xx (107999.1)

CINVESTAV

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

Algoritmos Adaptativos Útiles para Generar Comportamientos en Ambientes Virtuales Dinámicos

Tesis que presenta:
FABIEL ZÚÑIGA GALLEGOS

Para obtener el grado de:
MAESTRO EN CIENCIAS

En la especialidad de:
INGENIERÍA ELÉCTRICA

Guadalajara, Jal. Octubre del 2002

CINVESTAV

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

Adaptive Algorithms Useful to Generate Behaviours in Dynamic Virtual Environments

A thesis presented by
FABIEL ZÚÑIGA GALLEGOS

To obtain the degree of
MASTER IN SCIENCE

In the subject of
ELECTRICAL ENGINEERING

Guadalajara, Jal. October, 2002

Algoritmos Adaptativos Útiles para Generar Comportamientos en Ambientes Virtuales Dinámicos

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

Fabiel Zúñiga Gallegos

Ingeniero en Computación
Universidad de Guadalajara

Becario de CONACYT, expediente No. 157945

Director de Tesis:
Dr. Félix Francisco Ramos Corchado

CINVESTAV del IPN Unidad Guadalajara, Octubre del 2002

Adaptive Algorithms Useful to Generate Behaviours in Dynamic Virtual Environments

Master of Science Thesis
In Electrical Engineering

By:

Fabiel Zúñiga Gallegos

Engineer in Computer
Universidad de Guadalajara

CONACYT no. 157945

Thesis Director:
Dr. Félix Francisco Ramos Corchado

CINVESTAV del IPN Unidad Guadalajara, October, 2002

Algoritmos Adaptativos Útiles para Generar Comportamientos en Ambientes Virtuales Dinámicos

El problema que intentamos resolver es cómo utilizar algoritmos adaptativos para agregar comportamientos adaptativos a criaturas virtuales. Este trabajo nos perimitira establecer la factibilidad de utilizar este tipo de algoritmos en problemas de animación. Esto es, al final de nuestro trabajo, queremos saber si es posible a través de un lenguaje de script y algoritmos adaptativos crear una animación con entidades virtuales.

El trabajo que proponemos consiste en tres partes:

1) Adaptar la arquitectura GedA-3D (un trabajo anterior desarrollado por nuestro equipo) para que sea útil en este trabajo. Entre otras adaptaciones, GeDA-3D soportará un editor virtual creado por un miembro de nuestro equipo, y será capaz de manejar aplicaciones móviles.
2) Implementar algunos algoritmos adaptativos en esta arquitectura. Algunos algoritmos pertenecientes a la computación evolutiva (algoritmos genéticos y estrategias evolutivas) serán implementados para agregar comportamiento adaptativo a entidades virtuales que evolucionan en un ambiente específico integrado a GeDA-3D.
3) Concluir, con nuestra experiencia, si estos algoritmos serán útiles para obtener el objetivo final descrito anteriormente.

Adaptive Algorithms Useful to Generate Behaviours in Dynamic Virtual Environments

The problem we try to solve is how to use adaptive algorithms to add adaptive behavior to virtual creatures. This work also will help us to establish the feasibility of using this sort of algorithms in animation problem. That is, at the end, we want to know if it is possible through a script language and adaptive algorithms create an animation with virtual entities.

The work we propose consists in three parts:

1) To adapt GeDA-3D architecture (a previous work developed by our team) to be useful in this work. Among others, GeDA-3D will support a virtual editor created also by a member of our team, and it will be able to manage mobile applications.
2) To implement some adaptive algorithms on this architecture. Some algorithms belonging to evolutionary computation (genetic algorithms and evolution strategies) will be implemented to add adaptive behavior to virtual entities evolving in a specific environment embedded into GeDA-3D.
3) To conclude, with our experience, if these algorithms will be useful for getting our final objective described above.

Gratitudes

I would like to express my thanks to my parents Florentino Zúñiga Arévalo and Gloria Gallegos Luna for supporting my studies. I am very grateful to my father for having forced me to continue with the high school.

I thank my wife Mariana Edith Linares Hernández for tolerating my studies.

I thank my adviser Félix Francisco Ramos Corchado and Conacyt, I was the scholar 157945.

Contents

Chapter 1

Introduction

1.1 Objective

This chapter presents the problem that this work intends to contribute, thereafter, explains the solution proposed and results expected, and finally, shows the organization of the thesis.

1.2 Problem description

The problem we try to solve is how to use adaptive algorithms to add adaptive behavior to virtual creatures. Those algorithms for instance: will allow virtual entities to take decisions about different situations such as face an enemy, in the case of a game; will help a virtual entity to plan its movements in an specific scenario to get a target, etc.

This work also will help us to establish the feasibility of using this sort of algorithms in animation problem. That is, at the end we want to know if it is possible through a script language and adaptive algorithms create an animation with virtual entities.

1.3 Solution proposed

The solution we propose consists in three parts:

- To adapt GeDA-3D [TOSCANO00] architecture to be useful in this work. Among others GeDA-3D now support a virtual editor created also by a member of our team, it is able to manage mobile applications. The resulting architecture will be described in chapter 4.
- To implement some adaptive algorithms on this architecture. Some algorithms belonging to evolutionary computation will be implemented to add adaptive behavior to virtual entities evolving in a specific environment.
- To conclude about the experiences about these algorithms for getting our final objective described in section 1.2.

1.4 Organization of the thesis

The description of our work is organized as follows:

1. Chapter 2 review briefly the following topics that constitute the theoretical base concerning the first part of our work: Concepts related to Distributed Systems and Distributed Artificial Intelligence including Mobile Agents; some technologies such as CORBA and RMI useful to develop distributed systems; some concepts to achieve the goal of dependability are briefly described; and finally Petri nets as a formal method of verification are also described.
2. Chapter 3 exhibit the topics that constitute the theoretical base for the second part of our work. Concepts related to evolutionary computation and its main paradigms: evolution strategies, evolutionary programming, genetic algorithms and genetic programming; and self-adaptation constituting an interesting particular research direction.
3. Chapter 4 introduces our GeDA-3D platform useful to integrate and manage distributed applications. Such a platform is based on a *mobile agent architecture* and has been designed specifically for the development of dynamic virtual environments. This platform is the kernel of this project and in our specific case is used to implement and prove the adaptive algorithms proposed.

4. Chapter 5 presents a formal specification of the mobility platform incorporate to GeDA-3D using an extension of Petri Nets formalism we propose, and presents a formal verification of liveness in such a system.

5. Chapter 6 presents the evolutionary algorithms we test in this work. Those algorithms are the core to implement adaptive behaviors to the virtual entities participating in a strategy game we propose as a case study. The strategy game consists of a virtual combat environment integrated to GeDA-3D.

6. Chapter 7 concludes and present our future work.

Chapter 2

Fundamentals

2.1 Objective

The objective of this chapter is to exhibit a survey of theoretical-topics useful for the realization of our research, such as the paradigms of distributed systems and multi-agent systems, and such as formal methods of verification and dependable systems.

2.2 Introduction

The core of our work lies in the multidisciplinary field of CSCW [JERRY]. In this chapter we describe the basic concepts of main theoretical-topics that were useful to reach the first of our objectives, that is, the development of GeDA-3D's platform. First, is described briefly some concepts of distributed systems, which were necessary for our research such as CORBA, RMI, etc. In second place is included a description of Petri nets because they were used as a formal method for verifying our system. In third place is given an introduction of dependable systems to understand the form which we will implement fault tolerance. And finally, we present concepts of intelligent agents, multi-agent systems and mobile agents also important concepts for our work.

2.3 Distributed systems

A Distributed System (DS) consists of a collection of autonomous computers interconnected across a network and equipped with special software designed to maintain some shared state. DS enables computers to co-ordinate their operation and to share resources of the system (hardware, software, data). In addition, DS provides users a perception of a single, integrated computing facility, hiding complexities implicit in a computers interconnection.

A Distributed System intends to address six main issues:

1. *Resource sharing:* Allows a number of hardware components and software entities to be shared in the DS.

2. *Openness:* Extends the system by adding new resource-sharing services without disruption or duplication of existing ones.

3. *Concurrency:* Since many users can invoke commands simultaneously or interact with application programs, where many server processes are running concurrently and each of them is responding to different requests from client processes, a DS should allow several processes to be executed independently or in parallel.

4. *Scalability:* Operates effectively and efficiently at many different scales, ranging from a DS consisting of 2-workstations and a file server to a WAN containing several hundred workstations and may special-purpose servers.

5. *Fault-tolerance:* Keeping the system away from incorrect results or incomplete processing when faults occur in hardware or software is a critical issue for the DS. Two approaches are considered: hardware redundancy and software recovery.

6. *Transparency:* The DS is perceived to the user and the application programmer as a whole rather than as a collection of disjoint components

Several techniques that allow the communication and synchronization of remote processes within a Distributed System are available today. Two of them are described in this work – RMI and CORBA.

2.3.1 RMI

The Java Remote Method Invocation (RMI) [RMI1] [RMI2] is an object-oriented RPC mechanism that allows one to invoke a method on an object that exists in another address space (whether it is on the same machine or on a different one). More specifically, RMI allows an object running in one Java Virtual Machine (JVM) to invoke methods of an object running in another JVM. RMI provides remote communication between programs written in the Java programming language.

2.3.2 CORBA

The **C**ommon **O**bject **R**equest **B**roker **A**rchitecture (CORBA) [OMG95a] [OMG95b] is an emerging open distributed object computing infrastructure being standardized by the Object Management Group (OMG). CORBA provides interoperability between objects in a heterogeneous, distributed environment which is in a way transparent to the programmer.

Figure 2.1 illustrates the primary components in the OMG Reference Model architecture. Descriptions of these components are available in [VINOSKI], here we only describe the object services because they are related to our work.

Object Services. Domain-independent interfaces used by many distributed object programs. For example, a service provided for the discovery of other available services is almost always necessary regardless of the application domain. Examples of Object Services include:

- Naming Service: Allows clients to find objects based on their names
- Trading Service: Allows clients to find objects based on their properties
- Lifecycle management
- Security
- Transactions
- Event notification

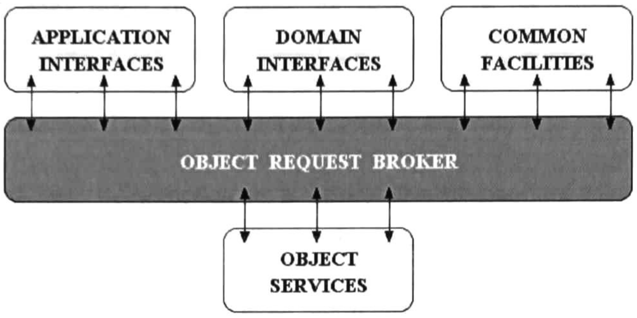


Figure 2.1 OMG Reference Model Architecture.

2.4 Dependable systems

Hazards to systems are a fact in life and so are faults. Yet we want our systems to be dependable. A system is dependable when it is trustworthy enough that reliance can be placed on the service that it delivers. For a system to be dependable, it must be available (e.g., ready for use when we need it), reliable (e.g., able to provide continuity of service while we are using it), safe (e.g., does not have a catastrophic consequence on the environment) and secure (e.g., able to preserve confidentiality) [HISSA].

Although these system attributes can be considered as non-related, in fact they are very closely interdependent. For instance, a system that is not reliable is also not available (at least when it is not operating correctly).

Achieving the goal of dependability requires effort at all phases of a system's development. Steps must be taken at design time, implementation time, and execution time, as well as during maintenance and enhancement. At design time, we can increase the dependability of a system through *fault avoidance* techniques. At implementation time, we can increase the dependability of the system through *fault removal* techniques. At execution time, *fault tolerance* and *fault evasion* techniques are required [HISSA].

2.4.1 Fault tolerance systems using replication

A component is considered faulty once its behavior is no longer consistent with its specification. There exist two representative classes of faulty behavior:

- Byzantine failures: The component can exhibit arbitrary and malicious behavior.
- Failstop failures: In response to a failure, the component changes to a state that permits other components to detect that a failure has occurred and then stops.

A system consisting of a set of distinct components is *t-fault tolerant* if it satisfies its specification provided that no more than *t* of those components becomes faulty during some interval of interest. It is important to notice that a t-fault tolerant system might continue to operate correctly even if more than t-failures occur, but correct operations cannot be guaranteed.

Replication

[COULOURIS] establish that main motivations for replication are: performance enhancement, enhanced availability and Fault tolerance.

When processors can experience Byzantine failures, an ensemble implementing a t-fault tolerant service must have at least 2t+1 replica, and the output of the ensemble is the output produced by the majority of the replicas. This is because with 2t+1 replica, the majority of the outputs remain correct even after t failures. If processors experience only failstop failures, then an ensemble containing t+1 replicas suffices, and the output of the ensemble can be the output produced by any of its members. This is because only correct outputs are produced by fail-stop processors, and after t failures one non-faulty replica will remain among the t+1 replica.

2.5 Intelligent agents

Let's start saying that Distributed Artificial Intelligence is part of the Artificial Intelligence devoted to the study about models of knowledge, communication and reasoning that entities (software and animal) we call agents need to cooperate to find the solution of a specific problem. Now, even if today there is a regulator organization called Fipa [FIPA] for agents like the OMG for objects, there is no definition of what an agent is. We do not want to get into a big discussion, so we say just that an ***agent*** is an entity (animal or software) able to do a work useful in the solution of a problem

Agents

There exist different classifications of agents, most of them in function of their complexity and their work. For instance, Brenner [BRENNER98] using the complexity and capabilities propose, a classification going from reactive agents to social agents. Goodrich [GOODRICH] classifies the agents in interface, proxies, etc. In the rest of this work we consider the following classification described by Russell [RUSSELL95] consisting in four types of agent:

1. Simple reflex agents.
2. Agents that keep track of the world.
3. Goal-based agents.
4. Utility-based agents.

In the simple reflex agent's structure, condition-action rules allow the agent to make the connection from percept to action.

The simple reflex agent will work only if the correct decision can be made on the basis of the current percept. Sometimes, the sensors do not provide access to the complete state of the world. In such cases, the agent may need to maintain some internal state information in order to distinguish between world states that generate the same perceptual input but nonetheless are significantly different. Here, "significantly different" means that different actions are appropriate in the two states. Updating this internal state information as time goes requires two kinds of knowledge to be encoded:

- Information about how the world evolves independently of the agent.
- Information about how the agent's own actions affect the world.

In the structure of the agents that keep track of the world, the current percept is combined with the old internal state to generate the update description of the current state.

Knowing about the current state of the environment is not always enough to decide what to do, together with a current state description, the agent needs some sort of goal information, which describes situations that are desirable.

Goals alone are not really enough to generate high-quality behavior. Goals just provide a crude distinction between "happy" and "unhappy" states, whereas a more general performance measure should allow a comparison of different states (or sequences of states) according to exactly how happy they would make the agent if they could be achieved. Because "happy" does not sound very scientific, the customary terminology is to say that if one world state is preferred to another, then it has higher utility for the agent.

Utility is therefore a function that maps a state onto a real number, which describes the associated degree of happiness. The overall utility-based agent structure appears in Figure 2.2. We use rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process.

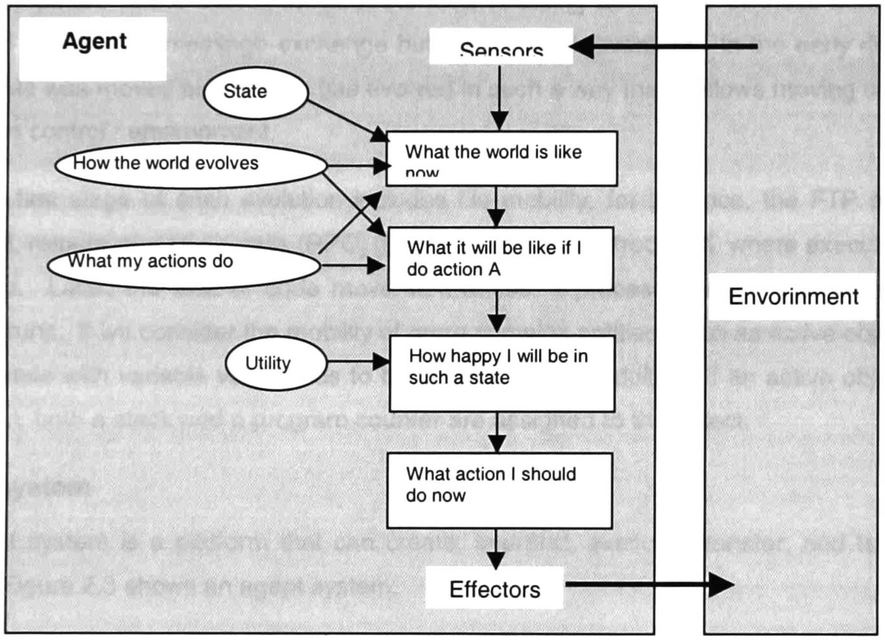

Agent
Sensors
State
How the world evolves
What the world is like now
What my actions do
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Effectors
Envorinment

Figure 2.2 A complete utility-based agent

2.6 Multi-Agent systems

Because it is quite possible that an agent be unable to solve a problem, it is necessary to create a Multiagent Systems (MAS), that is an ensamble of agents working together to get a comon objective. IAD is responsible to propose solutions to all the problems rise to make cooperate a group of agents for solving a problem.

The characteristics of MASs are that (1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (2) there is no global system control; (3) data is decentralized; and (4) computation is asynchronous.

MAS researchers develop communications languages, interaction protocols, and agent architectures that facilitate the development of agent based systems. MAS researchers draw on ideas from many disciplines outside of AI, including sociology, biology, economics, organization and management science, and even from philosophy.

2.7 Mobile agents

Since computers where interconnected, the idea of taking advantage of these connections emerged not only for message exchange but for entities movement. In the early days only single data was moved but mobility has evolved in such a way that it allows moving code and execution control / environment.

The first stage of such evolution includes file mobility, for instance, the FTP protocol; after that, remote procedure calls (RPC) [BIRRELL84] were introduced, where execution flow is moved. Later, the idea of code movement arises: a process is sent to other machine where it runs. If we consider the mobility of more complex entities, such as active objects, an implicit state with variable values has to be considered. In addition, if an active object is in execution, both a stack and a program counter are assigned to the object.

Agent system

An agent system is a platform that can create, interpret, execute, transfer, and terminate agents. Figure 2.3 shows an agent system.

A communications infrastructure provides communication transport services (e.g., CORBA), naming service, and security services for an agent system.

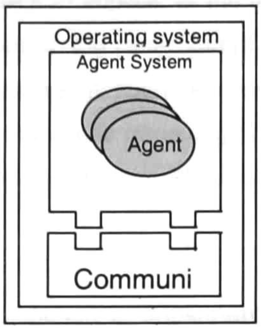


Figure 2.3 An agent system.

Mobile agent

A mobile agent is not bound to the system where it begins execution. It has the unique ability to transport itself from one system in a network to another. The ability to travel permits a mobile agent to move to a destination system that contains another agent with which the agent wants to interact [OMG00]. When an agent travels, its state and code are transported with it. In this context, the agent state can be either its execution state, or the agent attribute values that determine what to do when execution is resumed at the destination agent system. The agent attribute values include the agent system state associated with the agent (e.g., time to live).

