Generic Simulator Environment for Realistic Simulation

-

Autonomous Entity Proof and Emotion in Decision Making

Mickaël Camus
camus_m@epitech.net
and
Nabil El Kadhi
el-kad_n@epitech.net
L.E.R.I.A. - Laboratoire Epitech de Recherche en Informatique Appliquée,
{Epitech.}, 24 rue Pasteur 94270 Le Kremlin Bicêtre, France

Abstract

Simulation is usually used as an evaluation and testing system. Many sectors are concerned such as EUROPEAN SPACE AGENCY or the EUROPEAN DEFENCE. It is important to make sure that the project is error-free in order to continue it. The difficulty is to develop a realistic environment for the simulation and the execution of a scenario.
This paper presents PALOMA, a Generic Simulator Environment. This project is based essantially on the *Chaos Theory* and *Complex Systems* to create and direct an environment for a simulation. An important point is the generic aspect. PALOMA will be able to create an environment for different sectors (Aero-space, Biology, Mathematic, ...).
PALOMA includes six components : the Simulation Engine, the Direction Module, the Environment Generator, the Natural Behavior Restriction, the Communication API and the User API. Three languages are used to develop this simulator. SCHEME for the Direction language, C/C++ for the development of modules and OZ/MOZART for the heart of PALOMA.

Keywords: Simulation, Autonomy, Chaos Theory, Complex System, Generic, Environment, adaptative system, emotion, decison making.

1 Introduction

1.1 Context

Simulation is usually used as a first evaluation and testing system before developing commercial solutions. Many sectors are concerned, such as the European Space Agency with the development of **MYRIAD** [Tho01], a micro-satellite constellation, or the European Defence for the **MISURE** project [Ing02], **MI**ssion managment **S**ystem for **U**ninhabited ai**R** v**E**hicles and new techniques. It is always important to be sure that the simulation is safe and free from any errors before deciding whether or not to continue a project or an investment. Most current industry use simulators (such as Athena [CG01]) necessitates a specific description of a typical environment in order to run a simulation scenario. This description is important for the simulation since it must be as complete and accurate as possible in order to guarantee accurate simulation scenarios. The most difficult question here is how to give a simple description of an efficient environment and to conduct a realistic simulation without plunging into a lot of complicated syntax and constraints. Most of all the existing simulation tools [CG01] enable generated simulation scenario *automatically* and they rely totally on user input descriptions. The dilemma is how to conduct a complete and efficient simulation that is not limited or corrupted by user descriptions, and how to ensure that the simulation environment is realistic.
One solution is to consider a *dynamic environment* throught-out the simulation. In fact, the user will simply specify initial environment variables and then the simulator will modify the environment as in real life.

1.2 Project

PALOMA is a simulation tool for creating a *dynamic environment*. It is a *distributed generic simulator* composed of three essential elements:

- An Initial Environment Description;
- A Dynamic Simulation;
- A Simulation Scenario Generator.

All these components offer a set of functionalities ensuring a **realistic**, **coherent** and **natural** simulation. PALOMA will communicate with different **API**

(**A**pplication **P**rogramming **I**nterface) in distributed mode to dispatch computation on the network. In addition to the distribution computation model allowing PALOMA to schedule and dispatch computation throughout the network. PALOMA includes a "Direction Language" mainly used to indicate a starting point for simulation and a "Simulation Line". A "Simulation Line" is a kind of guideline or a set of rules to be respected during the simulation process.
Simulation process is very complicated, in fact, it can be divided in three major actions:

1. Analysing the context of the environment.
2. Defining the needed elementary actions.
3. Establishing scheduling and synchronisation.

This constantly-active simulation process is commonly known as *Thinking Model* and will be presented in section 2.

1.3 PALOMA Aims

This project has multiple goals, it is a simulation tool for the industry, but it is also an answer to a scientific problems: the simulation of an environment in evolution and the multi-critaria real time decision making. The features of PALOMA are:

- Simulation of a realistic evolving environment.
- Processing of autonomous entities.
- A proof of this autonomy.
- Control system for an entity group[Cam02].

