete example is to draw rectangles to represent variables that contain values, even further these variables allow to name memory cells. Memory, as a storage place, is also a metaphor. In any case the purpose is to describe something little tangible through something more concrete.

The use of the anthropomorphic metaphor for the description of programs is not a particularly new idea. In a greater or lesser measure it appears in object based programming, in

the actors model and even in Logo. Notwithstanding, in these models the metaphor has fundamentally a didactive role, it is used to illustrate initial concepts. The only animated part that persists in the systems developed under these models can be thought as an "instruction follower"

Agent based programming begins with the conventional anthropomorphic mapping but changes in two ways. First it establishes a mapping where the computer is no longer presented as a simple anthropomorphic entity, but as a society of interacting autonomous agents. Second, it extends the mapping between the anthropomorphic representation and the underlying computational reality to include new characteristics, such as goals and emotions based on the goals' status.

2.3 The Animated Metaphor

An anthropomorphic metaphor is a mapping that describes the activity of a machine or another inanimate object in terms of human qualities. Anthropomorphic metaphors are useful and powerful because they allow to explain the functioning of complex systems, too complex to be comprehended in mechanical or physical terms exclusively, from our knowledge of the human being and its actions.

Computing systems, hardware and software, are particularly propense to be anthropomorphized, due to their complexity and apparent autonomy. In every programming language there underlies a metaphoric model that is in some way anthropomorphic [Tra96].

An animated metaphor is a mapping even more general because it describes the activity of a machine or another inanimate object in terms of an animated creature. Animated metaphors allow to think in a system starting from its purpose and function, in a way that it is possible to understand it even without knowing its implementation. Computers erase the limit between the animated and the inanimate like few other objects can.

The animated metaphor has been exploited during years to explain the operations of the Logo language to children. Under this metaphor the computer is a little person capable of performing some specific tasks and ordering other little people to perform other tasks. Every little person is asleep until someone wakes it up so it can perform its task. When it is over it goes back to sleep. All the little people are reactive but not autonomous, the specialization is in some way a form of purpose.

The syntax of Logo approaches the natural language and puts the user in the role of the computer's instructor. In this way the animated metaphor is not only a didactic resource for the teachers to introduce the notion of procedure, but it also suggests a way to program with animated components.

In the design of graphic interfaces for users there also underlies an animated metaphor. An interface can be thought as an intermediary between the user and the system. In this focus an interface in which a character controls the environment is an anthropomorphic entity.

A goal of Artificial Intelligence is to derive computational models of human thought. The nature of this process is essentially metaphoric, but the direction of the mapping is opposite to the one we have seen up to now. In the field of distributed artificial intelligence, a system is divided into communicating concurrent components. The anthropomorphic metaphor is present in the representation of the components and in the way they communicate.

The animated metaphor has then played a transcendental role in diverse areas of computing. Notwithstanding, if our intention is to build virtual worlds that simulate a part of the real world, it is necessary to explore programming models with greater support for the animated metaphor.

2.4 Animated Systems

The concept of animation is so primitive that it is difficult to define it with precision. Animation can be thought as a basic category of the human mind, that appears in a variety of different forms in the different cognitive levels [Tra96, Min87].

The roots of animation are found based on perceptual processes, but its ramifications escape perception and affect the way we think about the actions and social behavior.

The division between animated and inanimate would seem to be universal and innate. The capacity to carry out this division is in some sense perceptual because in a first level it is based on the capacity to distinguish movement from immobility and then the autonomous movement from the induced one. Notwithstanding, in this capacity, animated and life are equivalent categories.

Children have the tendency to attribute properties of the animated to things that are not. The qualities projected over the inanimate world include life and conscience. The categories of animated and life are related but are clearly different. Plants are living things but are not animated while a robot is not in the category of living things and could be animated.