Kinds of mobility

Two kinds of mobility have to be distinguished: weak and strong [FUGGETA98]. The former case of mobility permits the migration of the code and the values of the agent variables. After the migration the agent is re-started and the values of its variables are restored, but its execution starts from the beginning or from a specific procedure (a method in the case of objects). In case of strong mobility, not only code is moved, but also the whole execution state, in order to restart the execution exactly from the point where it was stopped before migration. For example, in Java it is possible to implement a weak mobility by means of serialising objects [SUN97][OMG00] and sending them to another Java virtual machine via socket or RMI. In general the operating system must provide facilities to implement strong mobility. To implement a strong mobility of Java Threads, it is necessary to modify the JVM code [SUN97], in order to extract the Java stack and the program counter of the thread to be moved.

Further, the mobility can be explicit or implicit. In the former case, an agent asks explicitly to change its execution environment. In the case of implicit mobility, the execution environment hosting decides when to move the agent; such decision can be taken on the basis of different needs, such as load balancing, resources retrieving or meetings request. For example, when an agent asks to use a printer the system migrate it to the host where the printer is available.

2.7.1 Mobile agents systems

Different systems have been proposed to implement mobile agents. There is an effort to reach standard which permits interoperability among agents of different systems [OMG00], but, currently, most of the proposed systems are not compliant with each other. Some mobile agents systems are: Agent-Tcl [GRAY96], Sumartra [ACHARYA97], Telescript [ODISSEY], Odyssey [WHITE94], Jade [JADE], and Java-to-go [LI].

2.7.2 Co-ordination in mobile agents

During their nomadic life, mobile agents need to interact with other entities in order to carry out their jobs. In particular, they have to interact with the local environment of each site they arrive on, and with other agents. Therefore, for the design of mobile agents it is necessary to consider agent-local environment coordination and inter-agents coordination.

Direct co-ordination, meeting-oriented model, blackboard-based model and linda-like models [CABRI99] are model of agent co-ordination; each of them can be applied both to agent-local environment and inter-agents co-ordination [CABRI99].

2.7.3 Security

A mobile agent is a foreign entity that is able to access to local resources, this imposes the adoption of security care to defend the visited sites from malicious or bad-programmed agents. On the other hand, programmers have to protect their agents from unknown sites [OMG00], for example, if an agent that act as a virtual customer is in charge of visiting different virtual shops and then choose the cheapest one, a malicious site can alter the information about other sites to result the cheapest, even if it is not true.

Site security

The basic mechanism to support security in mobile agent systems is identification. Every agent must be identified. The authentication follows the identification, to grant that a given identity is true. Note that the identification and the authentication do not avoid system damages, but permit to identify the responsible of such damages. For example, a security policy could isolate incoming agents and to prevent them from doing everything but exploiting only few resources.

Agent security

Here, the security problem rises because the agent relies on resources, such memory, network connections and CPU, which are completely handled by the hosting site. So internal variables of the agent could be changed, results of its computation (exploiting local CPU) could be altered; agents themselves could be modified or substituted. The solution of this problem is subject of research.

2.7.4 Applications of mobile agent technology

Mobile agents present several advantages by rapport to non mobile agents [CABRI99]. In our work the two most important are: First, they contribute to a significant bandwidth save. For instance, in applications that require a large amount of remote data, a mobile agent can be translated to process information locally. Second, no stable connections are required to grant the execution of applications. In fact, an agent, after being sent, can directly operate on data on the remote server that host him, this means that is not required any connection. A stable connection is requested only at the beginning, to send the agent, and at the end of the application, to take the agent back, but it is not requested during the whole application execution. In order to describe the advantages of mobile agents, in the following of this section we describe some examples of applications that can take advantages by exploiting mobile agent technology.

- **Internet information retrieval:** The internet has given us the access to a large amount of information, but due to the wide spread of such information – the search of a particular topic can become difficult, implying a waste of time and use of bandwidth. A mobile agent-based solution providing agents that visit servers searching for interesting pages; this approach saves bandwidth because the only needed communications are those to send and receive the agent. Besides, this solution is the only that fits well mobile computing; in fact, in a scenario where a PDA (Personal Digital Assistant) is only occasionally connected to the internet, an user can send his searching agent, disconnect and reconnect later, and when the agent has finished its job then it comes back to the user to show him the results of his request. The advantages taken by agents are particularly relevant if the user has a slow connection to the main network (for example, a connection to the internet via modem): in this cases the slow connection is used only twice, limiting the time cost.

- **Data processing:** An example of intensive data processing is a system for retrieve graphic information, such an application process images trying to find the required. This sort of work is done on image databases, which different clients can access to. In a traditional client-server approach, the client can ask the image server for images, which are retrieved from the server and processed locally to the client host. This approach need of large bandwidth. In a mobile agent approach a processing agent is sent from the user site to the image server, where it carries out its processing job. This approach needs only two connections, one to send the processing agent, and one to take it back with the results of his job (for example, the references to the images that contain the researched particular).
- **Mobile computing systems:** The wide spread of notebook computers and PDAs points out the need of new paradigms for their applications. Mobile agents can be used to model applications that reside on a PDA and use resources spread over the network. In fact, even if such applications do not move through networks links, they are physically moved because the PDAs where they execute are mobile, and can be connected to different hosts of the network which become the execution environments of the applications.

2.8 Formal methods

Formal specification constitutes an important aspect in designing and building systems. At design time, a model of a system is built from user specifications of such system; once this model is probed we can pass to the implementation phase. There is a large variety of formal methods to probe that the model designed satisfies the user requirements for instance: logic expressions [PNUELI95], state-machines [KELLEY95], Petri nets [LOPEZ97] and process algebra [LOGRIPPO]. Each formal method offers elements and conventions to represent the system behavior, most of them in terms of system state, events and precedence relations between states. Furthermore, each formal method allows, with a different degree of clearness and compactness, to represent behaviors such as causal relation, parallelism, synchronization between processes whose evolution and decisions taking are both concurrent. In the next subsection we describe briefly Petri nets formalism because we used it in this work.

2.8.1 Petri nets

Petri nets are one of the most popular formal models of concurrent systems. The problem of how analyze Petri nets – i.e., given a Petri net and a property, how to decide if the Petri nets satisfies it or not – has been intensely studied.

The Petri net is a mathematical model of a parallel system, in the same way that the finite automaton is a mathematical model of a sequential system. Petri nets have a faithful and convenient graphical representation, which we will use in this informal introduction [DESEL95].

Petri nets structure

A Petri net is a directed graph with two sorts of nodes such that there is no edge between two nodes of the same sort. These two sorts of nodes are called places and transitions. Places are graphically represented by circles, and transitions by boxes [DESEL95]. Figure 2.4 show the appearance of one Petri net.

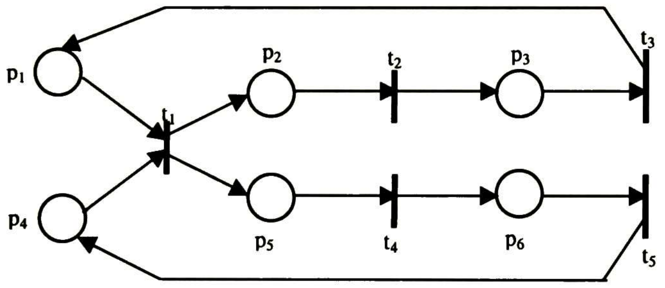


Figure 2.4 A Petri net

Note that t_1 has p_1 and p_4 as input places, and p_2 and p_5 as output places. The number of arcs that links two nodes can be whatever; by convention, multiple arcs are graphically denoted by a number that represents the multiplicity or weight of the arc. If all of the Petri net arcs have weight equal to one, then we call it an ordinary Petri net, otherwise we call it a generalized Petri net.

Marking of Petri nets

Petri net places can store tokens, represented by black dots. A Petri net represents a dynamic behavior with the help of these marks. A distribution of tokens on the places of a Petri net is called a marking, and corresponds to a state of the Petri net (actual system).

Evolution of ordinary Petri nets

The dynamic behavior of a Petri net can be described by the evolution of its tokens. A transition of a net is enabled at a marking if all of its input places (the places from which some edge leads to it) contain at least one token. An enabled transition can occur, and its occurrence changes the marking of the net: it removes one token from each of the input places of the transitions, and adds one token to each of its output places. Figure 2.5 shows on the left a Petri net containing an enabled transition, whose occurrence changes the marking to the one shown to the right.

In this way, the occurrence of transitions lead, from initial marking, to a sequence of markings of a Petri net or represented-system's states.

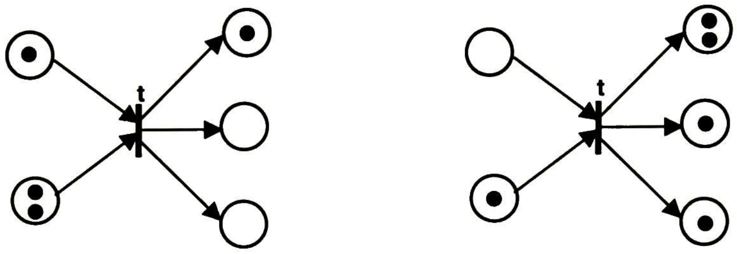


Figure 2.5 A Petri net before and after the occurrence of a transition

In the modeling of systems with Petri nets, places, transitions and tokens take a meaning in agreement to the context of the system modeled. Places may represent resources, operations, partial states, phases of a process. Tokens may represent the availability of resources, execution order of operations, information transferred, etc. Transitions may represent events, such as begin or end or both of activity, relevant information of environment [LOPEZ97]. Therefore, the occurrence conditions of a transition are lightly alter: a transition may occur if it is activated and the event assigned to such transition holds. A Petri net, in which its components have a meaning associated, is called an interpreted Petri net.

Properties of Petri nets

We describe in this section, in an informal way, some of the properties of Petri nets (systems modeled). Liveness and boundedness are two of the most important properties that determine the activity of the net [LOPEZ97].

A Petri net is ***live*** if every transition can always occur again. More precisely, if for every reachable marking (i.e., every marking which can be obtained from the initial marking by successive occurrences of transitions) and every transition *t* it is possible to reach a marking that enables *t*. Deadlock-freedom is a weaker property than liveness. A Petri net is deadlock-free if every reachable marking enables some transition.

A Petri net is ***bounded*** if there exist a number *b* such that no reachable marking puts more than *b* tokens in any place.

Liveness and boundedness are independent of each other. For instance, there exist Petri nets that are live but not bounded.

2.9 Conclusions

In this chapter, we have reviewed some basic concepts related to distributed systems, Distributed Artificial Intelligence (intelligent agents, multi-agent systems and mobile agents), we have introduced some useful technologies for the development of distributed systems (CORBA, RMI) and also useful concepts in order to achieve the goal of dependability – especially in distributed systems. This revision is the basis to accomplish the first part of our work that is the implementation of GeDA-3D.

A comparison of the technologies employed is not performed in this work but in [TOSCANO00], a previous work developed by our team.

Finally, we have described briefly Petri nets, a formal method of verification we use to verify liveness and security of our system. In Chapter 5 we describe the sort of Petri Net we propose to describe easily mobility, main characteristic of our system.

Chapter 3

Evolutionary computation

3.1 Objective

The objective of this chapter is to exhibit the state of the art of evolutionary computation. As we establish in the introduction of this document, the objective of this work is rather a feasibility study to use the evolutionary computation to implement adaptive behaviors of avatars in its virtual environment. This chapter resumes the work we use to implement the behavior of the virtual entities participating in the Virtual Combat Environment of the game we develop and describe in Chapter 6.

3.2 Introduction

Evolutionary computation techniques have received a lot of attention for the solution of complex real-world problems such prediction, generalization, optimization, learning, games etc. These techniques, based on the powerful principle of "survival of the fittest", model some natural phenomena of genetic inheritance and Darwinian strife for survival; they also constitute an interesting category of modern heuristic search. This chapter presents the main paradigms of evolutionary algorithms: evolution strategies, evolutionary programming, genetic algorithms and genetic programming. A particular research direction – self-adaptation – is discussed further in the last part of this chapter.

3.3 Evolutionary computation

The evolutionary computation techniques are stochastic algorithms whose search methods model some natural phenomena: genetic inheritance and Darwinian strife for survival [SCHOENAUER]. In this section we introduce a general framework accounting as much as possible for most of existing Evolutionary Algorithms [SCHOENAUER].

Let the search space be a metric space E, and let F be a function $E \to \mathbb{R}$ called the objective function. The problem of evolutionary optimization is to find the maximum of F on E (The case of minimization is easily handled by considering $-F$).

A population of size $P \in \mathbb{N}$ is a set of P individuals (points of E) not necessarily distinct. This population is generally initialized randomly (at time $t = 0$) and uniformly on E. The *fitnesses* of all individuals are computed (on the basis of the values of the objective function); a fitness value is represented as a positive real number – the higher the number, the better the individual. The population then undergoes a succession of generations: the process is illustrated in Figure 3.1.

```
Procedure evolutionary algorithm
Begin
        t ← 0
        initialize population
        evaluate population
        while (not termination-condition) do
        begin
                t ← t + 1
                select individuals for reproduction
                apply operators
                evaluate newborn offspring
                replace some parents by some offspring
        end
end
```

Figure 3.1 Structure of an evolutionary algorithm

Several aspects of the evolutionary procedure (Figure 3.1) require additional comments:

- **Statistics and stopping criterion:** The simplest stopping criterion is based on the generation counter *t* (or on the number of function evaluation). However, it is possible to use more complex stopping criteria, which depends either on the evolution of the best fitness in the population along generations (i.e. measurements of the gradient of the gains over some number of generations), or on some measure of the diversity of the population.

- **Selection:** Choice of some individuals that will generate offspring. Numerous selection processes can be used, either deterministic or stochastic. All are based on the fitness of the individuals. Depending on the selection scheme used, some individuals can be selected more than once. At that point, selected individuals give girth to copies of themselves (clones).

- **Application of evolution operators**: To each one of this copies some operator(s) is (are) applied, giving birth to one more offspring. The choice among possible operators is stochastic, according to user-supplied probabilities. These operators are always stochastic operators, and we usually are able to distinguish between crossover (or recombination) and mutation operators:

  - Crossover operators are operators from E^k to E, i.e., some parents exchange genetic material to build up one offspring (many authors define crossover operators from ExE to ExE. Two parents generate two offspring). In most cases, crossover involves just two parents (k=2), however, it need not be the case. Several authors have investigated the merits of "orgies", where more than two parents are involved in the reproduction process [EIBEN94].

  - Mutations operators are stochastic operators from E into E.

- **Evaluation:** Computation of the fitnesses of all newborn offspring. The fitness measure of an individual is directly related to its objective function value.

- **Replacement:** Choice of which individuals will be part of the next generation. The choice can be made either from the set of offspring only (in which case all parents "die") or from both sets of offspring and parents. In either case, the replacement procedure can be deterministic or stochastic.

Sometimes the operators are defined on the same space as the objective function (called phenotype space or behavioral space); in other cases, an intermediate space is introduced (called genotype space or representation space). The mapping from the phenotype space to the genotype space is termed *coding*. The inverse mapping from the genotype space to the phenotype space is termed *decoding*. Genotypes undergo evolution operators, and their fitness is evaluated on the corresponding phenotype.

3.4 Main paradigms

There is a general "agreement" that evolutionary computation is "make up" of 4 main branches:

- Evolution Strategies, born in Germany in the 60's [RECHENBERG73] [SCHWEFEL75], to deal with parameter optimization problems.
- Evolutionary Programming, a branch that appeared in California in the 60's as well, and was first applied to Finite State Automata[FOGEL66].
- Genetic Algorithms, which emerged in Michigan in the late 60's [HOLLAND75], and were primarily designed to optimally solve sequential decision processes more than to perform function optimization.
- Genetic Programming [KOZA94], at first considered a subset of genetic algorithms, but now turning into a research field by itself, addressing the challenging problem of employing evolution to teach computers to do things without being explicitly programmed to do so.

3.4.1 Genetic algorithms

In the canonical genetic algorithm (GA) [HOLLAND75] [SCHOENAUER], the genotype space is $\{0, 1\}^n$. The phenotype space can be any space, as long as it can be coded into bitstring genotypes. The selection scheme is a proportional selection (the best-known is the *roulette wheel selection*): *P* random choices are made in the whole population, each individual having a probability proportional to its fitness of being selected. The crossover operators replace a segment of bits in the first parent string by the corresponding segment of bits from the second parent, and the mutation operator randomly flips the bits of the parent according to a fixed user-supplied probability. In the replacement phase, all *P* offspring replace all parents.

Due to that generational replacement, the best fitness in the population can decrease: the original GA strategy is not *elitist*.

In some works [MICHALEWICS96], the genotype space can be almost any space, as long as some crossover and mutation operators are provided. Moreover, proportional selection has been gradually replaced by ranking selection (the selection is performed on the rank of the individuals rather than on their actual fitness), or tournament selection (one selects the best individual among a uniform choice of *T* individuals). Finally, most users use the elitist variant of replacement, in which the best individual of generation *t* is included in generation *t*+1, whenever the best fitness value in the population decreases.

3.4.2 Evolution Strategies

The original evolution strategy (ES) algorithm (1+1)-ES [RECHENBERG73] [SCHWEFEL75] [SCHOENAUER] handles a "population" made of a single individual given as a real valued vector [BACK] [RUDOLPH]. This individual undergoes a Gaussian mutation: addition of zero-mean Gaussian variable of standard deviation σ. The fittest from the parent and the offspring becomes the parent of next generation. The critical feature is the choice of parameter σ: Originally, the so-called 1/5 thumb rule was used to adjust parameter σ along evolution.

Rechenberg postulated his 1/5 success rule:

> The ratio of successful mutations to all mutations should be 1/5. If it is grater than 1/5, increase the mutation variance; if it is less, decrease the mutation variance [BACK] [RUDOLPH].

More recent ES algorithms [BACK] [RUDOLPH] are population-based algorithms, termed (μ,λ)-ES or (μ+λ)-ES: μ parents generate λ offspring (there is no selection at that level, i.e., every parent produces μ/λ offspring on average).

The main operator remains mutation. When working on real-valued vectors ES generally uses the powerful paradigm of self-adaptive mutation: the standard deviations of Gaussian mutations are part of the individuals, and undergo mutation as well.

The replacement step is deterministic, i.e., the best μ individuals become the parents of the next generation, chosen among the μ+λ parents plus offspring in the elitist (μ+λ)-ES scheme, or among the λ offspring in the non-elitist (μ,λ)-ES scheme.

3.4.3 Evolutionary programming

Originally designed to evolve finite state machines, evolutionary programming (EP) emphasizes the phenotype space [SCHOENAUER]. As in ESs, there is no initial selection: Every individual in the population generates one offspring. Moreover, the only evolution operator is mutation. Finally, the best *P* individuals among parents and offspring become the parents of the next generation.

Some researchers [FOGEL55] handle any space, they still emphasize the use of mutation as the only operator, and designed independently the self-adaptive Gaussian deviations for real-valued variables, and now use a stochastic tournament replacement scheme: each individual (among the 2*P* parents plus offspring) encounters *T* random opponents, increasing its score by one point if it has better fitness. The *P* individuals having the highest scores get along to the next generation. Note that EP replacement scheme is always *elitist*.

