MultiAgent Distributed Simulation with GALATEA

Jacinto Dávila,
Erasmo Gómez,
Klaudia Laffaille
CeSiMo. Facultad de Ingeniería.
Universidad de Los Andes.
Mérida, 5101. Venezuela
{jacinto, erasmog, klaudia}@ula.ve

Kay Tucci,
Mayerlin Uzcátegui
SUMA. Facultad de Ciencias
CeSiMo. Facultad de Ingeniería.
Universidad de Los Andes.
Mérida, 5101. Venezuela
{kay, maye}@ula.ve

Abstract

*In this article we have presented an example that allows showing the design and implementation details of the module **gFipaOS** that permit incorporate to the simulator of multiagents systems on GALATEA some platforms of distributed agents. In this module GALATEA's agents management is based on the multiagent communication standard called FIPA (*Foundation for Intelligent Physical Agents*).*

The model represents a Patio-like world which changes because agents change it and also due to natural events (food appears). The model is implemented on a platform that distributed agents across several computing hosts.

1. Introduction

GALATEA (*GLIDER with Autonomous, Logic-based Agents, TEmporal reasoning and Abduction*) [7, 8, 4] is a family of simulation languages which allow users to specify and design multi-agent systems in terms of high-level concepts such as goals, beliefs, preferences. Additionally, it will allow a user to model plans, roles, communications, coordinations, negotiations, and dialogues in order to generate efficient and effective solutions for complex simulations models. GALATEA is implemented as a DEVS platform to simulate multi-agent systems.

GALATEA is the product of two lines of research: simulation languages based on Zeigler's theory of simulation [19, 18] and logic-based agents [5]. There is in GALATEA a proposal to integrate, in the same simulation platform, conceptual and concrete tools for multi-agent, distributed, interactive, continuous and discrete event simulation. It is also GALATEA a direct descendent of GLIDER [11, 10], a DEVS-based simulation language which incorporated tools for continuous modeling as well. In GALATEA, GLIDER is combined with a family of logic programming languages specifically designed to model agents [5, 6, 3].

However, we do no aim at replacing traditional modeling techniques with a new modeling language, but instead we try to integrate well-known approaches (DEVS modeling and its extensions) together with a new approach that allow us to include, in a simulation model, those reasoning-acting entities: the agents. By appealing to engineering techniques from Artificial Intelligence and Logic Programming, we are capable of producing a light, reasoning engine to simulate each agent's observe-reason-act cycle. With as many agent engines as required for the particular application and with the traditionally conformed simulator for the environment in which those agents are placed (the main simulator), we create a federation (In HLA's sense [9]) which serves as the simulator for the whole multi-agent system.

GALATEA architecture (Fig. 1) is based on objects. Both the agents and the main simulator are designed according to an OO layout to support distribution (of objects), modularity, scalability and interactivity as demanded by the HLA specification. Our aim is a flexible platform from the software engineering point of view (which is, arguably, inaccessible to final users: the modelers) but based on a family of modeler-friendly languages with enough expressiveness to allow the modelers to describe a multi-agent system in a way that makes feasible its simulation. We think that this possibility is critically dependent on domain and application specific trade-offs. Thus, we allow the modelers to describe systems in which there are agents, but in which not everything is an agent and in which traditional discrete-event or continuous modeling techniques are good enough for most purposes (such as dealing with subsystems that required very aggregated models to make their simulation feasible at all).