Simulation allows us to evaluate component capacities and autonomy. An interesting feature will be to use the same simulation tool as a control and evaluation system in real situation. For example, in the case of Uninhabited Air Vehicle (UAV) mission. PALOMA can be used for both training and control.

2 Thinking Model

The model is based on human behavior. In fact, the thinking model mimics the human reasoning process. When you try to simulate an action you will:

- Analyse the context.
- Define/establish a scenario.
- Optimize event timing.
- Classify the entity's interaction with the environment.
- And finally, specify movement of the entity.

Concretly, a movement starts at a specific time, is realized within a specific time frame and in a given context and environment. This movement can be modified and use different parameters of this environment. By realising a movement, the environment parameter values are also modified. Let's take an example: three uninhabited air vehicles have a mission in a hostile environment. Each vehicle keeps a schedule to reach several goals (radio communication, photo, video, recognition ...). If radar or missiles detect their presence, the environment is modified and vehicles will not match their initial mission.

In this situation, planes have to react rapidly by either dynamically updating and modifying their mission plan or, to avoid any unpleasant surprise, cancel their mission and return to their starting point. Realistic environment evaluation and virtual entity behavior validation are hard to obtain because of the huge number of constraints to consider for in a realistic environment.
The previous example illustrates the importance of autonomy. In fact, in order to offer such abilities, simulation needs to be as realistic as possible and to include all *correct* and *possible* environment parameter modifications.

3 Global Software Architecture

PALOMA global architecture is defined in modular and easy extension modules. In fact, in order to follow a *Thinking Model*, PALOMA includes six parts such as described in Figure 1:

1. Simulation Engine (SE).
2. Direction Module (DM).
3. Environment Generator (EG).
4. Natural Behavior Restriction (NBR).
5. Communication API (C-API).
6. User API (U-API).

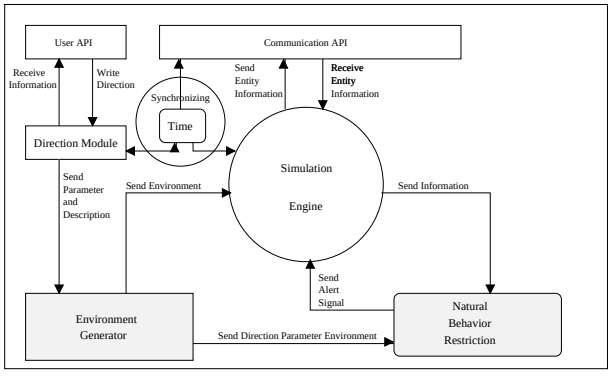

Figure 1: Global Software Architecture

Here is a rapid description of the PALOMA operator model. Human-machine interface is used to define simulation parameters and direct simulation environments. The DM: **Direction Module** will use parameters to limit the environment during simulation. The EG: **Environment Generator** constructs the environment based on parameters and the description of the **DM**. After this action, the **EG** sends the generated environment to the SE: **Simulation Engine** and to the NBR: **Natural Behavior Restriction**.
A major problem with simulating an environment evolution is how to guarantee that it follows a *realistic* modification scheme. The idea is to make sure that the simulation scenario is still coherent. It must represent a *realistic* situation. To do so, PALOMA includes the **NBR** module. This module is in fact a set of specific restriction rules applied to particular theorical models such as *Adaptative System* and *Complex System*. PALOMA evolution is synchronysed with the module time of the simulation and communicates with their **Communication API** and **Time API**. The user will be able to modify simulation steps, in real time, in order to establish certain specific events of the scenario.
To follow the **Thinking Model**, the context and the scenario are represented by the **DM**, the **EG** and the **NBR**. The interaction and the movement are represented by the **SE**, the **Communication API** and the **Time**.

4 Languages

To develop a simulator able to manage the highest and lowest level languages, three different programming languages are used:

- Oz language.
- C language.
- Scheme language.