3.4.4 Genetic programming

Genetic programming is a method for evolving computer programs that first appeared as an application of Gas [KOZA94] [SCHOENAUER] to tree-like structures. Original GP evolves tree structures representing LISP-like S-expressions. This allows defining very easily a closed crossover operator (by swapping sub-trees between two valid S-expressions, we always get a valid S-expression). The usual evolution scheme is the steady state genetic algorithm [BACK97]: a parent is selected by tournament (of size 2 to 7 typically), generates an offspring by crossover only (the other parent is selected by tournament of usually smaller size). The offspring is then put back in the population using a death-tournament: *T* individuals are uniformly chosen, and the one with the worse fitness gets replaced by the newborn offspring.

Genetic programming is an extension of the conventional genetic algorithm in which each individual in the population is a computer program (any computer program can be graphically depicted as a rooted point-labeled tree with ordered branches). The search space in genetic programming is the space of all possible computer programs composed of functions and terminals appropriate to the problem domain. The functions may be standard arithmetic operators, standard programming operations, standard mathematical functions, logical functions, or domain-specific functions.

3.5 Self-adaptation

As evolutionary algorithms implement the idea of evolution, and as evolution itself has evolved to reach its current state of sophistication, it is natural to expect adaptation not to be used only for problem solution finding, but also for algorithm tuning to a particular problem [SCHOENAUER].

In EAs, not only we need to choose the algorithm, representation and operators for the problem, but we also need to chose parameter values and operator probabilities for the evolutionary algorithm to find the solution and, which it also important, find it efficiently. This is a time consuming task and a lot of effort has gone into automating this process. Researches have used various ways of finding good values for the strategy parameters as these can affect the performance of the algorithm in a significantly manner. Many researches experimented with problems from a particular domain, tuning the strategy parameters on the basis of such experimentation (tuning “by hand”). Later, they reported their results of applying a particular EA to a particular problem, stating:

> For these experiments, we have used the following parameters:
>
> Population size = 80, probability of crossover = 0.7, etc.

Without much justification for the choice they made. Others researches tried to modify the values of strategy parameters during the run of the algorithm; it is possible to do this by using some (possible heuristic) rule, by taking feedback from the current state of the search, or by employing some self-adaptive mechanism. Clearly, by changing these values while the algorithm is searching for the solution of the problem, further efficiencies can be gained.

Self-adaptation, based on the evolution of evolution, was pioneered in evolution strategies to adapt mutation parameters to suit to the problem during the run. The method was very successful in improving the efficiency of the algorithm. This technique has been extended to other areas of evolutionary computation, but fixed representation, operators and control parameters are still the norm. [HINTERDING] shows a classification of adaptation.

3.6 Conclusions

In this chapter, we have reviewed a few basic concepts related to evolutionary computation since they serve as a background for the second part of our work. We have introduced the main paradigms of evolutionary computation although we used two of these paradigms in our work, namely genetic algorithms and evolution strategies. We employed these paradigms for the reason that they are appropriate to accomplish searches within big search spaces, and because these paradigms impose few restrictions of mathematical type on the form of the function that we pretend to optimize.

Chapter 4

Innovations to GeDA-3D

4.1 Objective

This chapter presents the components and main characteristics of the generic architecture GeDA-3D, and the innovations achieved on this architecture to provide services needed by our Virtual Editor [RAMOS02] and allow agents to work with the adaptive algorithms proposed.

4.2 Introduction

Nowadays, several programming environments that assist the development of agent-based applications are available. Nevertheless, issues such as graphic user-interface and application-integration are not addressed in most of these environments. We propose a 3D-Space platform useful to integrate and manage distributed applications. This platform is based on our *mobile-agent architecture* and has been designed specifically for the development of dynamic virtual environments. Several real-life applications are manageable by our platform, including: Computer Supported Cooperative Work, e-Commerce, Messaging Service, Networked Games, training systems, etc. It is our intend that GeDA-3D provides a number of tools necessary to generate and manage dynamic virtual environments. This work is strongly based on a previous work developed by our team and described in [TOSCANO00] [PUGA01]. Due to the nature of the applications intended to be managed by our platform, some features were added to the architecture previously proposed, in order to provide a certain degree of performance, flexibility, robustness and facilities for the development of virtual environments.

4.3 Model architecture

The model of the architecture proposed by [TOSCANO00] has been thought to facilitate the implementation of distributed systems with different nature. The architecture can be seen as a middleware providing a number of constant features useful to develop such cooperative systems. The design of this architecture follows the agent-oriented paradigm in order to take advantage of multiagent systems [SYCARA98]. Thus, agents help users to manage cooperative distributed applications.

The architecture is constituted by next community of three sorts of agents having specific skills:

- **Coordinator**: take in charge of the management of the various cooperative systems integrated with the architecture and also of the end-users connected to it.
- **Control agents**: represent distributed applications (services) integrated to the architecture.
- **Interaction agents**: act as an interface between the architecture and the end-user, in the form of a virtual environment.

Taking this informal specification as a basis, the Figure 4.1 illustrates the architecture, using CORBA [OMG95a] as a means of communication.

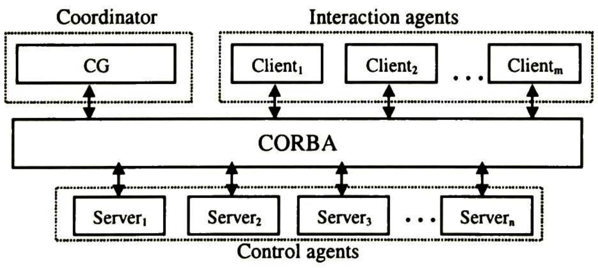


Figure 4.1 Architecture of GeDA-3D

4.4 Innovations to GeDA-3D

Our work intends to design and to implement a platform based on the model architecture, but providing some enhancements needed to support and facilitate the virtual environments development [UK][SNOWDON94][CARLSON93]. Three main innovations to GeDA-3D include: support for mobility, splits services provided by the coordinator and support for concurrent distributed applications.

In this section, the new architecture is introduced, and the services provided by the agents constituting GeDA-3D are described. Then, the functionality of two underlying enhancements, the consistency service and the mobility platform, are analysed deeply.

4.4.1 Community of agents

The community of agents and their competencies remain the same in general terms, but new features are added as explained lately. In order to simplify this presentation, from now on, control and interaction agents are called applications and clients, respectively.

4.4.1.1 Coordinator

This agent is constituted by six services running in different locations and bounded through our transport layer. We choose distribute this agent in order to allow more services to be easily added to the coordinator. This will enable GeDA-3D for instance to deal with dynamic groups, if ever the service is implemented.

- *Look Up:* Resolves the reference of all services and applications currently available in GeDA-3D.
- *Users*: Manage the operations occurring in the users' database and validates users logins.
- *Applications*: Manage the operations occurring in the applications' database and validates application's login.
- *Consistency:* Manages all the changes performed in the virtual environment, including users' displacements and rotations.
- *Chat:* Provides point to point messaging between connected users.

- *Security:* Provides mechanisms to prevent GeDA-3D from intruders with knowledge of the services references and its public methods.

4.4.1.2 Application

Any application developed on GeDA-3D should follow some defined templates and must include next three main elements:

- *Server:* Binds the application to GeDA-3D and provides public methods.
- *Mobile interface:* Stands for the user interface. It is cloned and sent to the client requester as many times as necessary. Once in execution, the Mobile Interface invokes server public methods.
- *Host:* Synchronizes with the client requester to send the mobile interface.

4.4.1.3 Client

Provides the means for the user becomes part of the MAS implemented using GeDA-3D. This agent allows a client take advantage of the provided services. The Client comprises three main elements.

- *Interface:* Provides a shared virtual environment where connected users may perform motions, rotations, interact with other users and launch distributed applications.
- *Listener:* Receives every change performed within the shared environment and messages from other users.
- *Host*: Synchronizes with the application provider to receive the mobile interface via a Socket.

4.4.2. Architecture

Figure 4.2 illustrates the final architecture proposed for GeDA-3D. All the services provided by the coordinator have a number of replicas which depends on how critical the services are. In addition, both the Consistency Service and the Client Interface keep a database of the virtual environment, including the state of the entities currently present; the Chat service, as well as the Consistency Service, include a list storing the references of the Clients Listeners in order to forward to them any changes performed in the virtual environment or in the user messages.

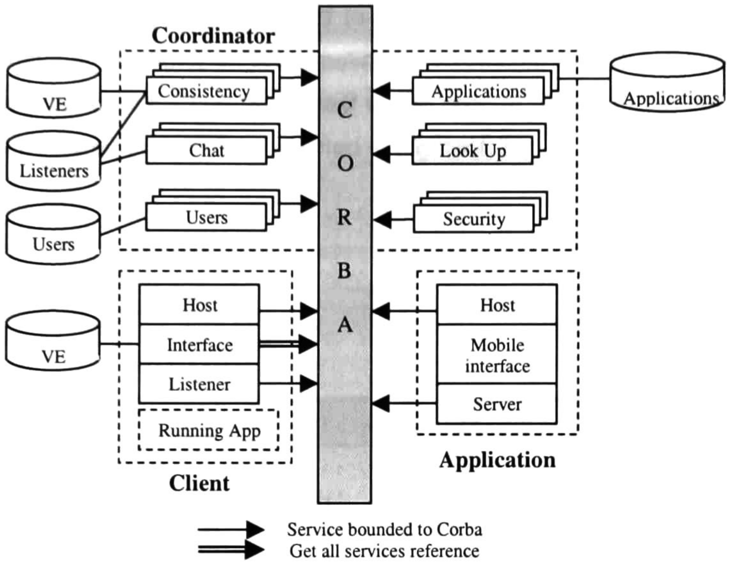


Figure 4.2 Architecture of GeDA-3D

4.4.3 Consistency service

As mentioned above, the Client provides a virtual environment, that is, a 3D-space where all the users and applications connected to GeDA-3D meet and interact with each other in the form of a 3D avatar. Clearly, this virtual space is common to all the users and, therefore, every change performed by an avatar is immediately notified to the rest of the connected users. The changes include: appearances, motions, rotations and logouts. When an avatar enters the virtual environment (VE), an *appearance* is performed; the avatar is assigned a 3D-coordinate (*x*, *y*, *z*) and an angle between 0º and 360º. The former stands for the virtual location of the avatar inside the VE, and the latter for its field of view. A displacement of an avatar in the VE results in a change of its 3D-coordinate that is called a *motion*; similarly, a change of direction of an avatar results in a change of its angle, called a *rotation*. When an avatar leaves the VE, a *logout* is performed.

As mentioned above, the main responsibility of the Consistency Service (CS) is to address all the changes performed by the avatars belonging to the VE. The operation of the CS is: as soon as a client *c* is both authorized to take part of GeDA-3D and assigned a virtual layout (avatar), a reference from its listener is sent to the CS where all the listener references are stored. Immediately, the CS resolves a pair {3D coordinate, angle} representing a virtual space free from collisions for the client's avatar; after that, the CS requests the appearance of this avatar to all the clients connected (except *c*) and sends back the environment to *c*. Whenever an avatar performs either a motion or a rotation, the new values are sent to the CS, where they are broadcasted to the rest of the clients. The general operation of the CS after changes occurred in the VE is depicted in Figure 4.3.

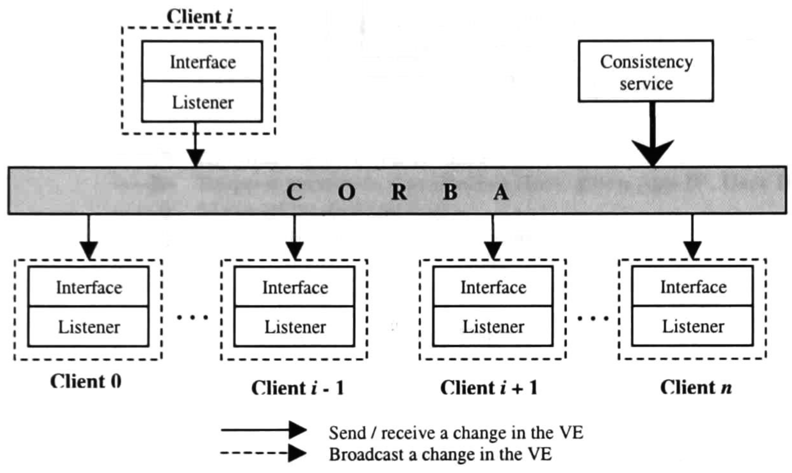


Figure 4.3 Consistency Service

4.4.4 Mobility platform

As mentioned above, GeDA-3D is constituted by a community of static (as apposed to mobile) agents but also has the capability for managing mobile agents and provides the mechanisms to support and perform with certain ease mobile applications. This new capability of our architecture allows the users to run remote applications locally. In this context, a mobile agent (application) is not forced to remain in the system where it was started; instead it has the ability to move from a system to another containing an agent or resource to interact with [OMG00].

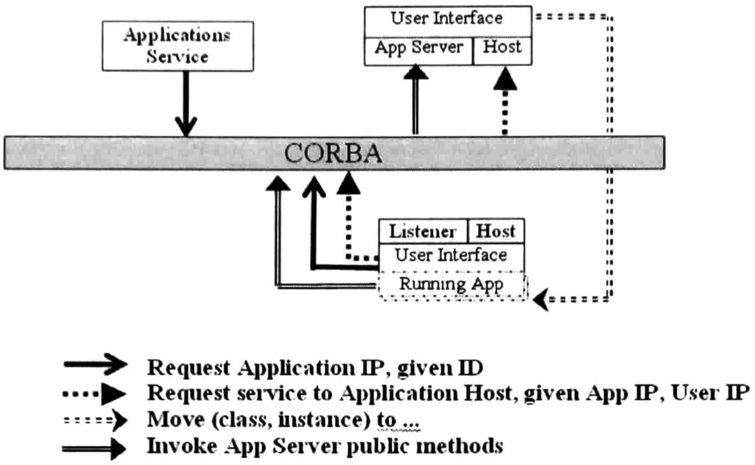

Applications Service
User Interface
App Server
Host
CORBA
Listener
Host
User Interface
Running App
Request Application IP, given ID
Request service to Application Host, given App IP, User IP
Move (class, instance) to ...
Invoke App Server public methods

Figure 4.4 Mobility Platform

Whenever an agent travels across the GeDA-3D network, both its state and source code are moved. The agent state includes some parameters determining which process resumes the agent execution as soon as the application arrives to the host requester.

Figure 4.4 depicts the way GeDA-3D performs agent mobility when an application is requested by a user. Both Client and Application agents include a service called Host that behaves as an agent system. Our implementation of mobility follows the OMG's lineament [OMG00]. This establishes that an agent system refers to a platform enabling agent creation, interpretation, execution, transference and completion. Figure 4.5 shows an agent system.

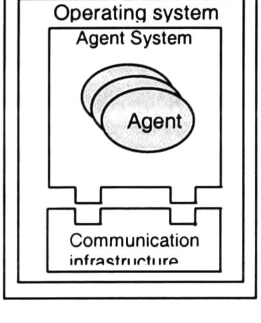


Figure 4.5 An agent system.

An agent-host system can communicate with other agent systems for agent transferring. The set of all the agent systems is called a Region – taken from the OMG specification. Figure 4.6 shows the interconnection of two agent systems.

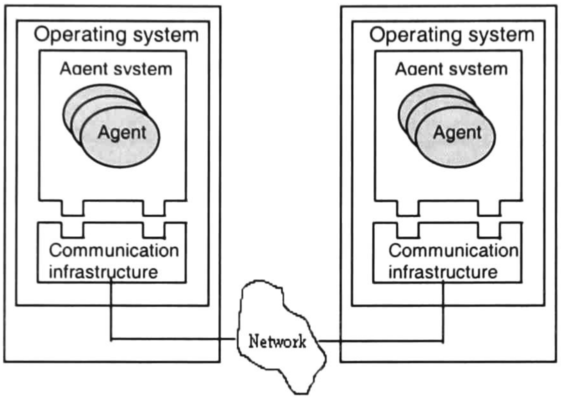


Figure 4.6 Agent systems to agent system interconnection

A communications infrastructure provides communication transport services (e.g. CORBA), naming service, and security services for an agent system. Serialization techniques are used before an agent is sent across the network and deserialization when the agent is received from the network.

The key to storing and retrieving agents is to represent the state of an agent in a serialized form that is sufficient to reconstruct the agent. Notice that the serialized form must be able to identify and verify the classes from which the fields were saved.

For not object-oriented agent systems, the agent state refers to the extraction of runtime data for the agent, and the classes that refer to the code implementing the agent.

4.5 Conclusions

In this work, a 3D-interface CORBA-based platform (GeDA-3D) useful to integrate and manage distributed cooperative applications from different nature and based on a mobile-agent architecture is proposed. Through a series of libraries, this platform supports and facilitates the development of virtual environments and mobile programming. The GeDA-3D conception is the result of animated discussions of our team. The implementation was made by H. Piza and me. My work was devoted mainly to implement the Mobility capability of GeDA-3D.

Chapter 5

Specification of GeDA-3D's mobility

5.1 Objective

Present an original approach we propose to specify and verify formally GeDA-3D using an extension of Petri Nets.

5.2 Introduction

GeDA-3D is a distributed system, it design was made following the spiral approach proposed by [BOEHM88]. However, we found necessary to prove formally its behaviour. After a study of possible formal techniques available, we found that formal specification work of GeDA-3D will be very hard. This fact makes us to propose an original approach using Petri Nets [DESEL95] [LOPEZ97] to prove at least liveness. That is, in our case to prove that if we are given a finite number of clients and applications connected to GeDA-3D, then its whole operation ends successfully. To prove the latter, it suffices to model the behaviour of the community of agents.

Particularly, we use Elementary Object Systems (EOS) [VALK1] [VALK2], an extension of Petri Nets. The Client and the Application have both specific behaviours and are represented by Object Nets; the Coordinator manages every single request of the clients and applications connected and are represented by a System Net. The model together with the *liveness* and *boundness* proof of the coordinator and Consistency service was achieved by H. Piza [ZÚÑIGA02], and it is included also on his thesis work [PIZA02].

The Mobility in GeDA-3D involves next three elements depicted in Figure 5.1: a region, agent systems and mobile agents; hence, three different levels are necessarily considered. Since the Valk's work [VALK1] [VALK2] provides only a 2-level abstraction, the third level can not be modelled, therefore we propose to extend the EOS one more level. We name this extension proposed *Extended Object System* (ExOS). We had a big problem to prove liveness in the mobility platform because mobile agents are represented by tokens of a net. Therefore, when an agent transfer is performed, the token must travel from the source Petri net (source agent system) to the destination Petri net (destination agent system). We had also to add the rewriting rules for the highest level of the proposed extension.