The three key concepts that define the properties of an animated creature are:

- Autonomy: an animated creature is capable of initiating an action without an external cause.
- Purpose: the actions that are carried out by an animated creature are oriented to the concretion of a goal. Thus, the actions can be evaluated in terms of their effectiveness.
- Reactivity: animated creatures modify their actions to adapt to the changes in the environment.

These properties are tied in concordance to a complex structure inside which these concepts confront themselves. The most controverted relation is given between autonomy and reactivity: as a creature reacts in function of the environment it loses autonomy.

An animated system is a virtual world formed by multiple dynamic and interactive objects. The objects of an animated system must be:

- Tangible, that is, concrete from the senses, and directly manipulable.
- Reactive, they react on their own motivated by the user or other objects.
- Incrementable, they must be modifiable to explore new possibilities.
- Flexible, the system must allow the new abilities to be built from the older ones.

A symbolic simulation, of the type we have named as valuable for the constructivist point of view, is an animated system.

3 Agent Properties

The idea of agents is not new. Many authors have studied problems involving the agent concept. It should then not be difficult to define an agent. Notwithstanding, the term agent, as has occured with many others, has been used to refer to or describe many related concepts but frequently different. The situation is further complicated if we consider the word agent related to some other term, for example intelligent agents, network agents, software agents, etc.

In an interpretation that supports the animated metaphor, the agent must be characterized by the three basic properties of animation: purpose, autonomy and reactivity. This characterization must be able to be represented in concrete computational terms.

In the implementation the goals of the agents must be explicit, that is they must have a concrete and accessible representation. The goals can support reactivity and conflict detection, in a way that the agent can react before events that affect its goals or interferences with other agents' goals.

In a more ample sense, an agent based system is that whose functionality is distributed among active functional modules. Every module that carries out actions to reach a purpose is considered part of the agency. In concordance to this criterion the roots of the metaphor of agents are in procedural programming. The procedures in a program can be thought as agents that communicate through calls. Notwithstanding the three basic properties of animation, now associated to the concept of agents, are not present in the former programming models.

3.1 Purpose, Goals and Conflict

In a software system each component has a specific goal, but in a complex system it can be difficult to understand the function of every module an the way they relate.

In an agent based system, the purpose of each agent should be explicit inside its computational representation. The goals are then the conceptual and computational base, giving each agent an explicit representation of its purpose.

Each agent uses the explicit goals with different purposes, in particular:

- **Control:** It is the basic use, the goals are used to control and structure the agent's activity, calling procedures when it is necessary. That is, the goals indicate an agent when it should be run because it's goal is not yet satisfied.
- **Verification:** Goals allow an agent to monitor the success or failure of its actions. An agent can create a subagent to try and reach its own goal following a specific method, if the method fails the subagent is removed and the superior agent grows in knowledge and can create another agent with the same goal but another method.
- **Conflict Detection:** When an agent satisfies its goal it can continue monitoring it to detect if the intervention of another agent provokes his goal to become unsatisfied again.
- **Organization:** Agent based programming can be thought as a way of organizing the fragments that compose a program. In the way that object oriented programming organizes the procedures in function of the manipulated objects, agent based programming organizes them in function to the goals they try to reach. Both modularization alternatives offer a high level of encapsulation, that is, they allow to separate the external interface from the internal implementation. In object oriented programming, an object communicates

with another by sending messages, without knowing about its internal representation or the implementation of the methods with which it will attend its message. In agent based programming an agent can try to satisfy its goal and detect conflicts without knowing about the other agents' implementations.

- **Visualization:** Explicit goals allow to write easier to read programs and thereafter easier to verify and modify.

The most simple alternative for representing explicit goals is to associate textual comments to the programs. Notwithstanding, this alternative only satisfies the last purpose. A richer possibility is to associate to each agent declarative information that allows it to know when to run, when to create subagents to aid it in fulfilling it's objectives, monitor it's own actions and those of it's subordinates and detect conflicts.

Goals form the conceptual and computational base that allows to associate explicit representations to the agents and contribute to the realization of the other two principle qualities of animation: autonomy and reactivity.