PALOMA uses an *Expressive* environment description language: we voluntarily decided to use an existing and well used language to describe initial environments and constraints. The **Scheme** language [Abe96] has been chosen since it allows complex prototyping. A **Scheme** interpreter is also integrated in our simulator in order to create (or generate) *automatically* a potentially user controled environment description through a flexible interface.
The simulator is mainly developped with a high level programming system: **OZ/MOZART** [Roy02]. **OZ/MOZART** offers a good level of abstraction for communication, creation and evolution (dynamic aspect) of the environment. **OZ/MOZART** also includes several paradigms to solve distinct problems such as chaos control and constraints programming.
To integrate the Scheme interpreter in the simulator, we use a low level language: the C language. This creates a low level module in Oz. Severals macros have been written by the OZ/MOZART developers to implement an object with C language for OZ/MOZART.

4.1 Scheme Interpreter

The interpreter integrated in the project is the ELK for **Extension Language Kit** [Bre87]. ELK is developed by the Bremen University in Germany. It unables us to integrate Scheme in a program developed in C or C++ language.
In the future, we may be able to use GUILE [GNU00], an interpreter, compiler for Scheme developed by the MIT, It is a GNU Project. It also allows us the integration of Scheme in C/C++.

4.2 Oz/Mozart

The Mozart Programming System is a development platform for intelligent and distributed applications. In fact, Mozart is the name of the compilator and Oz is the name of the language. This system is developed by a consortium including “l’Université Catholique de Louvain”, “Swedish Institute of Computer Science” and “Universität des Saarlandes”.
Oz is a multi-paradigm language. It is object oriented, functional oriented and scripting oriented. Oz is cross-compiled and it is compatible with different platforms such as Linux, FreeBSD, MacOSX and Windows.

5 Initial Environment Description and Direction

This part integrates the User API and the Directive Module of the *Global Software Architecture*. It allows to describe and direct the evolution of the environment. This environment has to correspondto the initial environment of the other simulator.
Differents parameters must be initialized to ensure the environmental evolution. To do so, a set of specific macros are implemented as described in section 5.1.

5.1 Descriptive and Directive Method

Different methods have been implemanted to generate the environment :

- Initalization of the ELK parameters.
- Recovery of public variables.
- Some functions for population present in the environment, for example, radar or missiles for uninhabited air vehicles.
- Interaction and report function for species present in the environment.

- Input parameters of the *Logistic Map* [Fla99] to predict the evolution of the population.
- Stability and instability functions for the population.
- Input parameters for the *Shadowing Lemma* [Fla99] to rectify computation of the machine to mirror reality.
- Input parameters for the "Henon Map" [Fla99] to regulate "attractors". For example: for an aircraft's simulation, missiles attracted by aircraft.

6 Dynamic Simulation

This part is very important for a simulation because it allows users to discover new sub-problems by modifying environment parameters. Most simulators do not offer this facility, because they are not developped with a high level language. This part integrates different elements of the *Global Software Architecture*: the User API, the Directive Module, and the Simulation Engine.
Dynamic environment modification is implemented by using multi-thread programmation style, as see in Figure 2. In fact, **U-API** and **SE** will be dispatched in two different threads. Please note that a specific communication module is also implemented in order to allow their synchronisation.

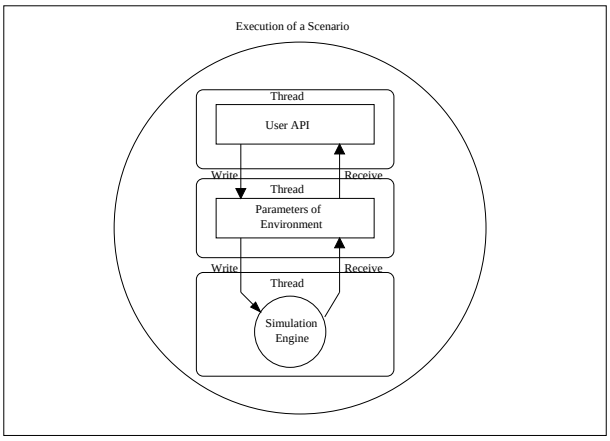

Figure 2: Thread's Architecture

7 Simulation Scenario Generator

This part integrates the **SE**, the **EG** and the **NBR** of the "Global Software Architecture". Inriched with filtering criteria, the correctness and the realistic aspect of the simulation will be guaranteed. This part is based on CHAOS and COMPLEXE SYSTEM Theory.