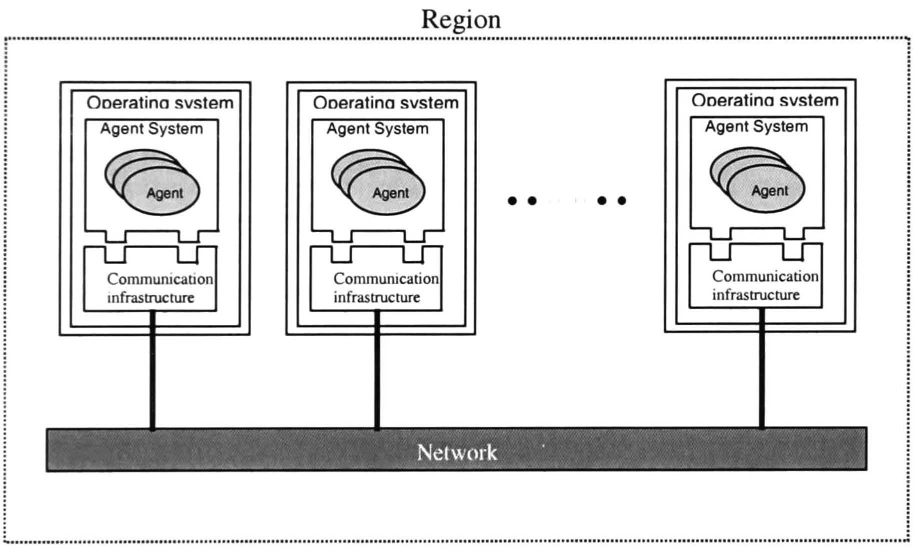


Figure 5.1 Elements of GeDA-3D's Mobility

5.3 A Petri Nets-based methodology

This section presents formal definitions of the kind of Petri nets used to model our system. We give some formal definitions of ordinary Petri nets (see chapter 2) and following the work of Rüdiger Valk in [VALK1][VALK2], we present a formal definition of Elementary Object Systems (an extension of Petri Nets) with few changes. Also we provide an extension of the Rüdiger Valk work[1].

Definition 1. Nets, pre-sets, post-sets, sub-nets

A net N is a triple(S, T, F), where S and T are two finite disjoint sets, and F is a relation on S $\cup$ T such that F $\cap$ (S $\times$ S) = F $\cap$ (T $\times$ T) = $\varnothing$.

The elements of S are called places, and they are graphically represented by circles. The elements of T are called transitions and they have the shape of a box. F is called the flow relation of the net, represented by arrows from places to transitions or from transitions to places.

Given a node *x* of N,

$\bullet x = \{y \mid (y, x) \in F\}$ is the pre-set of *x*

$x\bullet = \{y \mid (x, y) \in F\}$ is the post-set of *x*.

The elements in the pre-set (post-set) of a place are its input (output) transitions. Similarly, the elements in the pre-set (post-set) of a transition are its input (output) places.

A triple (S', T', F') is a sub-net of N if: S' $\subseteq$ S, T' $\subseteq$ T and F' = F $\cap$ ((S' $\times$ T') $\cup$ (T' $\times$ S')).

If X is a set of elements of N, then: (S $\cap$ X, T $\cap$ X, F $\cap$ (X $\times$ X)) is a sub-net of N, called the sub-net of N generated by X.

Definition 2. Paths

A path $x_1 \dots x_k$ of a net (S, T, F) is a nonempty sequence of nodes which satisfies: $(x_1, x_2), \dots, (x_{k-1}, x_k) \in F$. A path $x_1 \dots x_k$ is said to lead from x_1 to x_k.

[1] thanks to the help of H. Almeyda, R. Campos and E. Lopez

Definition 3. Connectedness

A net (S, T, F) is connected iff no two sub-nets (S_1, T_1, F_1) and (S_2, T_2, F_2) with disjoint and nonempty sets of elements satisfy $S_1 \cup S_2 = S$, $T_1 \cup T_2 = T$ and $F_1 \cup F_2 = F$ A net (S, T, F) is strongly connected if $(x, y) \in F^*$, i.e., for every two nodes *x*, *y* there is a path leading from *x* to *y*.

Definition 4. Markings, occurrence rule

A marking of a net (S, T, F) is a mapping M: $S \rightarrow \mathbb{N}$. A marking is often represented by a vector $(M(s_1)...M(s_n))$, where $s_1, s_2, \ldots, s_n$ is an arbitrary fixed enumeration of S.

A place s is marked at a marking M if M(s)>0. A set of places R is marked if some place of R is marked.

The total number of tokens in a set of places R is denoted by M(R), i.e., M(R) is the sum of all M(s) for $s \in R$. The restriction of a marking M to a set of places R is denoted by $M|_R$. The null marking is the marking which maps every place to 0.

A marking M enables a transition *t* if it marks every place in $\bullet t$. If *t* is enabled at M, then it can occur, and its occurrence leads to the successor marking M' (written $M \rightarrow_t M'$) which is defined for every place s as follows:

$$M'(s) = \begin{cases} M(s), & \text{if } s \notin \bullet t,\ s \notin t\bullet \text{ or } s \in \bullet t,\ s \in t\bullet \\ M(s) - 1, & \text{if } s \in \bullet t,\ s \notin t\bullet \\ M(s) + 1, & \text{if } s \notin \bullet t,\ s \in t\bullet \end{cases}$$

(A token is removed from each place in the pre-set of *t* and added to each place in the post-set of t).

A marking M is called *dead* if it enables no transition of the net.

Definition 5. Transition occurrences and reachable markings

Let M be a marking of N. if $M \rightarrow_{t1} M_1 \rightarrow_{t2} \ldots \rightarrow_{tn} M_n$ are transitions occurrences, then $\sigma = t_1, t_2, \ldots t_n$ is an occurrence sequence leading from M to M_n and we write $M \rightarrow_\sigma M_n$. This notion includes the empty sequence λ; we have $M \rightarrow_\lambda M$ from every marking M.

We write $M \rightarrow^{*} M'$, and call M' reachable from M, if $M \rightarrow_{\sigma} M'$ for some occurrence sequence σ. The set of all markings reachable from M is denoted by [M>.

Definition 6. Net Systems, initial and reachable markings

A net system (or just system) is a pair $(N.M_0)$ where.-

- N is a connected net having at least one place and one transition, and
- M_0 is a marking of N called the initial marking.

A marking is called reachable in a system if it is reachable from the initial marking.

Definition 7. Liveness and related properties.

A system is live if, for every reachable marking M and every transition *t*, there exists a marking $M' \in [M>$ which enables *t*. If (N, M_0) is a live system, then we also say that M_0 is alive in N.

A system is deadlock-free if every reachable marking enables at least one transition; in other words, if no dead marking can be reached from the initial marking.

Definition 8. Elementary object systems

An elementary object system is a 5-tuple EOS = (SN, ON, ρ, type, M) where:

- SN = (P,T,W) is a net called the system net of EOS
- $ON = \{ON_1, \ldots ON_n \mid n \geq 1\}$ is a finite set of net systems, called object systems of EOS, denoted by $ON_i = (B_i, E_i, F_i, m_{0i})$.
- ρ is the interaction relation defined as follows: $\rho \subseteq T \times E$, where $E := \cup\{E_i \mid 1 \leq i \leq n\}$
- type: $W \rightarrow \{\{1, 2,..n\}\} \cup \mathbb{N}$
- M is a marking as defined in definition 9

Intuitively, an object net ON_i can be moved along an arc(*x*, *y*) if $i \in$ type(*x*, *y*). Arcs of type = k $\in \mathbb{N}$ are labeled by $k \in \mathbb{N}$. They are used as in the case of the nets defined formerly. $x\rho y$ holds iff *x* and *y* are marked by the same label of the form $<i_j>$. Next, a marking will be defined as an assignment of a subset of the object nets together with a current marking to the places. It is also possible to assign a number *k* of tokens.

Definition 9. Object-marking

The set obj:=$\{(ON_i, m_i) \mid 1 \leq i \leq n, m_i \in [m_{0i}\rangle\}$ is the set of objects of the EOS. An object-marking (O-marking) is a mapping $M : P \rightarrow 2^{obj} \cup \mathbb{N}$ such that $M(p) \cap obj \neq \varnothing \Rightarrow M(p) \cap \mathbb{N} = \varnothing$, for all $p \in P$

Definition 10. *i* – components of an EOS

Let EOS = (SN, ON, ρ, type, M) be an elementary object system as given in definition 8, but in some arbitrary marking M.

- The i-component ($1 \leq i \leq n$) of EOS is the net system $SN(i) = (P, T, W(i), M_{0i})$ defined by $W(i) = \{(x, y) \mid i \in type(x, y)\}$ and $M_{0i}(p) = 1$ iff $(ON_i, m_i) \in M(p)$. The 0-component (zero-component) is the net $SN(0) = (P, T, W(0), M_{00})$ with the arc weight function $W(0)(x, y) = k$ if $type(x, y) = k \in \mathbb{N}$ and $M_{00}(p) = k \in \mathbb{N}$ iff $k \in M(p)$.

- The sub-net $SN(1\ldots n) = (P, T, W(1\ldots n), M_{1\ldots n})$, where $W(1\ldots n) = \cup\{W(i) \mid 1 \leq i \leq n\}$ and $M_{1..n}(p) = M(p) \cap obj$ is said to be the object component.

- EOS is said to be a simple elementary object system if $SN(1\ldots n)$ is a structural state machine and all *i*-components of SN are state machines.

Restriction 1. We will focus on elementary object systems where every transition $t \in SN(1\ldots n)$ has one input place.

Definition 11. Occurrence rule in elementary object systems

Let EOS = (SN, ON, ρ, type, M) be an elementary object system as in definition 8 and $M : P \rightarrow 2^{obj} \cup \mathbb{N}$ an O-marking and $t \in T$, $e \in E_i$.

a) Transition $t \in T$ is activated in M(denoted $M\rightarrow_t$) if $t\rho = \varnothing$ and the following holds:

 1. *t* is activated in the zero-component of SN (definition 10) (i.e. in the net part).

 2. if there exists a place $p \in \bullet t$ where $type(p, t) = \{1, 2, \ldots n\}$ (for restriction 1 there exists one at most) then there exists at least an object $(ON_i, m_i) \in M(p)$ where $i \in \{1, 2, \ldots n\}$.

 if *t* is activated, then *t* may occur ($M\rightarrow_t M'$) and the following marking M' is defined as follows: with respect to the zero-components tokens are changed according to the ordinary Petri nets occurrence rule. In case of a.2 (ON_i, m_i) is removed from *p* and added to every place $l \in t\bullet$ where $i \in$ type(*l*, *t*).

b) A pair [*t*, *e*] $\in T \times E_i$ with $t\rho e$ is activated in M (denoted by: $M \rightarrow_{[t, e]}$) if the following holds:

1. *t* is activated in the zero-component of SN (definition 10) (i.e. in the net part).
2. (ON_i, m_i) $\in$ M(*p*) where $p \in \bullet t$ with type(*p*, *t*) = {1, 2, ... *n*} (for restriction 1 there exists one at most).
3. Transition e is also activated for ON_i in m_i.

if [*t*, *e*] is activated, then it may occur ($M \rightarrow_{[t, e]} M'$) and the following marking M' is defined as follows: with respect to the zero-components tokens are changed according to the ordinary Petri nets occurrence rule. In case of b.2 and b.3 instead of (ON_i, m_i) the changed object (ON_i, m_{i+1}) where $m_i \rightarrow_e m_{i+1}$ is added to every place $l \in t\bullet$ where $i \in$ type(*l*, *t*) and (ON_i, m_i) is removed from *p*.

c) A transition $e \in E_i$ with $\rho e = \varnothing$ is activated in M (denoted by $M \rightarrow_e$) if for some place $p \in P$, the following holds: (ON_i, m_i) $\in$ M(*p*) and $m_i \rightarrow_e m_{i+1}$. In the following marking M' the object (ON_i, m_i) is replaced by (ON_i, m_{i+1}).

From this point, is defined the extension of the Valk's work we propose to model formally our architecture.

Definition 12. Extended object system

An extended object system is a tuple ExOS = (ESN, EON, σ, EM, R) where:

- ESN = (P', T', W') is a net called extended system net of ExOS.
- EON = {EOS_1, ... EOS_m}($m \geq 1$) is a finite set of elementary object systems, denoted by EOS_i = (SN_i, ON_i, ρ_i, $type_i$, M_i).
- $\sigma \subseteq T' \times T$ is the interaction relation where $T := \cup \{T_i \mid 1 \leq i \leq m\}$
- EM is a marking as defined in 13
- R is a set of rewriting rules

$x\sigma y$ holds iff *x* and *y* are marked by the same label of the form $<i_j>$.

Definition 13. Extended object-markings

Let ExOS = (ESN, EON, σ, EM, R) be an extended object system and let Z = $\cup$ {co-domain *r* | $r \in$ R}. Eobj = $\{(SN_i, M_i) \mid 1 \leq i \leq m, M_i \in [M_0\rangle\} \cup \{(SN, M) \mid (SN, ON, \rho, type, M) \in Z\}$ is the set of extended objects of the ExOS. An extended object-marking (EO-marking) is a mapping EM : $P' \rightarrow 2^{Eobj}$.

Definition 14. Rewriting rule

Let ExOS = (ESN, EON, σ, EM, R) be an extended object system and $t \in$ T' Let EOS* be the set of all elementary object systems. A rewriting rule $r \in$ R assigned to *t* is one of the next three kinds of relations:

1. *r*: EON $\rightarrow$ EOS*

2. *r*: EON $\times$ EON $\rightarrow$ EOS*

3. *r*: EOS* $\rightarrow$ EON $\times$ EON

When *t* occurs, the following marking EM' is defined as follows:

1. Let $r(EOS_1) = EOS_2$, then every place p $\in$ t• receives (SN_2, M_2) and (SN_1, M_1) is removed from *p*, where *p* is the only input place of *t*.

2. Let $r(EOS_1, EOS_2) = EOS_3$, and let $\{p_1, p_2\} =$ •t ($p_1 \neq p_2$ is not forced). (Without loss of generality) where $(SN_1, M_1) \in ME(p_1)$ and $(SN_2, M_2) \in ME(p_2)$, then every place $p \in$ *t*• receives (SN_3, M_3) and (SN_1, M_1) is removed from p_1 and (SN_2, M_2) is removed from p_2.

3. Let $r(EOS_1) = (EOS_2, EOS_3)$, and let $\{p_1, p_2\} =$ *t*• ($p_1 \neq p_2$ is not forced). Then (without loss of generality) p_1 receives (SN_2, M_2) and p_2 receives (SN_3, M_3), and (SN_1, M_1) is removed from *p*, where *p* is the only input place of *t*.

Definition 15. Occurrence rule in extended object systems

Let ExOS = (ESN, EON, σ, EM, R) be an extended object system, EM:$P \rightarrow 2^{Eobj}$ an EO-marking, $t \in$ T', $e \in T_i$ and $r \in$ R assigned to *t*.

a) *t* is activated in EM(denoted by $EM \rightarrow_t$) if $t\sigma = \varnothing$ and r is appropriate to *t*, i.e. *r* is defined to t (as in definition 14) and the parameters of *r* are satisfied by EM. If *t* is activated, then t may occur ($EM \rightarrow_t EM'$) and the following marking EM' is defined as in definition 14.

b) A pair $[t, e] \in T' \times T_i$ with $t\sigma e$ is activated in EM (denoted by: $EM \rightarrow_{[t, e]}$) if in addition to case a), e is also activated for SN_i in M_i. Instead of (SN_i, M_i) the modified object $(SN_i\ M_{i+1})$ where $M_i \rightarrow_e M_{i+1}$, is added, i.e. the following marking of SN_i is modified according to the EOS occurrence rule. In this case r is applied to (SN_i, M_{i+1}).

c) A transition $e \in T_i$ with $\sigma e = \varnothing$ is activated in EM (denoted by: $EM \rightarrow_e$) if for some place $p \in P'$, both $(SN_i, M_i) \in EM(p)$ and $M_i \rightarrow_e M_{i+1}$ holds. In the following marking M' the object (SN_i, M_i) is replaced by (SN_i, M_{i+1}), i.e. the following marking of SN_i is changed according to the EOS occurrence rule.

Restriction 2. In this context, a transition t is assigned an event e (possibly null). Therefore, t may occur if it is activated and e holds.

5.4 Mobility model

In this section, a formal specification and verification of the Mobility platform is performed. As mentioned in the introduction of this chapter, the Mobility platform involves modelling a region, a System Agent and mobile agents.

We will model the mobility platform as the region where agent systems interact. Let MP = (ESN, EON, σ, EM, R) be an extended object system that represents the mobility platform as defined in Figure 5.2.

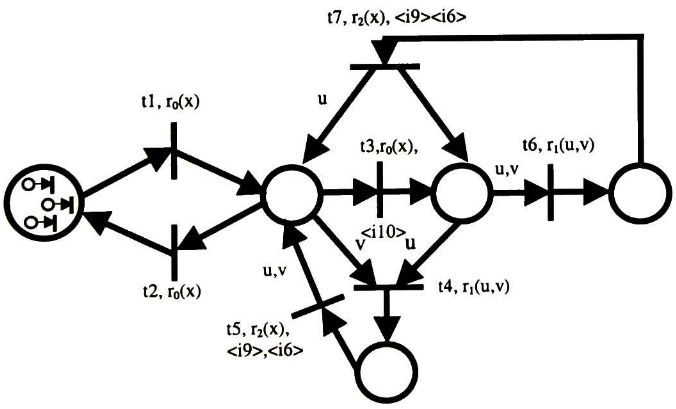


Figure 5.2 Region

Notice that $R = \{r_0(x), r_1(u,v), r_2(x)\}$, $EON = \{EOS_1,\dots, EOS_m\}(m \geq 1)$ and $EM(p1) = \cup\{(SN_i, M_i) | 1 \leq i \leq m\}$ is the initial marking. $EOS_i = (SN_i, ON_i, \rho_i, type_i, M_i)$ is an elementary object system that represents the Agent System as defined in Figure 5.3.

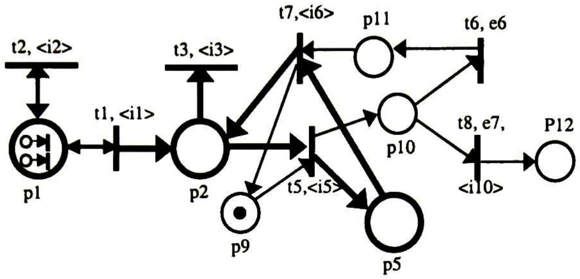


Figure 5.3 Agent system

$ON_i = \{ON_{i1}, ON_{i2},..ON_{in_i}\}$. At this point $\sigma = \{(t_3, t_{i8}) | 1 \leq i \leq m\}$ ($t_3 \in ESN$, $t_{i8} \in T_i$), $M_i(p1) = \cup\{(ON_{ij}, m_{ij}) | 1 \leq j \leq n_i\}$ $M_i(p_9)=1$ is the initial marking. In Figure 5.3 (and in all figures), type$(x, y) = \{1, 2,..n\}$ if the arc (x, y) is thick, type$(x, y) = 1$ in other case. $ON_{ij} = (B_{ij}, E_{ij}, F_{ij}, m_{0ij})$ is defined in Figure 5.4.