3.2 Autonomy

In the standard procedural model the procedures are only run when they are explicitly called. On the other hand, agents are autonomous, this is to say they are capable of initiating actions on their own, without external calls.

It is clear that this autonomy is in last instance apparent and that at last the running of an agent will respond to an external condition. In this way the actions of an agent can be seen in to levels: **causal** and **autonomous**.

The most simple alternative to reach autonomy is based on concurrence. Each agent is a concurrent process that runs a simple loop. Concurrence is the minimum mechanism to support autonomy but it requires some additional facilities for following goals and detecting conflicts.

Another alternative is to allow agents to go off when an event is produced or a specific state is reached. This alternative is more efficient because the constant checking that the agent performs on the environment to decide if it should carry out an action is avoided. Notwithstanding, this causality compromises autonomy because the agent is activated by an external force.

3.3 Reactivity

Agents operate on a changing world, in a way that they must be capable of sensing these changes and react adequately. Reactivity is the property that distinguishes an agent from the dumb executor suggested by the "follower of instructions". Even though an agent is capable of executing instruction sequences, it should also be capable of responding to the changes that the environment suffers.

Reactivity is a property that is highly bonded with autonomy and purpose. An autonomous entity initiates an action on its own, but it must have a reason to do it. The most simple reason is a reaction to a change in the environment.

3.4 Other anthropomorphic qualities of Agents

The characterization of agents in anthropomorphic terms can be extended in various ways.

It is possible to increase the computational states to reflect emotional states. Moreover, it is possible to use these computational states to modify the facial and corporal appearance of the characters.

The activity and behavior of an agent can be reflected following different alternatives. One little exploited alternative, and without a doubt one of the most interesting open areas of agent based programming is the use of narrative techniques.

An agent system must support the possibility to reach goals through different methods. One alternative to attack this problem is that each agent governs the work of a set of subordinate agents or subagents. Subagents have the same goal as the main agent, but they try to satisfy it through different methods. The main agent activates subagents following some strategy, until the goal is satisfied. Subagents lose in some way the property of autonomy because their behavior is ultimately reactive.

An agent's actions are oriented towards the concretion of a goal. The agent should then have some form of planning for its actions that reflect its intentions, without violating the metaphor of agents, that is without contradicting the principles of reactivity and autonomy.

An agent system requires some mechanism to resolve conflicts among agents. One alternative is to use "mediators" that use the world's information and implement some form of negotiation to find a solution of compromise for each dispute.

Agents should be able to acquire experience, remembering the methods that resulted adequate to attack a problem and the circumstances under which they worked. Expert agents are capable of remembering their decisions without having to repeat actions that resulted fruitless. In particular, mediator agents should keep a record of the disputes that occured and the decisions taken to resolve them.

The challenge consists in finding an adequate representation for each situation in a way that finding coincidences results relatively easy. It is clear that the agents expertness compromises the system's simplicity.

As in a real society, an agent system requires an organization that controls its operation. Whatever the chosen alternative there will be cost to pay.

The hierarchic model is the most popular in the real world and in agent systems. The world of agents is divided in domains that are controlled by subordinate agents. The greatest challenge of the hierarchic organization is to maintain the balance between subordination and the autonomy of the subagents. A subagent with few attributions will handle little information and will transfer the majority of the decisions to its superior. Through communication even more relevant information might be lost. This problem can be attacked by increasing the level of autonomy of the subagents, the risk being the loss of control and the potential incoherence of the system.

There are other ways of organizing an agent system, for example through "groups of negotiation by consensus" These organizations are little scalable and as the system grows it is more difficult to comprehend. Hierarchy makes systems easy to understand because they can be studied in a modular form.

4 Environments for the development of agent based systems

An agent is a program, or better yet, a portion of program. The interpretation of this portion can be made in accordance to different views. As any program, it is possible to qualify an agent in virtue of its correctness, efficiency and reliability.