7.1 Characteristics

For a realistic simulation, PALOMA relies on CHAOS THEORY. Gary William Flake [Fla99] proved that we can control and manage Chaos. A Chaotic system is:

- **Deterministic**,
- **Sensitive**,
- **Ergodic**,
- **Embedded**.

A set of functions is used to gather information about the future of the population and movement evolution[Fla99]:

- "Logisctic Map".
- "Stability and Instablitity".
- "Shadowing Lemma".
- "Henon Attractor".

Collected information allows simulation control. In fact, all the parameter values will be analyzed and may be modified (automatically by the simulator) to avoid a simulation crash. A simulation crashes if it is evaluate in a unrealistic way.

7.2 Programming Technics

One major goal of our work is having a realistic simulation environment with a special focus on reuse, generic and autonomy components [RH04]. To do so, we use four paradigms :

1. Object Programming.
2. Funtional Programming.
3. Logic Programming.
4. Constraint Programming.

Object Programming is used essentially for the software architecture. The development of the different parts of the project need a lot of modularity. This aspect is crucial to facilitate further extensions.
Functional Programming is used to have a user interface language and facilitate the processing of the data in the simulation.
Logic Programming and Constraint Programming are used in the **NBR** Module to limit the evolution of the environments in a realistic scenario.

8 Control System: Emotion for Decision Making

One of the most important fonctionalities of PALOMA is to offer decision making exchange and export between entities. This export and exchange is required, because of the lack of power and processing time within embeded systems in general. PALOMA future developpment aims to reach autonomous entities as illustrated by figure 3. Autonomy is based on emotions [Car04] and relies on an agent society with a set of specific features[Min88].

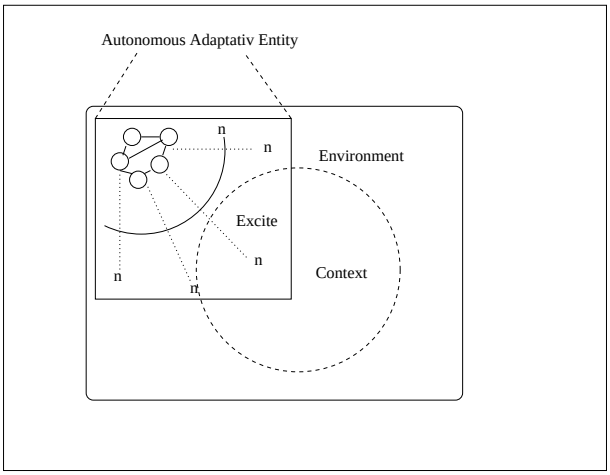

Figure 3: Adaptative autonomous entity

Please notice this decision making process is far from being a simple one. In fact, it is a multi-critaria real time decision making. This feature is important for the Army in the case of an urban war for example. The town is destroyed by missiles and tank, the environment is in constant evolution, and very important, it is hostile for a human. The time frame of a decision is critical to rescue human life. For a formation of *UAV*, it is difficult to evolve in this environment and follow objectives of the mission. A ground station PALOMA could synchronize the multipe data generated by the different actions to calculate an emotion and send it at the formation. With this method, the systeme doesn't need human factor, the decision is make in real time, so, the hostile frame time is diminued such show in figure 4.