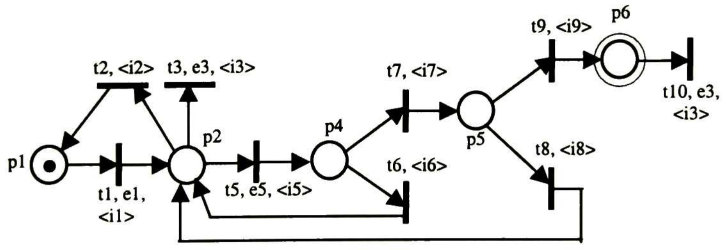


Figure 5.4 Mobile Agent

At this point $\rho = \{(t_1, t_{ij1}), (t_2, t_{ij2}), (t_3, t_{ij3}), (t_5, t_{ij5}), (t_7, t_{ij6}) | 1 \leq j \leq n_i\}$ where $t_{ijk} \equiv t_k \in E_{ij} \in ON_{ij} \in ON_i \in EOS_i \in EON \in MP$ ($k \in \{1,2,3,5,6\}$), $m(p1) = 1$ is the initial marking.

Finally, let's define the rewriting rules R={ $r_0(x)$, $r_1(u,v)$, $r_2(x)$} of MP.

- $r_0(x) = x$ is a relation of type 1.
- $r_1(u, v) = x$ is a relation of type 2 .
- $r_2(x) = u,v$ is a relation of type 3.

The rewriting rule r_0 is an identity relation. Figure 5.5 shows the consequence of applying r_1. For simplicity, the marking and the transition labels are omitted.

In order to understand the result of r_1, a new Petri Net representing the trading protocol in the transfer process between two agent systems is depicted in Figure 5.6. The net in Figure 5.6 is not defined formally but only used as an illustration.

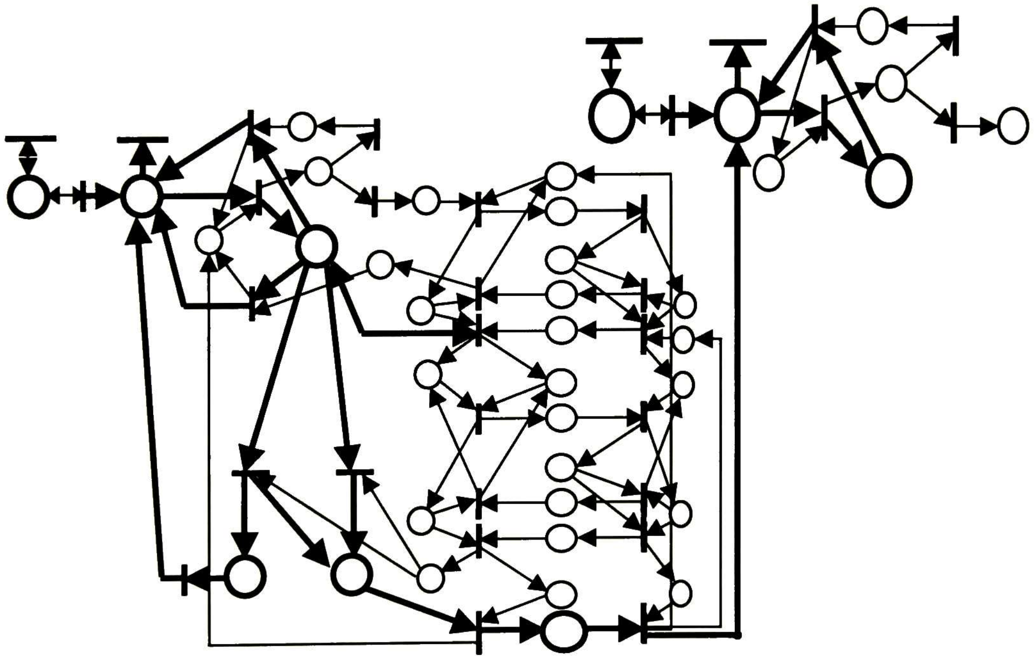


Figure 5.5 Result of applying r_1

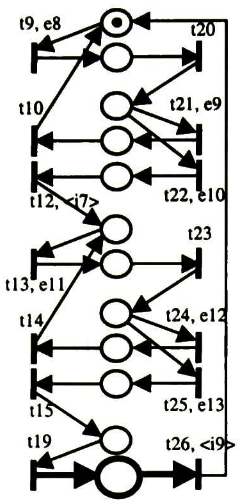


Figure 5.6 Trading protocol

The consequence of applying r_1 is the model resulting from the agent system to agent system interconnection as shown in figure 4.6. Two agent systems are modeled by negotiating the agent transfer. The agent system in Figure 5.3 is extended in a way to accept such negotiation. Figures 5.7 and 5.8 depict such extension for the source agent system (the one storing the agent to migrate) and the destiny agent system.

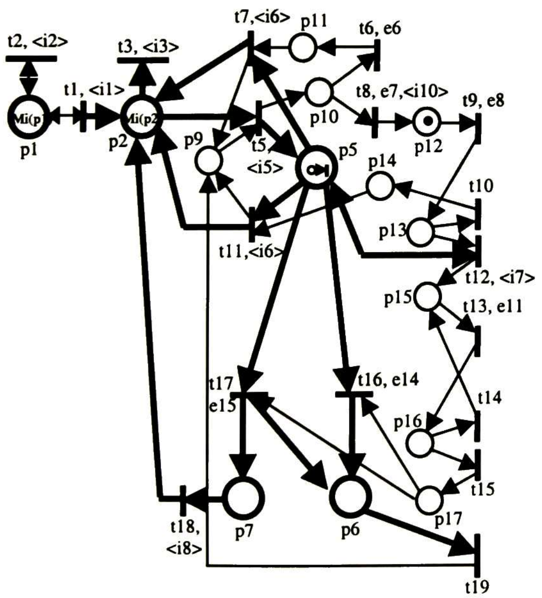
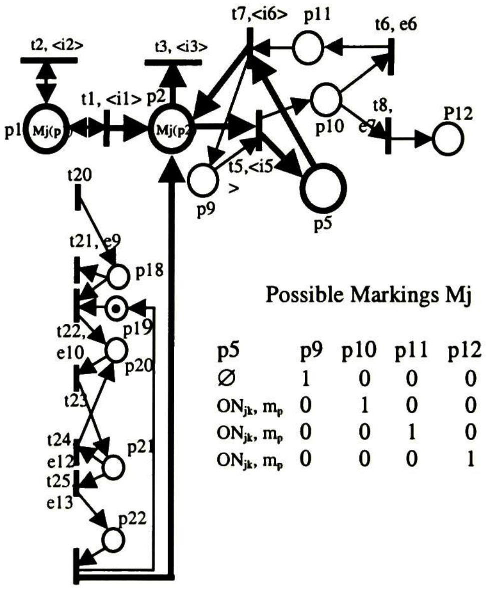


Figure 5.7 Source agent system



Possible Markings Mj

p5	p9	p10	p11	p12
∅	1	0	0	0
ON_{jk}, m_p	0	1	0	0
ON_{jk}, m_p	0	0	1	0
ON_{jk}, m_p	0	0	0	1

Figure 5.8 Destination agent system

Let *u* be a Source Agent System, v a Destiny Agent System, and N a Trading protocol; *u* and *v* are fused into N as follows: transitions from *u* (or *v*) and N having the same name become one, joining the pre-sets and the post-sets. Figure 5.9 shows how the final net is generated (net performing the agent transfer) from *u*, N and *v* (omitting again marking and transition labels) with no transition fusion.

The rewriting rule r_2 performs a function similar to the inverse function, i.e. $r_2(r_1(u, v)) = \{u', v'\}$, where u' and v' are nets like the one in Figure 5.3, but keeping the marking that was reached during the negotiation.

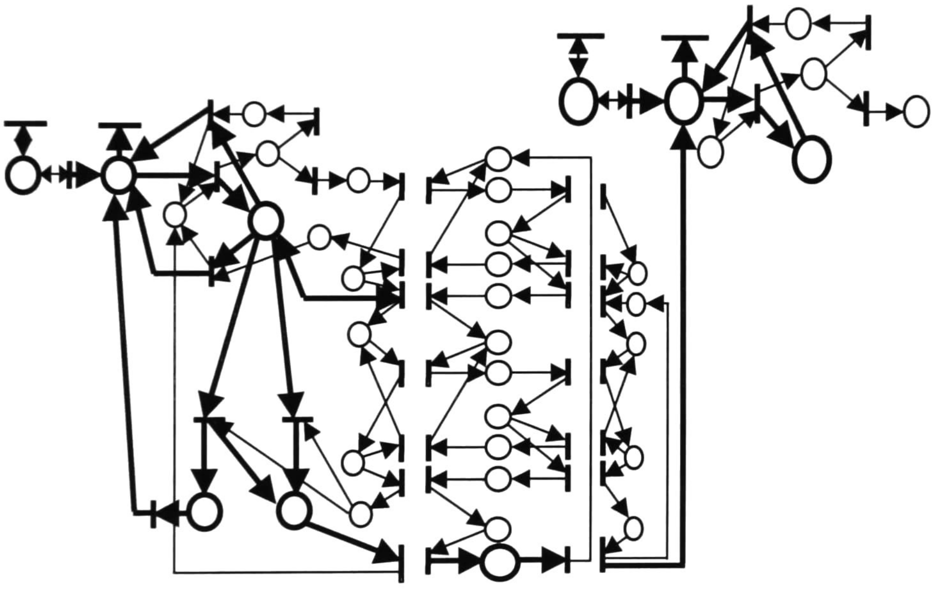


Figure 5.9 Fusion of *u*, N, *v*

The description of the events (labels) assigned to transitions are the next:

Mobile Application events:

e_1: beginning of the application

e_3: end of the application in the host where it belongs

e_5: request of mobility from the application

Source host events:

e_6: destiny host is not available (mobility rejection)

e_7: destiny host is available

e_8: request of application transfer

e_{11}: connection proposal

e_{14}: application mobility

e_{15}: application clonation and clone mobility

Destination host events:

e_9: Application reception denied.

e_{10}: Application reception accepted

e_{12}: Connection proposal rejected.

e_{13}: Connection proposal accepted

The mobile agents share the same initial behaviour (modeled in Figure 5.4) allowing the mobility only once. The place p_6 from Figure 5.4 performs the specific behaviour of every mobile agent and, therefore, p_6 may be replaced by another Petri Net modelling the particular behaviour of the mobile agent; with an adequate construction using the facilities provided by the mobility platform, mobility can be reached as many times as necessary.

5.5 Mobility platform liveness proof

In this section we prove that the mobility platform is live. The boundness prove is omitted because is easy to see that every place of the nets are bounded except p_2 in SN_i.

We prove that the mobility platform is live using coverage trees and the possible occurrence sequences. The building of coverage trees is based on definitions given in section 5.3.

Proof

Assumption 1. Suppose that if $t_k \in B_{ij}$ is activated in ON_{ij}, then it will eventually occur, i.e., t_j and t_k will be activated, where $t_j \rho t_k$ there exist, $t_j \in T_i \in SN_i \in EOS_i$ and $t_k \in B_{ij} \in ON_{ij} \in ON_i \in EOS_i$ (abusing a little of the language).

Figure 5.10 shows the ON_{ij}(net of Figure 5.4) coverage tree. Note that there exist a path that leads from m_0 to the dead marking m_5, only if the interaction relation $<i_3>$ ensues with SN_i. But $<i_3>$ in SN_i is assigned to $t_3 \in T_i$, which when it occurs it misses the token. But this token is ON_{ij} for definition. Therefore, we can think ON_{ij} as if it had a life cycle, which finalizes when the interaction relation $<i_3>$ ensues. Therefore, if assumption 1 holds, then we can say ON_{ij} is dead-lock free, even, it is like a life net during its life cycle.

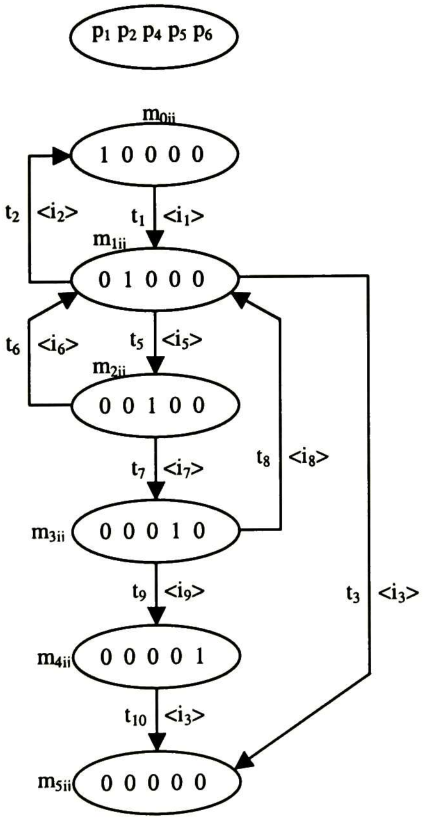


Figure 5.10 ON_{ij} coverage tree

Now, we prove that the nets $SN_i \in EOS_i$ are live and that the assumption 1 holds.

Assumption 2. Suppose that if t_k is activated in SN_i(where $t_k \in T_i$ and $t_k \in \sigma$), then it will eventually occur, i.e., t_j will be activated, where $t_j \sigma t_k$ exists and $t_j \in T' \in$ ESN (abusing a little of the language).

We will make a few simple assumptions, which do not affect the system proof, because of the particular interaction between the entities that constitute it. The main objective of the following assumptions is to reduce the coverage trees.

1. Suppose that a transition t with an event e = null occurs immediately when it is activated.
2. We consider nets SN_i where $|M_{0i}(p1)| = 1$ (with only one object ON_{ij} in p1).
3. Let $EOS_i = (SN_i, ON_i, \rho_i, type_i, M_i)$ be an elementary object system and let $ON_{ij} = M_{0i}(p1)$. There will exist only one copy of ON_{ij} in SN_i. This is possible and doesn't affect the system proof because an agent system can compute the transfer of only one mobile agent at the same time.

Figure 5.11 shows the SN_i(net of Figure 5.3) coverage tree. When SN_i reaches the marking M_5, then it is sure that the net SN_j exists, where SN_j represents the destiny agent system (in this case SN_i represents the source agent system). Therefore, it waits for the r_1 result ($r_1 \in R$).

The consequence of applying r_1 to two agent systems is the model resulting from the agent system to agent system interconnection; this is shown in Figure 4.6 and modeled in Figure 5.5. This last Figure model two agent systems negotiating the agent transfer. Therefore, two coverage trees are fused too.

During the negotiation of the agent transfer, the marking of ON_{ij} has few changes (and so it has a small interaction relation), therefore, now we show the possible occurrence sequences instead of the new coverage tree.

Figure 5.12 shows the possible occurrence sequences (of the new net generated by r_1) from M_5. The marking shown under a transition (transition that produces such marking) of an occurrence sequence, is the marking of the sub-net generated by the nodes of the net shown in Figure 5.3.

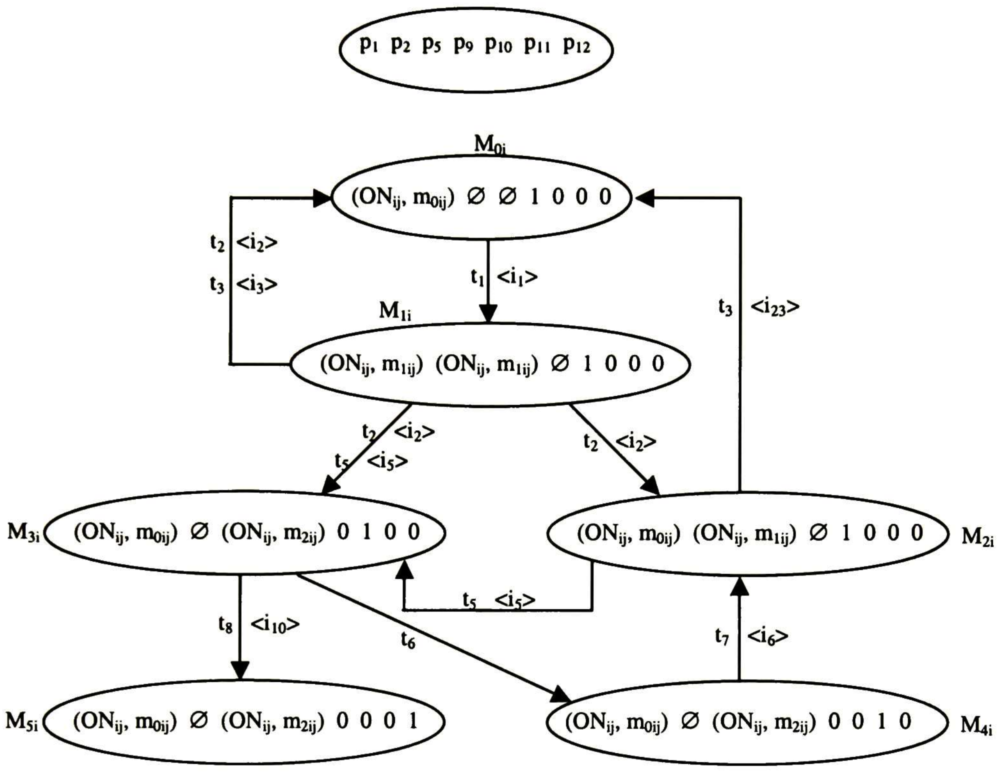

p1 p2 p5 p9 p10 p11 p12
M0i
(ONij, m0ij) Ø Ø 1 0 0 0
M1i
(ONij, m1ij) (ONij, m1ij) Ø 1 0 0 0
M2i
(ONij, m0ij) (ONij, m1ij) Ø 1 0 0 0
M3i
(ONij, m0ij) Ø (ONij, m2ij) 0 1 0 0
M4i
(ONij, m0ij) Ø (ONij, m2ij) 0 0 1 0
M5i
(ONij, m0ij) Ø (ONij, m2ij) 0 0 0 1
t1 <i1>
t2 <i2>
t3 <i3>
t3 <i23>
t5 <i5>
t6
t7 <i6>
t8 <i10>

Figure 5.11 SN_i coverage tree

In the case 1 of the Figure 5.12, when transition t_{11} occurs, the rewriting rule r_2 also occurs, and the net SN_i returns to the marking M_{2i}. The marking of SN_j is any from Figure 5.11. In case 2, when transition t_{26} occurs, the rewriting rule r_2 also occurs, and SN_i returns to the initial marking M_{0i} and the marking of SN_j is M'_j, where $M'_j(p_k) = M_{qj}(p_k)$, $k \in P - p_2$ (P is the set of places in SN_j), where M_{qj} is any from Figure 5.11, and $M'_j(p_2) = M_{qj}(p_k) \cup (ON_{ij}, m_{4ij})$. This signifies that the mobile agent ON_{ij} migrates from SN_i to SN_j. Cases 3 and 4 are similar to case 2.

Thus, it is easy to see that SN_i is live in all cases, representing the source agent system and representing the destination agent system. Therefore, if assumption 2 holds, we can say that SN_i is live.