In this article we present an example which illustrates one of the newest modules of GALATEA, **gFipaOS**[13]. By means of this module GALATEA's agents management

Proceedings of the 2005 Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'05)
0-7695-2462-1/05 $20.00 © 2005 IEEE

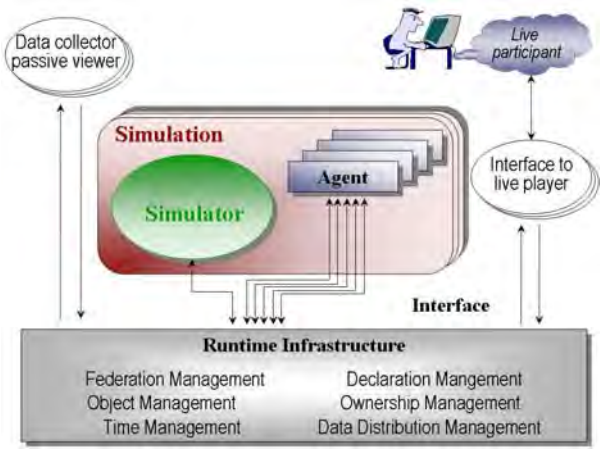

Figure 1. Architecture of the multiagent simulation platform GALATEA.

is based on the multiagent communication standard called FIPA (*Foundation for Intelligent Physical Agents*)[12]. The example, however, is a much wider application. We present another extension of GALATEA to model urban spaces and simulate the dynamics of agent mobility inside buildings or open spaces.

2. A distributed simulation of urban spaces

Today, it is possible to develop computational models, using modeling and simulation techniques, which allow for proving and comparing the feasibility and effectiveness of designs for the evacuation of mobile agents in case of catastrophes. There are two classic ways to define a model for this kind of systems. One way is using cellular automata [1, 16, 17], where space, states, and movement rules are discrete. The second one is modeling the system using continuous space, states, and movement rules [14] to obtain more realistic models. In both cases the system evolves in constant discrete time steps. In this example we used gSpaces[15], a hybrid and flexible GALATEA's module to develop and simulate models for this systems, that combine characteristics of both classical paradigms.

With **gFipaOS** we offer a communication protocol for the agents embedded in a GALATEA model, allowing for interoperability of these agents in a distributed platform, using a group of services and formats previously established by FIPA. The current implementation uses the agent management system of FIPA-OS (*FIPA-Open Source*), with high level communications systems and agent management services, which facilitates the associated tasks of creating and manage multiagents systems in GALATEA.

3. GALATEA's gFipaOS Module

This section presents the design and implementation details of gFipaOS. Figure 2 shows the agent-oriented, distributed architecture based on Fipa-OS conformed by multiple hosts: **`Main Host`** is where the base `Simulator`, which controls the simulation model (environment variables, simulation time, synchronization of the agents, etc.), is located. There is also `GInterface`, an interface controller that permits interactions between GALATEA `Agents` and the simulation model. Additionally, `Main Host` also hosts a group of communication-supporting fipa agents (`SeudoAgent, GalServer, ServerRemote`)

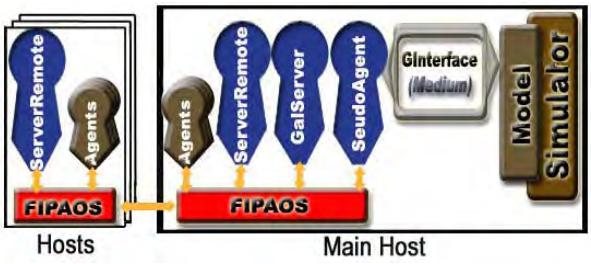

Figure 2. gFipaOS Design. Agents Distributed Architecture.

`SeudoAgent` coordinates requests between `GInterface` and `Agents`. `GalServer` controls the status of the hosts in the distribution architecture and maintains the registration of the `Agents` into the `Hosts`. `ServerRemote` initializes the `Agents` and extract their influences as demanded by `GInterface`.

The distribution architecture may be constituted by many **`Hosts`** in each of which there is a `ServerRemote` which controls the communications with `Main Host` and the `Agents` of the GALATEA Model.

Implementations details are shown with the class diagrams in (Fig. 3), where:

- `Ag` and `Agent` are basic classes for GALATEA's agents. GALATEA `GInterface` is in charge of communications between each agent and the simulator.
- `GFipaOS` incorporate FipaOS specifications into GALATEA agents.
- `InfoHost` and `ListHost` store hosts information.
- `SeudoAgent, GalServer` and `ServerRemote` are the communication-supporting agents.