9 Proof system

One of the most interesting aspect in such platform and simultation tools is trusting the platform capabilities and transitions. PALOMA include proof process in component description and definition. We are working on defining a concrete semantic for each decisional step and agent behavior adaptation. The upper estimation of the concrete

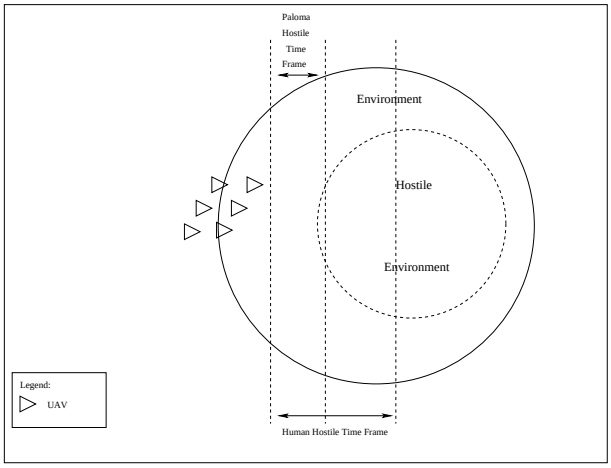

Figure 4: Hostile Time Frame

semantic -commonly knowen as abstract semantic-[CC77] will be used to infer agent and simulation properties by constraint propagation as done in [EK01] for cryptographic protocol verification. For the moment, we just include such process in the global component definition, we are working on detailing semantics for both the classical agent platform operations such as clonning, dispatching agents, and the PALOMA specific fonctionnalities such as environement ananlysis or dynamic parameters adaptation. The initial step is almost acheived by describing *'what to verify'*. Autonomy, correcteness and behavior stability or convergence will be described in a set of finite and computable properties. Each action, of any component of PALOMA, will be described by a transition rule and it's impact on the propagated constraints. Additional details are available from the authors and will be published soon.

10 Monitoring

An immediate extension is to integrate a **M**onitoring **S**ystem. The **MS** allows users to follow the simulation and to see the extrapolations of the scenario. If this extrapolation is not realistic, the user could re-initialize parameters dynamically.

11 Conclusion

PALOMA is a generic environment simulator with three essential elements: the initial environment description, the dynamic simulation and the simulation scenario generator. PALOMA is generic. In fact, it is possible for PALOMA to communicate with other simulations (Aerospace, Biology, Mathematical). Such integration is possible if other simulators use PALOMA as an initial environment (**C-API**).
PALOMA is based on *Chaos Theory* and *Complex Systems* to follow natural aspects and mirror reality.

An important goal of this project is to prove that a software entity is autonomous and do a multicritary real time decision making. With these tools, we could prove the autonomy of an entity in different sciences.
The module that permits the initial environment description is developed, but some modifications have to be developed on the type of data. There is a prototype for the **SE** and **EG** but not for the **NBR** at this time. In the future, the goal is the development of the **NBR**, and later, to work on the autonomy proof.

References

[Abe96] Abelson. *Structure And Interpretation of Computer Programming 2nd Edition*. 1996.

[Bre87] Universitat Bremen. the extension language kit, 1987.

[Cam02] Mickaël Camus. **MISURE** - **mi**ssion managment **s**ystem for **u**ninhabited ai**r** v**e**hicles and new techniques : PLANIFICATION ANYTIME. Technical report, {Epitech.} - Ecole pour l'Informatique et les nouvelles TECHnologies, sept 2002. Mémoire de fin d'étude.

[Car04] Alain Cardon. *Modéliser et concevoir une machine pensante*. 2004.

[CC77] Patrick Cousot and Radhia Cousot. Abstract Interpretation : a Unified Lattice Model for Static Analysis of Programs by Construction of Approximation of Fixpoints. In *4th ACM Symposium on Principles of Programming Languages*, 1 1977.

[CG01] Jean-Francois Tilman Christophe Guettier, Bruno Patin. Validation of autonomous concepts using the athena environment, 2001.

[EK01] N. EL Kadhi. Automatic verification of confidentiality properties of cryptographic programs. *Networking Information System*, 6, 2001.

[Fla99] Gary William Flake. *The Computationnal Beauty of Nature*. 1999.

[GNU00] GNU. Guile, 2000.

[Ing02] Jean-Claire Poncet Axlog Ingénierie. Présentation du projet misure, 2002.

[Min88] Marvin Minsky. *Society of Mind*. 1988.

[RH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. 2004.

[Roy02] Peter Van Roy. Oz/mozart environment, 2002.

[Tho01] Michel H. Thoby. Myriade : Cnes microsatellite progam. 2001.