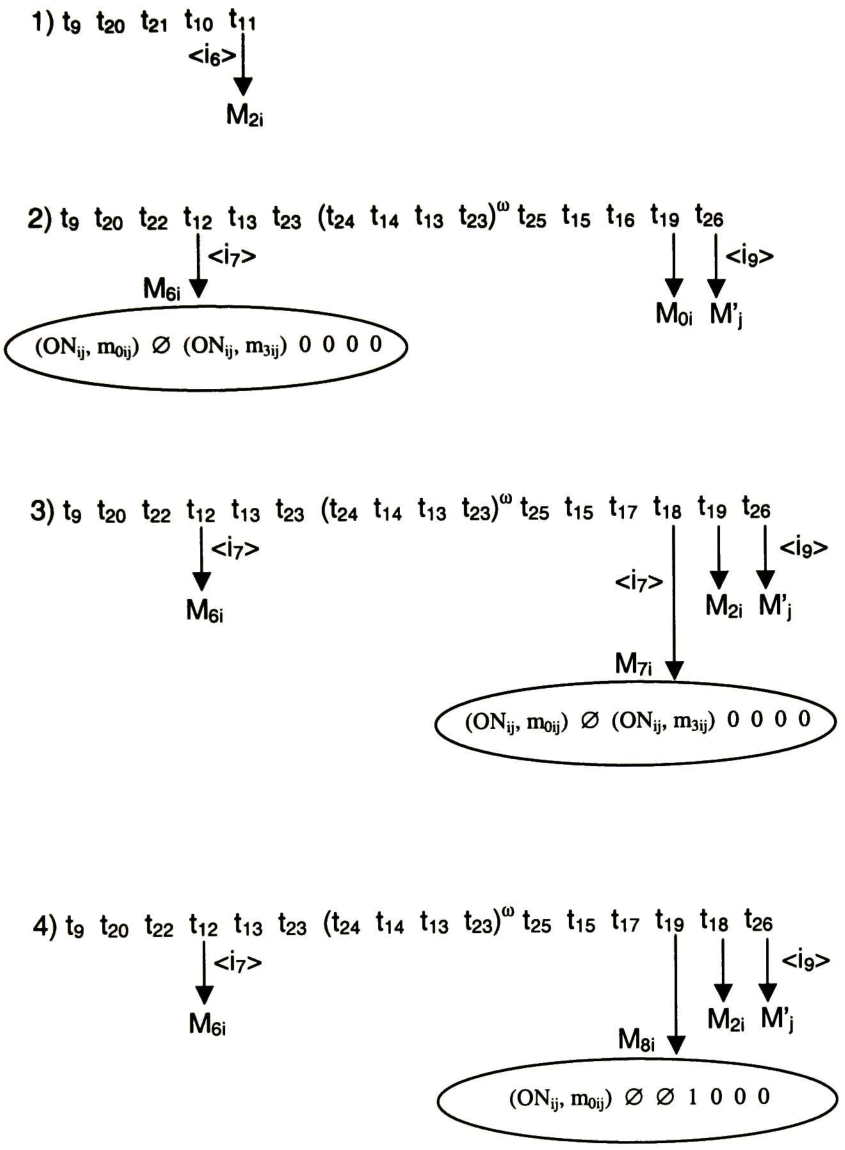

1) t9 t20 t21 t10 t11
<i6>
M2i
2) t9 t20 t22 t12 t13 t23 (t24 t14 t13 t23)ω t25 t15 t16 t19 t26
<i7>
M6i
<i9>
M0i M'j
(ONij, m0ij) ∅ (ONij, m3ij) 0 0 0 0
3) t9 t20 t22 t12 t13 t23 (t24 t14 t13 t23)ω t25 t15 t17 t18 t19 t26
<i7>
M6i
<i7>
<i9>
M2i M'j
M7i
(ONij, m0ij) ∅ (ONij, m3ij) 0 0 0 0
4) t9 t20 t22 t12 t13 t23 (t24 t14 t13 t23)ω t25 t15 t17 t19 t18 t26
<i7>
M6i
<i9>
M2i M'j
M8i
(ONij, m0ij) ∅ ∅ 1 0 0 0

Figure 5.12 Possible occurrence sequences from M_5

We can see that the interaction relations in ON_{ij} are possible in the occurrence sequences shown in Figure 5.12. Therefore, if assumption 2 holds, then assumption 1 holds too.

Now, we prove that MP is live and that assumption 2 holds. We can see that every rewriting rule $r \in$ R assigned to a transition $t \in$ T' is appropriate, and ESN is live and bounded in its structure. Therefore, to conclude it is necessary to prove that assumption 2 holds.

The unique interaction relation sequence in ESN is: $<i_{10}>$ $<i_9>$ $<i_6>$. This sequence is possible in every SN_i. Therefore, the assumption 2 holds.

5.6 Conclusions

In this chapter, we have proved that the Mobility of GeDA-3D we propose is a live system using an extension we propose of Petri Nets.

Chapter 6

Adaptive behaviour in virtual entities

6.1 Objective

This chapter presents the evolutionary algorithms and adaptive behaviors that are proposed, those are used for the virtual entities participating on a strategy game (Virtual Combat Environment) that was implemented and successfully embedded into GeDA-3D.

6.2 Introduction

We propose some evolutionary algorithms which are employed to assign adaptive behaviors to virtual entities. We employed evolutionary algorithms belonging to two of the main paradigms in evolutionary computation: Genetic algorithms and evolution strategies. Each of these paradigms was originated independently and they have different motivations. These algorithms imitate the principles of natural evolution as a method to solve parameter optimization problems.

We employed these paradigms for the reason that they are appropriate to accomplish searches within big search spaces, and because they impose few restrictions of mathematical type on the form of the function that we pretend to optimize. Moreover, the application of these algorithms (case of study) fulfills the following characteristics:

- The search space (i.e., possible solutions) is limited.
- It is possible to define an objective function.
- It is possible to define a fitness function in the case of genetic algorithms.
- It is possible to encode the solutions in a correct form.

The case of study is a strategy game that was integrated to GeDA-3D and it consists of a set of virtual entities participating in a dynamic environment. The virtual entities have an adaptive behavior that is especially governed by one of the evolutionary algorithms proposed. In section 6.3 we describe what is involved on the case of study. In section 6.4 we introduce and carefully describe the evolutionary algorithms employed, and in section 6.5 we describe how such algorithms are employed by the virtual entities in the case of study. In section 6.6 we show some results obtained in the run of those evolutionary algorithms.

6.3 Case of study

A virtual editor capable to generate one kind of environment was implemented and successfully embedded into GeDA-3D. Such a editor generates a virtual combat environment ant it allows users to assign a specific adaptive behavior to the virtual entities participating in the combat. As a consequence, these entities perform changes to the environment, and therefore, we have entities participating in a dynamic environment.

The game consists of a Virtual Combat Environment(VCE) where a number of empires fight against each other for power. Every empire is constituted by a set of soldiers and miners commanded by a king, each having a virtual representation in the VCE. The characteristics of the game are explained below.

- The entities (agents) participating in the VCE are capable of performing the following actions:
    - *King* : Nothing, creates an agent (soldier or miner), moves or changes the empire strategy.
    - *Soldier*: Nothing, moves, attacks or defends himself from a near enemy.
    - *Miner*: Nothing, moves or extracts gold.

- The VCE contains certain amount of gold (located at different areas into the environment), trees and big rocks (in addition to the participant agents).
- Every unit of gold extracted is used to add a new agent to the empire.
- A miner is capable of extracting a unit of gold located by his side.
- A soldier is capable of attacking an enemy located by his side.
- The agents can not move over the trees and rocks.
- The agents spend the same time executing any action except for the action of moving.
- The agents travel at different constant speeds: A soldier is faster than a miner, and a miner is faster than a king.
- The game is divided by execution cycles, where every agent may perform at most one action.
- When the king is attacked, the agents commanded by him can no longer perform actions.
- The game is over as soon as only one king is left alive.
- If one soldier attacks an enemy miner (or king), then the miner (or king) dies.
- If one soldier attacks an enemy soldier who:
    - attacks him back, then both soldiers die.
    - moves to another place, then the enemy dies.
    - defends himself from him, then no one die.
    - defends himself from another soldier, then the enemy dies.
    - performs no actions, then the enemy dies.

6.4 Algorithms employed

We employed adaptive algorithms belonging to the paradigms of Genetic algorithms and evolution strategies. In this section, we introduce and carefully describe the algorithms employed.

First, a short introduction to the basic terminology concerning the parameter optimization problem is given. The overall goal of a parameter optimization problem $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, $M \neq \varnothing$, where f is called the *objective function*, is to find a vector $\chi^* \in M$ such that: $\forall \chi \in M : f(\chi) \geq f(\chi^*) = f^*$, where f^* is called a *global minimum*; χ^* is the *minimum location* (point or set). $M = \{ \chi \in \mathbb{R}^n \mid g_j(\chi) \geq 0 \ \forall j \in \{1,\ldots,q\}\}$ is the set of feasible points for a problem with inequality constraints $g_j : \mathbb{R}^n \rightarrow \mathbb{R}$.

Since $\max\{ f(\chi) \} = -\min\{ -f(\chi) \}$, the restriction to minimization is taken without loss of generality. In general the optimization problem is complicated by the existence of non-linear objective functions with multiple local optima. A *local minimum* $f' = f(\chi')$ is defined as: $\exists \omega > 0 \ \forall \chi \in M : ||\chi - \chi'|| < \omega \Rightarrow f' \leq f(\chi)$.

Even if there is only one local optimum, it may be difficult to find a path towards it in case of discontinuities in the objective function or its derivatives.

6.4.1 Genetic algorithms

Genetic algorithms (GAs) are adaptive methods which may be used to solve search and optimization problems. A genetic algorithm transforms a population (set) of individual objects (chromosomes), each with an associated fitness value, into a new generation of the population using the Darwinian principle of reproduction and survival of the fittest and it analogs naturally occurring genetic operations such as crossover (sexual recombination) and mutation.

Each individual in the population represents a possible solution to a given problem. The genetic algorithm attempts to find a very good (or the best) solution to the problem by genetically breeding the population of individuals over a series of generations.

The algorithm is started with a set of solutions (represented by chromosomes) called a population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. Solutions which are selected to form new solutions (offspring) are selected according to their fitness - the more the suitable they are, then the more the chances that they have to reproduce. This process is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied. Figure 6.1 shows the basic genetic algorithm.

1. [Start]Generate random population of n chromosomes (suitable solutions for the problem)
2. [Loop] Iteraively perform the following substeps on the population until the termination criterion has been satisfied:
    a. [Fitness] Assign a fitness value to each individual in the population using the fitness measure.
    b. [New population] Create a new population of chromosomes by applying the following three genetic operations. The genetic operations are applied to individual in the population chosen with a probability based on fitness.
        i. [Selection] Reproduce an existing individual by copying it into the new population.
        ii. [Recombination] With a recombination probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents. The offspring take the place of their parents into the new population.
        iii. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome).
3. [Solution]The chromosome that is identified by the method of result designation (e.g., the best-so-far individual) is designated as the result of genetic algorithm for the run. The result may represent a solution (for an approximate solution) to the problem.

Figure 6.1 The basic genetic algorithm

The GA can be described as the following 10-tuple:

$$GA = (P^0, p, pr, pm; F, s, r, m; f, g, T)$$

where

P^0	=	$(a_1^0, \dots, a_p^0) \in I^p$	population I = $\mathbb{R}^n$
p	$\in$	$\mathbb{N}$	number of individuals
pr	$\in$	$\mathbb{R}$	recombination probability
pm	$\in$	$\mathbb{R}$	mutation probability
F	:	$I \rightarrow \mathbb{R}$	Fitness function
s	:	$I^p \rightarrow I^p$	selection operator
r	:	$I \times I \rightarrow I \times I$	recombination operator
m	:	$I \rightarrow I$	mutation operator

f	:	$\mathbb{R}^n \rightarrow \mathbb{R}$	objective function
g_j	:	$\mathbb{R}^n \rightarrow \mathbb{R}$	constraint function $j \in \{1,\dots,q\}$
T	:	$I^p \rightarrow \{0,1\}$	termination criterion

P^0 denotes the initial population consisting of a set of initial chromosomes. Chromosomes are selected from the population to be parents to crossover. According to Darwin's evolution theory the best ones should survive and create a new offspring. There are several methods to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and many others[OBITKO98]. After that, the selection operator selects individuals to become the members of the next generation:

$$P^{t+1} \quad = \quad s(P^t)$$

$$s(a_i^t) \quad = \quad \text{true} \quad \text{if } F(a_i^t) > \text{X. where X is a random number.}$$

Note that an individual may be selected more than once (s : $I^p \rightarrow I^p$). The recombination operator selects genes from parent chromosomes and creates a new offspring.

$$r(a_i^t, a_j^t) = a'^t_i \ , a'^t_j \ \in \mathbb{R}^n \times \mathbb{R}^n$$

$$a'^t_{i\mathrm{k}} = \begin{cases} a^t_{i\mathrm{k}} \ , \mathrm{k} \leq \mathrm{C} \\ a^t_{j\mathrm{k}} \ , \mathrm{k} > \mathrm{C} \end{cases} \quad \forall \mathrm{i} \in \{1,..,\mathrm{n}\}$$

$$a'^t_{j\mathrm{k}} = \begin{cases} a^t_{j\mathrm{k}} \ , \mathrm{k} \leq \mathrm{C} \\ a^t_{i\mathrm{k}} \ , \mathrm{k} > \mathrm{C} \end{cases} \quad \forall \mathrm{i} \in \{1,..,\mathrm{n}\}$$

Where C is a randomly crossover point. The mutation operator is applied to all components of the object parameter a_i^t. According to the biological observation that offsprings are similar to their parents and that smaller changes occur more often than larger ones.

Mutation is realized in the following way:

$$m(a'^t_{\mathrm{i}}) \quad = \quad a''^t_{\mathrm{i}}$$

$$a''^t_{\mathrm{ik}} = \begin{cases} a'^t_{i\mathrm{k}} \quad 1 & pm > \mathrm{C}_{1\mathrm{k}} \wedge \mathrm{C}_{2\mathrm{k}} \leq \tfrac{1}{2} \\ a'^t_{i\mathrm{k}} + 1, & pm > \mathrm{C}_{1\mathrm{k}} \wedge \mathrm{C}_{2\mathrm{k}} > \tfrac{1}{2} \\ a'^t_{i\mathrm{k}} & \text{else} \end{cases} \quad \forall \mathrm{i} \in \{1,..,\mathrm{n}\}$$

Where C_{1k} and C_{2k} are two random numbers and they are sampled anew for each component of a''^{t}_{ik}.

Figure 6.2 shows the algorithm previously defined

```
t ← 0;
initialize $P^0 \leftarrow \{a_1^0,\dots, a_p^0\} \in I^p$; such that $\forall j : g_j(a_i^0) \geq 0$;
while termination criterion not fulfilled do
begin
    select $P^{t+1} := s(P^t)$:
        i ← 0;
        while(i < $p$) do
        begin
          X = random;
          for($j$←0; $j < p$; $j \leftarrow j$+1)
          begin
            if($s(a_i^t)$) then i++;
            if(i = $p$) break;
          end
        end
    recombine $P^{t+1}$;
        for($j$←0; $j < p$; $j \leftarrow j$+2)
        begin
          if($pr$ > random)
          begin
            C ← random(n);
            $r(a_j^t, a_{j+1}^t)$;
          end
        end
    mutate $P^t$ : $a''^t_k = m(a'^t_k)$
    $P^t \leftarrow P^{t+1}$
    t ← t + 1;
end
```

Figure 6.2 Genetic Algorithm

6.4.2 Evolution Strategies

We employed two kind of algorithms belonging to the evolution strategies: the two-membered ES ((1+1-ES)) algorithm and multimembered ES algorithms (($\mu+\lambda$)-ES and (μ, λ)-ES)[BACK].

(1+1) - ES

The two-membered ES is a simple mutation-selection scheme. It is based upon a population consisting of a one parent individual (a real-valued vector), and one descendant, created by means of adding normally distributed random numbers. The better from both individuals then serves as the ancestor of the following generation. Such a (1+1)-ES can be described as the following 8-tuple:

$$(1+1)\text{-ES} = (P^0, m, s, c_d, c_i, f, g, T)$$

where

P^0	=	$(x^0, \sigma^0) \in I$	population $I = \mathbb{R}^n \times \mathbb{R}^n$
m	:	$I \rightarrow I$	mutation operator
s	:	$I \times I \rightarrow I$	selection operator
c_d, c_i	$\in$	$\mathbb{R}$	step-size control
f	:	$\mathbb{R}^n \rightarrow \mathbb{R}$	objective function
g_j	:	$\mathbb{R}^n \rightarrow \mathbb{R}$	constraint function $j \in \{1,\dots,q\}$
T	:	$I \times I \rightarrow \{0,1\}$	termination criterion

P^0 denotes the initial population consisting of a single parent which produces, by means of mutation, a single offspring resulting in:

$$\begin{aligned} P'^t &= (a'^t_1, a'^t_2) \in I \times I \\ a'^t_1 &= P^t = (x^t, \sigma^t) \\ a'^t_2 &= m(P^t) = (x'^t, \sigma^t) \end{aligned}$$

The mutation operator is applied to all the components of the object parameter x^t. According to the biological observation that offsprings are similar to their parents and that smaller changes occur more often than larger ones, mutation is realized by normally distributed random numbers:

$$x'^t = x^t + N_0(\sigma^t)$$

where N_0 denotes a vector of independent Gaussian random numbers with zero mean and standard deviations σ_i^t ($i = 1,\dots,n$). For theoretical considerations, all components of $\sigma^t \in \mathbb{R}$ are identical, i.e., $\forall ij\{1,..,n\} : \sigma_i^t = \sigma_j^t =: \sigma$. The selection operator then determines the fitter individual to become the parent of the next generation:

$$P^{t+1} = s(P'^t) = \begin{cases} a'^t_2 \text{ if } f(x'^t) \leq f(x^t) \wedge g_j(x'^t) \geq 0 \ \forall j \in \{1,\dots,q\} \\ a'^t_1 = P^t \text{ else} \end{cases}$$

In the case of minimization, the iteration process $P^t \rightarrow P^{t+1}$ stops when the termination criterion $T(a'^t_1, a'^t_2) = 1$ holds. Function T depends on the implementation and may utilize elapsed CPU time, elapsed number of generations, absolute or relative progress per generation etc. In our implementation T utilizes elapsed number of generations.

Although, in general, problems of interest may have different characteristics, the following heuristic often helps to dynamically adjust σ^t. Hence, the mutation operator m is extended by the following equation(see Rechenberg 1/5 rule in chapter 3):

$$\sigma^{t+n} = \begin{cases} c_d \cdot \sigma^t & \text{if } p_s^t < 1/5 \\ c_i \cdot \sigma^t & \text{if } p_s^t > 1/5 \\ \sigma^t & \text{if } p_s^t = 1/5 \end{cases}$$

Where p_s^t is the frequency of successful mutations, measured for example over intervals of 10n trials. Schwefel [SCHWEFEL81] gives reasons to use the factors $c_d = 0.82$ and $c_i = 1/0.82$ for the adjustment, which should take place every n mutations. Note that m consists of a random (definition of m) and a deterministic component (extension of m), now.