By nature, agent based programming emphasizes the **actions** over the **computing**. It is fundamental, then, that the agents be imbibed in an environment in which they can act upon other agents, passive or active.

An agent based programming language must respect the basic principles of understandability, "expressitivity", modularity and "composicionallity" It must be possible to represent goals, detect conflict with other agents and react in correspondence.

4.1 Agents as processes

The most simple alternative consists in implementing each agent as a concurrent process that runs an iterative block. Inside the iterative block a specific action can be executed -specialized agents- or call any procedure -general purpose agents.

Agents do not have explicit goals or a declarative aspect, nor do they communicate with each other. The condition of agent is circumscribed to carry out actions without control by the environment. Agents modify their environment, modifying the values in **slots**.

The system is completed with a driver that consists of a simple loop, in each cycle the agents are called in a random sequence -round robin- and each one is executed in a clock cycle. The concurrence is then implicitly synchronized.

This execution by turns creates impredictability problems and dependence of the execution order. Two identical agent systems can exhibit different behavior according to the order in which the agents are executed.

Example: Multilogo. In Multilogo the agents are processes that can execute any procedure. Agents are then general purpose entities. Each one of them has assigned a turtle and can communicate with the Lego creatures' sensors and effectors. Many agents can then control the same creature. Agents communicate among themselves by messages.

In Multilogo agents are autonomous but there is no explicit support for goals, reactivity and conflict detection in trying to control the same creature. The goals can be implemented through conditionals in the procedures.

Multilogo's greatest problem is that modularization is performed in two levels: processes for concurrent execution and procedures because the base language is Logo. This division by levels can result particularly complex for a child.

4.2 Simulated Concurrence

Ideally the agents should be executed concurrently, that is, the order of execution should not affect the results. One way of simulating concurrence keeping the above scheme is to postpone the sideffects produced by each agent over the slots until a complete cycle is performed.

The driver works in two phases: it computes and it refreshes. This scheme presents two inconveniences. First, if an agent modifies a slot and then it accesses its value, the modification

will not be visible. Moreover, if two agents provoke sideffects over the same slot, the final value will depend on the order in which the second faze is carried out.

The first problem is resolved by preventing the agents to access the slots they modify. The second conflict is resolved associating to each agent an "urgency" value and to each slot a list of pairs (new value, reference to the agent that produces the value). Urgency is a dynamic value that can change.

In the first phase each slot collects a list of "proposals" for new slots. In the second phase the conflicts are resolved using a function that maximizes the utility, in this case the urgency. For each slot the urgency values of the agents that propose a change are compared. It is assumed that the urgency values are different.

The resolution of conflicts can be performed by the user himself, that is, once the conflict is detected, the effect of carrying out the different proposals is shown. This alternative is not reasonable in complex systems in which the list of proposals can be very large.

4.3 Agents as rules

Another model for agents is based on rules. Each agent is associated to a rule, that is a condition or action. The agent executes the action when the condition is verified.

A rule system is composed of a set of agents and a procedure that resolves conflicts. The system is not managed in function of goals, but by the current state. The system is essentially reactive. The agents can have purposes but not explicit goals.

This alternative is more restricted but allows to obtain more efficient implementations and simplifies the resolution of conflicts.

Example: Kidsim is a rules based programming environment designed to allow children to build graphic simulations [Cyp95]. Kidsim provides a very simple interface based on the direct manipulation that allows to create and modify agents. Each agent is associated to some rules that are edited following the graphic interface principles for users (GUI). Kidsim uses demonstration programming to transform the graphic rules into conventional programs.

4.4 Agents as extended objects

In this alternative an agent is an object, of object oriented programming, extended with the internal state and its communication protocol redefined to support intention. A conventional object keeps arbitrary values and communicates with unstructured messages. The state of an extended object is defined as a mental state, it contains beliefs, in contraposition to conventional objects that can be thought as simple minds.