Figure 6.3 shows the algorithm previously defined:

```
t ← 0;
initialize $P^0 \leftarrow \{x^0\}$; such that $\forall j : g_j(x^0) \geq 0$;
while termination criterion not fulfilled do
begin
        mutate $P^t : x'^t = x^t + N$
                where
                        $Ni = 1/sqrt(2\pi)\ exp(-((X_i - X_i b)^2)/2\sigma^t)$;
                        $X_i b=0$ and $\sigma$ are mean and standard deviation, respectively.
                        X is a vector of random numbers.
        evaluate $P^t : f(x^t), f(x'^t)$;
        select $P^{t+1}$ from $P^t$:
                if $f(x'^t) \leq f(x^t) \wedge g_j(x'^t) \geq 0\ \forall j \in \{1,\ldots,q\}$ then
                        $x^{t+1} \leftarrow x'^t$
                else
                        $x^{t+1} \leftarrow x^t$
        t ← t + 1;
end
```

Figure 6.3 Algorithm (1+1)-ES

$(\mu + \lambda)$-ES

As the nomenclature $(\mu + \lambda)$-ES suggests, μ parents produce λ offsprings which are reduced again to the μ parents of the next generation. In $(\mu + \lambda)$-ES all population undergo selection.

$$(\mu, \lambda)\text{-ES} = (P^0, \mu, \lambda; r, m, s; \Delta\sigma, f, g, T)$$

where

P^0	=	$(a_1^0, \ldots, a_\mu^0) \in I^\mu$	population $I = IR^n \times IR^n$
μ	$\in$	IN	number of parents
λ	$\in$	IN	number of offspring ($\lambda > \mu$)
r	:	$I^\mu \rightarrow I$	recombination operator
m	:	$I \rightarrow I$	mutation operator
s	:	$I^{\mu+\lambda} \rightarrow I^\mu$	selection operator

$\Delta\sigma$	$\in$	$\mathbb{R}$	step-size meta-control
f	:	$\mathbb{R}^n \rightarrow \mathbb{R}$	objective function
g_j	:	$\mathbb{R}^n \rightarrow \mathbb{R}$	constraint function $j \in \{1,\dots,q\}$
T	:	$I^{\mu} \rightarrow \{0,1\}$	termination criterion

With the introduction of μ parents instead of only one, the imitation of sexual reproduction is possible, which is provided by the additional recombination operator r:

$$r(P^t) = a' = (x', \sigma') \in I,\ x' \in \mathbb{R}^n,\ \sigma' \in \mathbb{R}^n$$

$$x_i' = \begin{cases} x_{a,i}, & X \leq ½ \\ x_{b,i}, & X > ½ \end{cases} \qquad \forall i \in \{1,..,n\}$$

$$\sigma_i' = \begin{cases} \sigma_{a,i}, & X \leq ½ \\ \sigma_{b,i}, & X > ½ \end{cases} \qquad \forall i \in \{1,..,n\}$$

Where a =(x_a, σ_a), b =(x_b, σ_b) $\in I$ are two parents internally chosen by r. By convention, all parents in a population have the same mating probabilities, i.e., the parents a and b are determined by uniform random numbers. X denotes a uniform random variable on the interval [0, 1], and it is sampled anew for each component of the vectors x' and σ'.

σ^t is incorporated into the genetic information of an individual $a_1{}^t = (x^t, \sigma^t) \in I$ and is not controlled by some meta-level algorithm like the 1/5 success rule anymore. Consequently, it is subject to recombination and mutation as well. Those individuals with better adjusted strategy parameters are expected to perform better (better parameter settings will emerge by means of self-adaptation). As a result, the mutation operator not only works on x^t but also on σ^t.

$$a'^t_i = r(P^t)$$

$$m(a'^t_i) = a''^t_i = (x''^t, \sigma''^t)$$

$$\sigma''^t = \sigma'^t \exp N_0(\Delta\sigma)$$

$$x''^t = x'^t + N_0(\sigma''^t)$$

The selection operator s removes the λ - μ individuals with least fit from the offspring an the μ parents in the population.

$$P'^{t} = (a'^{t}_{1}, \ldots, a'^{t}_{\mu}, a'^{t}_{\mu+1}, \ldots, a'^{t}_{\mu+\lambda}) = (a_1^{t}, \ldots, a_\mu^{t}, m(r(P^{t})_1), \ldots, m(r(P^{t})_\lambda))$$

$$P^{t+1} = s(P'^{t}) \text{ such as } \forall a_i^{t+1} = (x, \sigma) \ \neg\exists a'^{t}_{j} = (x', \sigma') : f(x') < f(x)$$

Figure 6.4 shows the algorithm previously defined:

```
t ← 0;
initialize $P^0 \leftarrow \{a_1^0,\ldots, a_\mu^0\} \in I^\mu$; such that $\forall j : g_j(x_i^0) \geq 0$;
while termination criterion not fulfilled do
begin
        recombine $P^t : a'^{t}_{k} = r(P^t)\ \forall k \in \{1,.., \lambda\}$;
        mutate $P^t : a''^{t}_{k} = m(a'^{t}_{k})$
            where
                $\sigma''^{t}_{i} = \sigma'^{t}_{i} \exp(1/\text{sqrt}(2\pi) \exp(-((X_i - X_i b)^2)/2\Delta\sigma))$;
                $x''^{t} = x'^{t} + N$
                where $N_i = 1/\text{sqrt}(2\pi) \exp(-((X_i - X_i b)^2)/2\sigma''^{t}_{i})$;
                $X_i b = 0$ and $\sigma''^{t}_{i}$ are mean and standard deviation, respectively.
                X is a vector of random numbers.
        evaluate $P'^{t} := \{a''^{t}_{1},\ldots, a''^{t}_{\lambda}\}$;
            $\{f(x''^{t}_{1}),\ldots, f(x''^{t}_{\lambda})\}$;
        select $P^{t+1} := s(P'^{t})$:
        t ← t + 1;
end
```

Figure 6.4 Algorithm (1+1)-ES

6.5 Behavior of agents from the case of study

The editor is in charge of modifying the VCE according to the actions performed by the agents. In every cycle, the VCE state is sent to all the agents by the editor; these agents, in turn, send back an action (possibly null) to be displayed in the VCE, if valid. As soon as the editor receives all the actions, a new cycle begins. This is illustrated in Figure 6.5.

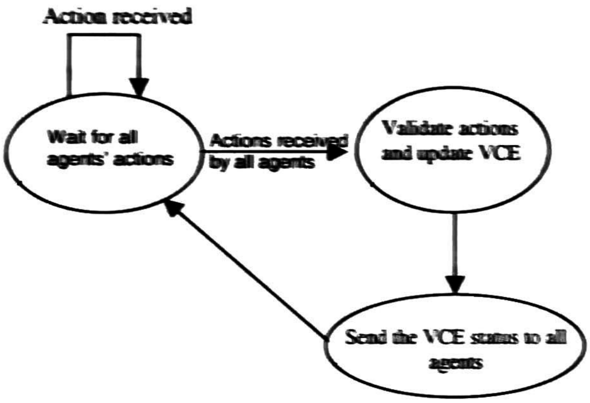

Action received
Wait for all agents' actions
Actions received by all agents
Validate actions and update VCE
Send the VCE status to all agents

Figure 6.5 Operation of the editor

There are some important characteristics of the case of study to be considered to design the adaptive behaviors:

- The state of the environment at time t may be different at time t+1. Therefore, an agent's plan consisting of a sequence of actions must be able to change dynamically.
- The agent's own actions affect the environment and the environment evolves independently of the agent.
- The environment is *accessible*, i.e., the agent has access to the complete state of the environment.
- The environment is *deterministic*, i.e., the next state of the environment is completely determined by the current state and the action selected by the agents.

A part of the behavior of each agent is completely deterministic because of the characteristics of the environment. And a part of the behavior of each agent can be resolved by evolutionary computation.

Evolutionary algorithms are employed to generate trajectories which are based in the strategy proposed by the king. A trajectory consists of a sequence of movement actions. There exists only three strategies: to attack, to flee and to extract.

To attack: In soldiers, the evolutionary algorithm generates a trajectory that leads from the agent's position to the enemy-king's position; in miners it generates a trajectory that leads from the agent's position to a place where there is gold (in this case, the evolutionary algorithm is used as a search algorithm too). In some cases, in soldiers, the evolutionary algorithms generate trajectories to make an ambush, i.e., surround a enemy.

To flee: In miners, the evolutionary algorithm generates a trajectory that leads from the agent's position to the empire's position; in soldiers it generates a trajectory that leads from the agent's position to the king's position (in this case, the soldiers try to protect the king).

To extract: In miners, the evolutionary algorithm generates a trajectory that leads from the agent's position to a place where there is gold (avoiding enemy's soldiers); in soldiers, the evolutionary algorithm generates a trajectory that leads from the agent's position to a miner's position (a soldier follows a miner to protect him).

The best trajectory is the shortest one, in which there are not obstacles in its route. We consider as obstacles the gold, trees, big rocks and the agents. Note that agents are mobile obstacles.

The king changes the strategy in base of some parameters: approximate number of enemy soldiers and miners, number of soldiers and miners of the same empire (friends), position of the enemy agents (this may guess the enemy's strategy) and the approximate quantity of gold in the environment. The change of the strategy is obtained from an evolutionary algorithm too. Such algorithm calculates each of these parameters and calculates the best strategy.

Figures 6.6, 6.7 and 6.8 show the behavior of the agents participating in the game. All agents belonging to the same empire have an adaptive behavior that is especially governed by one of the proposed evolutionary algorithms.

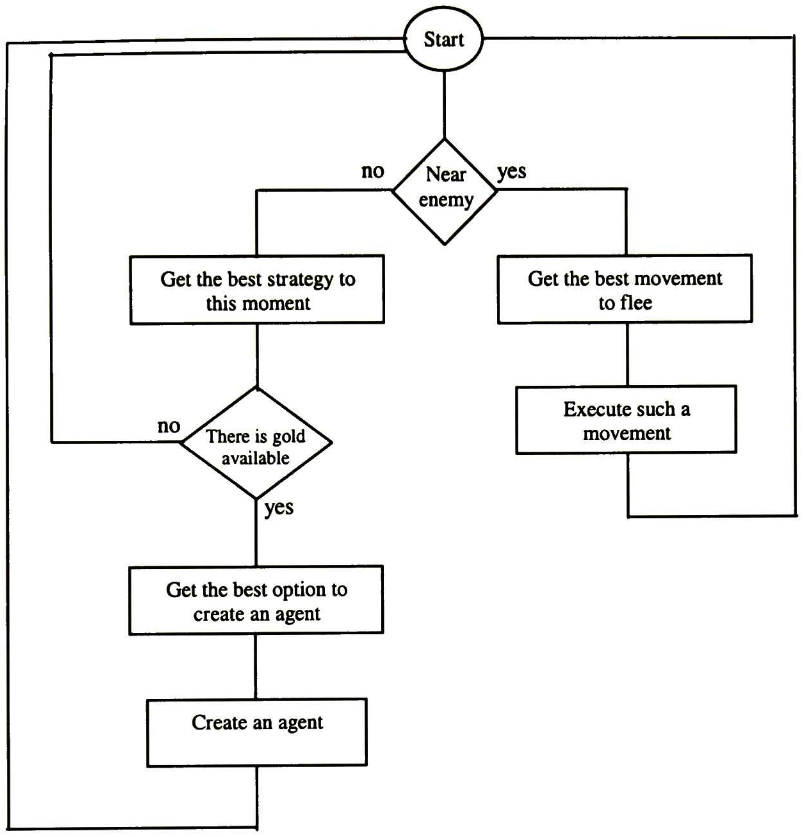

Start
Near enemy
no
yes
Get the best strategy to this moment
Get the best movement to flee
no
There is gold available
yes
Execute such a movement
Get the best option to create an agent
Create an agent

Figure 6.6 Behavior of the King

In each cycle (time t) of the game, the algorithm begins at the start point. In Figure 6.6, the evolutionary algorithms are used to determine the best movement of the king when an enemy is near, to determine the best strategy at time t and to determine the best option to create an agent: miner or soldier (in base of the same parameters as in the change of strategy). Note that the change strategy is an action that does not spend execution time.

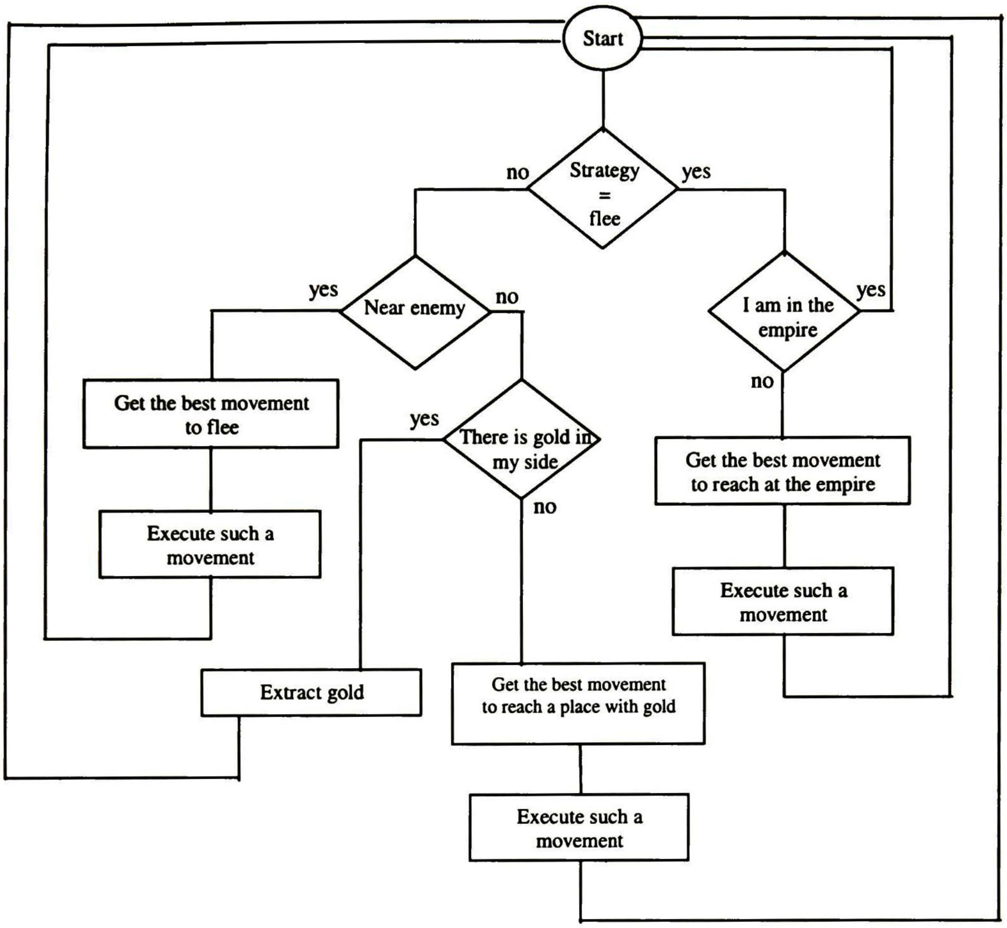


Figure 6.7 Behavior of a miner

In Figure 6.7, as in Figure 6.6, the evolutionary algorithms are used to determine the best movement of the miner when an enemy is near, to determine the best movement to arrive at a place where there is gold and to determine the best movement to arrive in the empire.

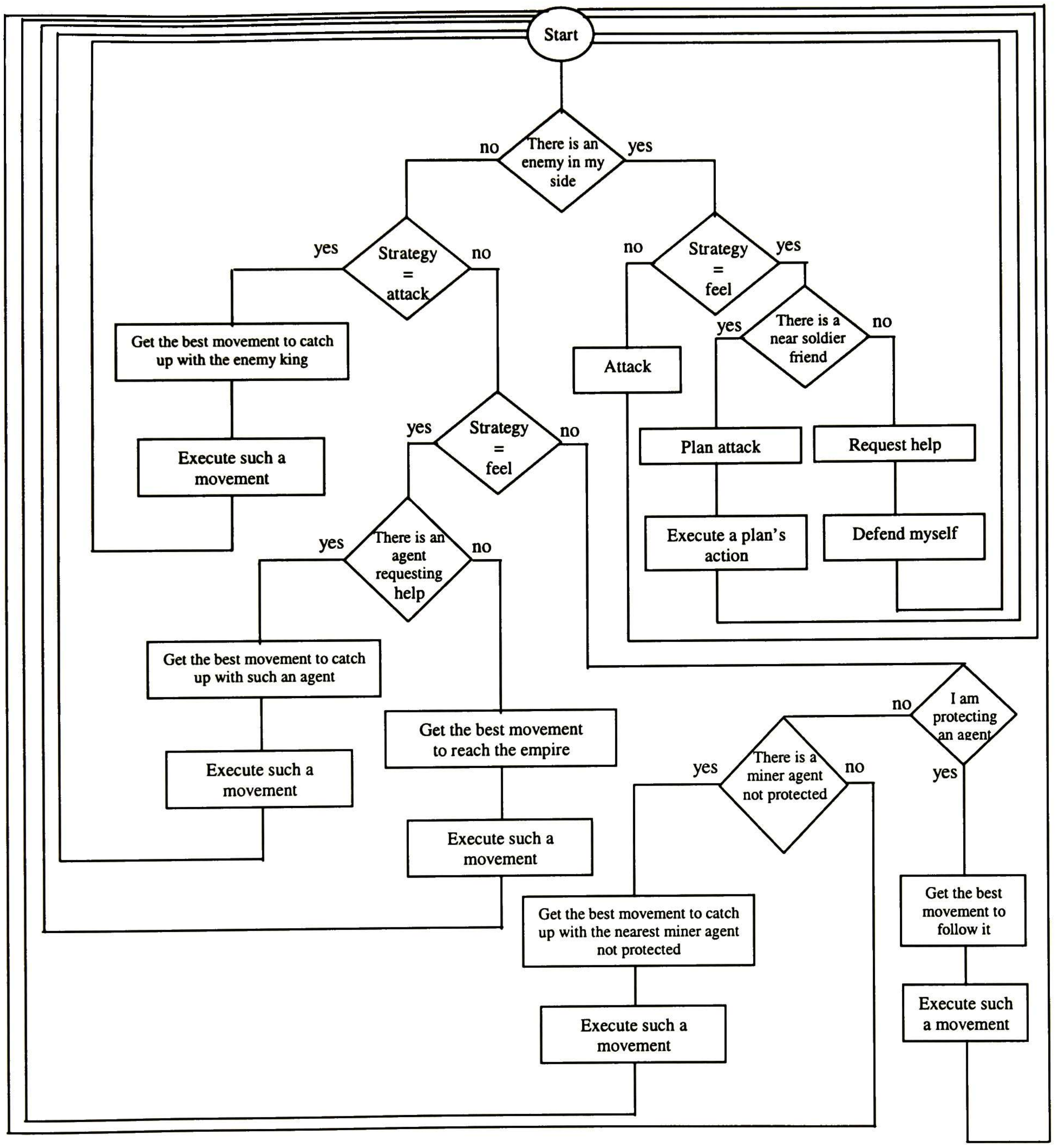

Start
There is an enemy in my side
no
yes
Strategy = attack
yes
no
Strategy = feel
no
yes
Get the best movement to catch up with the enemy king
Execute such a movement
Attack
There is a near soldier friend
yes
no
Plan attack
Request help
Execute a plan's action
Defend myself
Strategy = feel
yes
no
There is an agent requesting help
yes
no
Get the best movement to catch up with such an agent
Execute such a movement
Get the best movement to reach the empire
Execute such a movement
I am protecting an agent
no
yes
There is a miner agent not protected
yes
no
Get the best movement to catch up with the nearest miner agent not protected
Execute such a movement
Get the best movement to follow it
Execute such a movement

Figure 6.8 Behavior of a soldier

In Figure 6.8, as in Figure 6.6 and Figure 6.7, the evolutionary algorithms are used to determine the best movements.