The role of an agent is completely representational, it maintains a belief system. There is no notion of explicit goal and there isn't autonomy because the agents are passive, therefor there isn't conflict either. Each agent is also responsible for maintaining consistency through memories. Agents communicate among themselves through a restricted set of types of messages: reports, demands, resolutions, devolutions.

Example: Shoham Agent Oriented programming formalism is a computational environment designed as an extension to object oriented programming [Sho93].

4.5 Agents as slots and value generators

In this view the actions of an agent only affect the agent itself and not the environment that surrounds it. The agents are active but their actions do not affect the world. The functional metaphor can be extended in accordance to this view.

The agents are autonomous in the sense that they permanently change their own value, but they do not have an explicit goal representation or purpose. Because there exists one agent per slot there is no possibility for conflict. An agent can be thought as a cell in a worksheet. In this case an agent is part of an object.

Example: Playground [Fen89] is an interactive environment organized from the functional model and designed as a platform for building worlds of graphic objects that execute actions. The basic unit of computing is an agent that works like a cell in a worksheet. The processing is fundamentally based on the parallel recomputation of the cells, starting from functional expressions.

4.6 Agents as behavior controllers

An intelligent action arises in response to a condition in the world. Systems are organized as modules more or less independent each one of which has associated a specific behavior. the modules are connected through a net but moreover each one has its own sensors and effectors by which they communicate with the world.

5 Conclusions

The technology in our days provides an enormous variety of forms of communication. Some of them can be used to express ideas even without being an expert. On the other hand, others require certain specialization to be used effectively.

Programming is a very powerful alternative for it's interactivity, dynamism and generality, but it requires certain dexterity, hard to acquire for the inexpert user. For years a great number of investigations and projects have been oriented to the study of paradigms, languages and environments that reduce the complexity of programming. Even though it is true that today programming is a relatively accessible activity, the difficulty level is still considerable.

In the case of children learning to program it is required, as in any other genuine learning, that they feel motivated to doing it and the best way to obtain this is to design activities that reconcile the interests of children with those if the school [Car93]. The conventional programming environments are generally little flexible for a novice user to express his ideas.

The objective of this work has been to explore new ways of thinking about animated systems programming. Of this exploration arises the metaphor of agents that captures the basic properties of an animation: purpose, autonomy and reactivity.

Instead of writing programs with some formal commands, agent based programming allows to develop them in animated terms. Children program a world inhabited by characters with goals and conflicts, starting probably with a simple model and making it grow incrementally.

References

[Tra96] Travers, M., *Programming with Agents: New metaphors for thinking about computation.* **Ph.D. Thesis**, MIT, Boston, Massachussets. 1996.

[Car93] Carretero, M., *Constructivismo y Educación.* AIQUE. 1993.

[Cyp95] Cypher, A. and D. Smith, *KidSim: End User Programming of Simulations.* CHI95. Denver, USA. 1995.

[Cyp94] Cypher, A., D. Smith and J. Spohrer, *KidSim: Programming Agents without Programming Language.* Communications of the ACM, **(37) 7**, pp 54-67, 1994.

[Lak93] Lakoff, G., *The contemporary theory of metaphor* in *Metaphor and Thought*, A. Ortony Ed., Cambridge University Press. 1993.

[Pap80] Papert, S., *Mindstorms: Children, Computers and Powerful Ideas.* Basic Books, New York.1980.

[Min87] Minsky, M., *Society of Mind.* Simon & Schuster, New York. 1987.

[Sho93] Shoham, Yoav,*Agent Oriented Programming.* Artificial Intelligence, **60(1)**, 51-92, 1993.

[Rue98] Rueda, S., *La simulacion simbolica desde una perspectiva constructivista.* CACIC'98, Neuquen, Argentina, 1998

[Fen89] Fenton, J. and K. Beck, *Playground: An Object Oriented Simulation System with Agents Rules for children of all ages.* in Proceeding of OOPSLA'89, ACM. 1989.