As we cited above, there are mobile obstacles (the agents) and therefore, the best trajectory generated by an algorithm at time t may not be the best at time t+1. Therefore, the behavior of the agents calculates the best trajectory at each time and the agent adapts to the new changes.

6.6 Results

The algorithm described above was proved (with few changes) to generate trajectories in environments with extreme circumstances, i.e., environments with static obstacles that may not appear in the game. The mobile obstacles are not a problem if the algorithms operate in a good form in those extreme circumstances.

The possible agent's movements are depicted in Figure 6.9. The solutions were encoded into a sequence of integers from 0 to 7, and the individuals in the evolutionary algorithms are vectors of such integers.

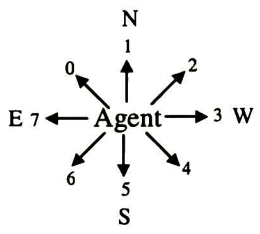


Figure 6.9 Possible agent's movements

The select movement does not depend of the angle of vision of the agent. The mutation operator was extended to increment or decrement the length of an individual and, like this, to allow to bring it near to its destination. If the mutation increments the length of an individual, it adds c movements to the trajectory (where c is a constant); if the mutation decrements the length of an individual, it removes c movements from the trajectory. Figure 6.10 shows some results obtained in the run of the three evolutionary algorithms. Each algorithm generates similar results. The differences are the following:

- The genetic algorithm operated in a good manner on all cases, because the algorithm is elitist and there are several individual per generation. For these experiments, we have used the following parameters:
    - population size = 50
    - initial length of individuals = 2
    - probability of crossover = 0.8
    - probability of mutation = 0.04
    - constant of increment/decrement in the length of individuals $c = 1$
    - maximum of generations = 200
- The (1+1)-ES is the fastest one and it operated in a good manner but it had its problems in cases b, c, d and e with $c = 1$ and initial length of individuals = 2, because, although it is elitist, there is only one individual per generation. This provokes that it difficult surround big obstacles (the algorithm falls easily in local minimums). The problem is solved with a higher constant c or employing some self-adaptive mechanism. This algorithm is elitist by definition. For this experiments, we have used the following parameters:
    - initial length of individuals = 2 (10 in the cases b, c, d, and e).
    - maximum of generations = 10000
- The ($\mu + \lambda$)-ES is the slowest but operated in a good manner, similar to genetic algorithms. The difference with the genetic algorithm is that it always maintains the best μ individuals. This algorithm is elitist by definition too. For this experiments, we have used the following parameters:
    - initial length of individuals = 2
    - maximum of generations = 500
    - $\mu = 20$
    - $\lambda = 30$

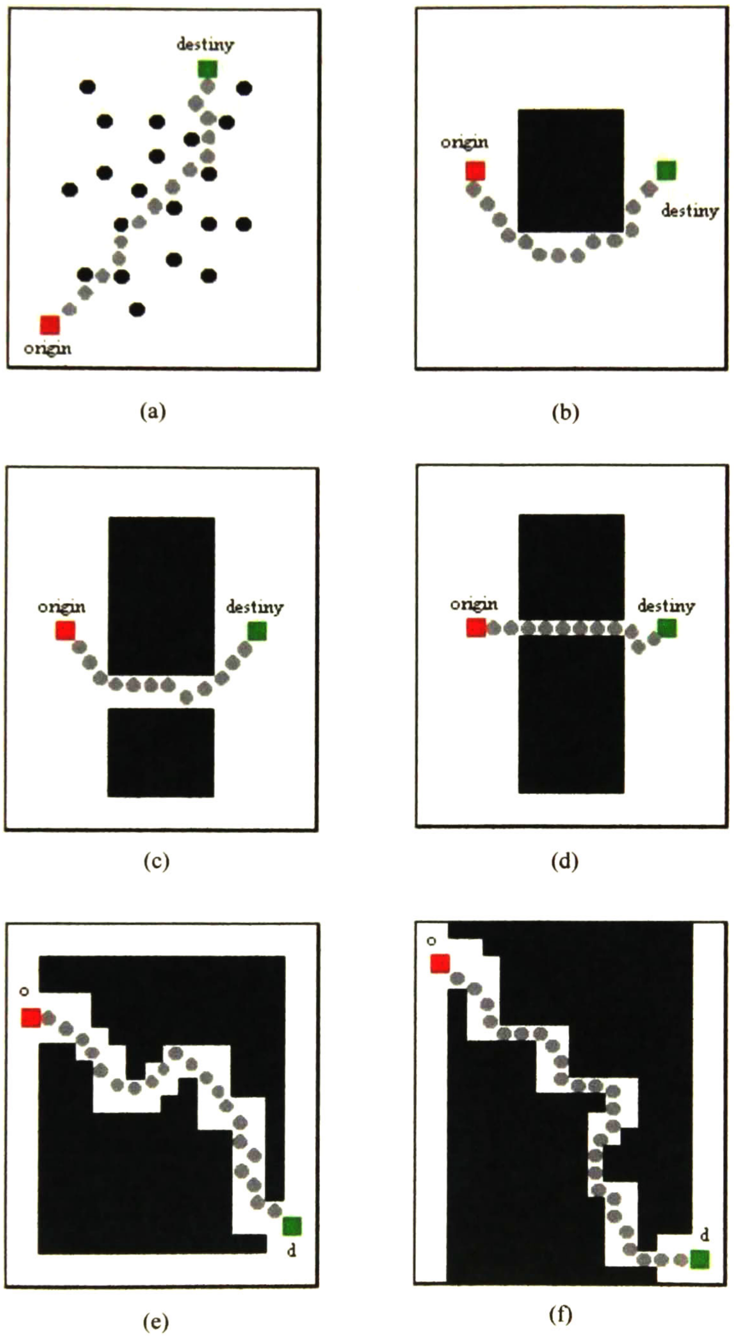


Figure 6.10 A few obtained results

6.7 Conclusions

In this chapter we have employed evolutionary algorithms to provide agents with adaptive behavior. These agents are virtual entities participating in a virtual combat environment that runs in the GEDA-3D platform. The algorithms that we have used belong to two of the main paradigms in evolutionary computation: Genetic algorithms and evolution strategies. We also implemented the algorithm (μ , λ)-ES that belongs to evolution strategies, but for the cases in which we considered the algorithm the obtained results were not satisfactory because the algorithm is not elitist.

Chapter 7

Conclusions

7.1 Objective

In this chapter we summarizes the results obtained in this work and present some ideas we got that will help to address the future work to get our final objective described at the beginning of this thesis.

7.2 Results

- We have enhanced a 3D-Space platform useful to integrate and manage distributed applications – GeDA-3D.
- We have created a basic mobility platform that is part of GeDA-3D.
- We have extend the work of Rüdiger Valk in [VALK1][VALK2], and with this little extension we have achieve a formal specification of the mobility platform incorporate to GeDA-3D and we have achieved a formal verification of liveness in such system.
- We have created the means to integrate a distributed application in GeDA-3D.
- We implement a case study and we have integrated it in GeDA-3D.

- We add adaptive behaviors to the agents participating in the case study, based them on three evolutionary algorithms belonging to two of the main paradigms in evolutionary computation: genetic algorithms and evolution strategies.
- We have proved the search subject is feasible to continue in a doctorate work.

7.3 Future work

Next we describe future work we propose to follow to get our main objective described at the beginning of this work, that is, create animation of virtual entities using adaptive algorithms.

Let describe a division of this work:

1. Describe and implement a platform useful to implement distributed applications where a 3D interface will be useful. This stage begins with the work of Maria Eugenia [PUGA] and Silvia Toscano [TOSCANO00] ex members of our team. Their main contributions were: the study of technologies necessitated to implement our platform, describe and implement a prototype of our platform GeDA-3D. Most of the time we have we devoted to implement a robust platform from this work and implement also different services not provided but necessitated to implement the sort of application we are interested in. There are still a work must be done along all the development using GeDA-3D and it is, the maintenance and implementation of new services for instance physical devices useful to implement immersive virtual reality, surely I will be involved in this part of work. Once, the architecture implemented next works can be started in parallel.
2. The study and implementation of a distributed editor of 3D-scenarios where autonomous entities can evolve. The leader of this part is Ivan Piza[1] member of our team.
3. Find the appropriate algorithms for helping autonomous virtual entities to evolve in its environment. The solutions for this part of the project constitute the objective of my present work. The next step will be the addressed toward: first, to compare the performance of these algorithms; start an study for establishing how useful are these algorithms to improve the performance of virtual entities trying to get an objective, for instance to plan a displacement, to solve non considered situations,

etc. In the second part we must be able to propose our implementation of agents in a real application, such as a virtual eco-system.

4. Animation of virtual entities is another part of this project; this part is quite related with the part described in point number 3. The main objective is to study and propose new adaptive algorithms to help a virtual entity to evolve physically in order to get an animation. Let explain with an avatar representing a human been, our objective is to use adaptive algorithms to help the avatar in its process to change form a position a to *a* position *b*. The adaptive algorithm will help the avatar for solving specific problems for instance how to go through a reduced space, change the position of the leg of the avatar if there are some problems like objects along the trajectory described by the leg to change its position. My intention is to work with the person or persons working on this problem.

5. Another problem we are interested to solve is how to address the evolution of the virtual entities in order to get an animation. The idea of our team is create a like script language, similar to that used by a real actor. This document will be described in a very high level language and will allow any user to create an animation.

[1] The e-mail of Ivan is hpiza@gdl.cinvestav.mx

BIBLIOGRAPHY

[ACHARYA97]A. Acharya, M. Ranganathan, J. Saltz. "Sumatra: a language for resource aware mobile programs" Mobile object systems, Lecture Notes in Computer Science, No 1222, Springer Verlag(D), pp. 111-130, February 1997.

[BACK] Thomas Bäck, Frank Hoffmeister, Hans-Paul Schwefel. "*A Survey of Evolution Strategies*". University of Dortmund. Department of Computer Science XI. Germany.

[BACK97] T Bäck, D. Fogel and Z. Michalewicz. "*Handbook of Evolutionary Computation*". Oxford University Press, New York, February 1997

[BIRRELL84] A. D. Birrell e B. J. Nelson, "*Implementing Remote Procedure Calls*", ACM Trans. On Computer Systems, vol. 2, pp. 39-59, February 1984.

[BOEHM88] B. Boehm. "*A spiral model for software development and enhancement*". Computer, vol 21. Mayo 1998, págs 61-72.

[BRENNER98] W. Brenner, R. Zarnekow, and H. Wittig. "*Intelligent Software Agents*". Springer, 1998.

[CABRI99] G. Cabri, L. Leonardi, F. Zambonelli. "Mobile Agents Technology: Current Trends and Perspectives". Dipartimento di Dcienze dell' Ingegneria – Università di Modena, 1999.

[CARLSON93] C. Carlsson and O. Hagsand, "DIVE A Platform for Multi-User Virtual Environments, Computers and Graphics 17(6), 1993

[CORBA] "The ORBACUS Home Page", http://www.ooc.com/ob/, Object-Oriented Concepts, Inc.

[COULOURIS96] George Coulouris. Jean Dollimore. Tim Kindberg. *Distribuited Systems Concepts and Design*. Addison Wesley 1996.

[DALE97] Jonathan Dale & David C. DeRoure. "A Mobile Agent Architecture for Distributed Information Management". University of Southampton. March 1997.

[DESEL95] Jörg Desel, Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press 1995.

[EIBEN94] A. E. Eiben, P.-E. Raue, and Zs. Ruttkay. "*Genetic Algorithms with Multi-parent Recombination*". Proceedings of the third International Conference on Parallel Problem Solving from Nature(PPSN). New York, 1994.

[FIPA] The Foundation for Intelligent Physical Agents. http://www.fipa.org/.

[FOGEL55] D. B. Fogel. "*Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*". IEEE Press, Piscataway, NJ, 1995.

[FOGEL66] L. J. Fogel, A. J. Owens and M. J. Walsh. "*Artificial Intelligence through Simulated Evolution*". New York. 1966.

[FUGGETA98] A. Fuggeta, G. Picco, G. Vigna. "*Understanding Code Mobility*". IEEE Transactions on Software Engineering, vol 24, No. 5, pp. 352-361, may 1998.

[HARRISON95] Harrison, C.G., Chess, DM & Kershenbaum. "Mobile Agents: Are they a good idea?" IBM Research Report, IBM Research Division, 1995.

[HINTERDING] Robert Hinterding, Zbigniew Michalewics, and Agoston E. Eiben. "*Adaptation in Evolutionary Computation: A Survey*".

[HISSA]"Conceptual Framework for System Fault Tolerance". http://hissa.ncsl.nist.go/chissa/SEI_framework/

[HOLLAND75] J. H. Holland. "*Adaptation in Natural and Artificial Systems*". University of Michigan Press, Ann Arbor, 1975.

[HYACINTH96] Hyacinth S. Mwana. "Software agents: An overview". Intelligent Systems Research. AA&T, BT Laboratories, 1996.

[JADE] "*Java Agent DEvelopment Framework*". http://jade.cselt.it/.

[JERRY] "*Trabajo cooperativo asistido por computadora*". http://isdale.com/jerry/VR/WhatIsVR/frames/WhatIsVR4.1.html & http://isdale.com/jerry/VR/techReview.html.

[KELLEY95] Dean Kelley. "*Teoría de autómatas y lenguajes formales*". 1995

[KNABE] Federick Knabe P. "An Overview of Mobile Agents Programming" Universidad Católica de Chile, Casilla 306, Santiago 22, Chile.

[KOZA94] J. R. Koza. "*Genetic Programming: On the programming of computers by means of Natural Evolution*". MIT Press, Massachusetts, 1994.

[LI] W. Li, D. G. Messerschmitt. "Itinerative Computing Using Java". http://ptolemy.eecs.berkeley.edu/dgm/javatools/java-to-go.

[LOGRIPPO] L. Logrippo, M. Faci, M. Haj-Hussein. "*An Introdiction to LOTOS: Learning by Examples*". University of Ottawa. Protocols research group. Department of computer science.

[LOPEZ97] Ernesto López Mellado. *Introducción a las Redes de Petri.* Universidad Autónoma de Nuevo León. Octubre 1997.

[MICHALEWICS96] A. Michalewics. "*Genetic Algorithms + Data Structures = Evolution Programs*". Springer Verlag, New York, 1996.

[MULLENDER95] Sape Mullender. *Distributed Systems.* Addison Wesley 1995.

[OBITKO98] Marek Obitko "Genetic Algorithms". Czech Technical University. September 1998. http://cs.felk.cvut.cz/~xobitko/ga/.

[ODISSEY] "*General Magic Odissey Page*". http://genmagic.com/agents/odissey.html

[OMG95a] Common Object Request Broker Archictecture, OMG, July, 1995.

[OMG95b] Common Object Services Specification, OMG 95-3-31, 1995

[OMG00] *Mobile Agent Facility Specification.* January 2000. OMG Specifications.

[PIZA02] Huga I. Piza, Felix F. Ramos. "A *Virtual Editor for GeDA-3D*". Multi-Agent Systems Development Group. Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional Guadalajara, Jal., México. 2002

[PNUELI95] Amir Pnueli, Zohar Manna. "*Temporal Verification of Reactive Systems*". 1995

[RAMOS02] Fabiel Zúñiga, Huga I. Piza, Felix F. Ramos. "*A 3D-Space Platform for Distributed Applications Management*". Multi-Agent Systems Development Group. Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional Guadalajara, Jal., México. 2002

[RECHENBERG73] Ingo Rechenberg. "*Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*". Frommann Holzboog Verlag, Stuttgart, 1973.

[RMI1] Java Remote Method Invocation (RMI). http://java.sun.com/products/jdk/rmi/index.html.

[RMI2] Java RMI Tutorial. http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html.

[RUDOLPH] Thomas Bäck, Gunter Rudolph, Hans-Paul Schwefel. "*Evolutionary Programming and Evolution Strategies: Similarities and Differences*". University of Dortmund. Department of Computer Science XI. Germany.

[RUSELL95] Stuart Rusell & Peter Norving. "Artificial Intelligence a Modern Aproach". Prentice Hall, 1995.

[SCHOENAUER] Marc Schoenauer, Zbigniew Michalewicz. "*Evolutionary Computation*". Control and cybernetics 26(3) pp 307-338.

[SCHWEFEL75] Hans-Paul Schwefel. "*Evolutionsstrategie und numerische Optimierung*" Dissertation, Technische Universität Berlin, May 1975.

[SCHWEFEL81] Hans-Paul Schwefel. "*Numerical Optimization of Computer Models*". Wiley, Chichester, 1981.

[SNOWDON94] "The AVIARY Distributed Virtual Environment", David Snowdon and Adrian West. The 2nd UK VR-SIG conference, 1st December 1994, Theale, UK.

[SUN96] "*The Java Virtual Machine*", Sun Microsystems White Paper, Sun Microsystems, 1996.

[SUN97] "*Remote Method Invocation*", Sun Microsystems White Paper, Sun Microsystems, 1997.

[SYCARA98] Katia P. Sycara. "*Multiagent systems*". American association for artificial intelligence. Summer 1998.

[TOSCANO00] Silvia Toscano G. "*Ambiente Genérico Virtual Distribuido*". Cinvestav Guadalajara. Septiembre 2000.

[UK] "*Distributed Virtual Reality Systems*". http://www.doc.ic.ac.uk/~np2/virtual_reality/distributed.html

[VALK1] Rüdiger Valk. "*Petri nets as token objects: An introduction to elementary object*". Universität Hamburg, Fachbereich Informatic.

[VALK2] Rüdiger Valk. "*Concurrency in Communicating Object Petri nets*" Universität Hamburg, Fachbereich Informatic.

[VINOSKI] Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, February, 1997.

[WHITE94] J. E. White. "Telescript Technology: the foundation for the electronic marketplace". White paper, General Magic, Inc, 1994.

[ZÚÑIGA02] Fabiel Zúñiga, Huga I. Piza, Felix F. Ramos. "*Specification and liveness proof of GeDA-3D Using Petri Nets*". Multi-Agent Systems Development Group. Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional Guadalajara, Jal., México. 2002

ANIVERSARIO
Cinvestav

Centro de Investigación y de Estudios Avanzados del IPN

Unidad Guadalajara

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la tesis: Adaptive Algorithms Useful to Generate Behaviours in Dynamic Virtual Environments, del(a) C. Fabiel ZÚÑIGA GALLEGOS el día 1 de Octubre de 2002 .

[illegible]

DR. LUIS ERNESTO LÓPEZ
MELLADO
INVESTIGADOR CINVESTAV 3A
CINVESTAV GDL
GUADALAJARA

[illegible]

DR. FÉLIX FRANCISCO
RAMOS CORCHADO
INVESTIGADOR CINVESTAV
2A
CINVESTAV GDL
GUADALAJARA

[illegible]

DR. VICTOR MANUEL LARIOS
ROSILLO
INVESTIGADOR TITULAR
UNIVERSIDAD DE
GUADALAJARA
ZAPOPAN