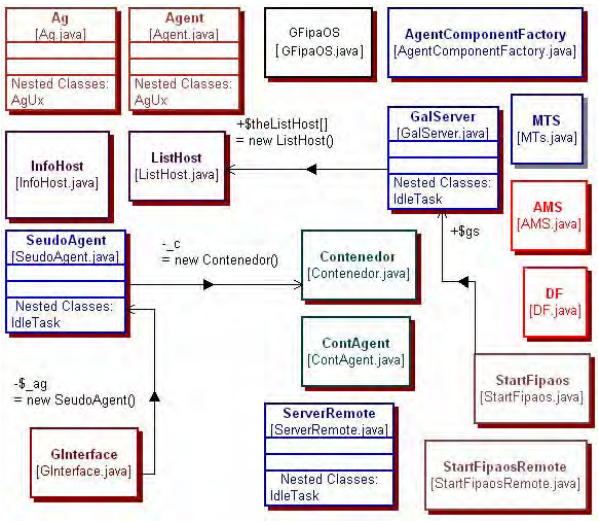

Figure 3. gFipaOS implementation. Class Diagram.

- `DF, AMS, MTS` and `AgentComponentFactory` are the FipaOS components
- `StartFipaos` and `StartFipaosRemote` start the agent supporting platforms.
- `Contenedor` and `ContAgent` control the agent messaging systems.

The main tasks performed by gFipaOS are `Agent_Initiation`, assigning the parameters to configure the agents, `Agent_Request`, to consult an agent, and `Add_Host`, to add a host to the platform. Figure 4 include the communication model to a simple agent request which includes eight steps:

1. The simulation Model invokes `GInterface` in order to start a request.
2. `GInterface` invokes `SeudoAgent` to consult the location of an agent.
3. `SeudoAgent` invokes `GalServer` to obtain the host IDs.
4. Depending on `GalServer`'s response, `SeudoAgent` sends the request to the specific host. As shown in Fig. 4, there exist three types of requests. One is chosen depending on the class-location of the method to be invoked. It can be an `Ag` associated method, or `Agent` associated method, or a method in model-specific subclass of those classes.

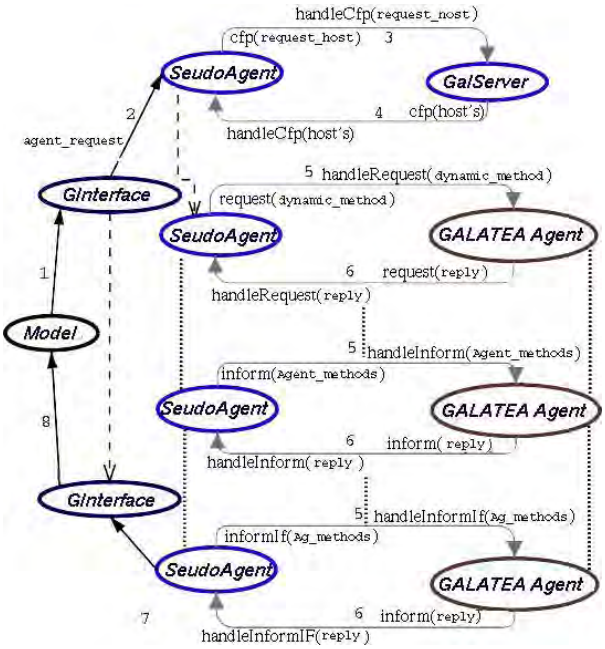

Figure 4. Request Communication Model.

5. `SeudoAgent` sends the corresponding method to the `GALATEA Agent`.
6. `GALATEA Agent` replies to `SeudoAgent`
7. `SeudoAgent`, in turn, replies to `GInterface`.
8. Finally, `GInterface` sends the reply to the simulation `Model`.

4. Example

Figure 5 shows the space for the example. For the sake of space, we will use that display both to explain the process of developing a model of mobile agents and of simulating with it. The model represents a Patio-like world which changes because agents change it and also due to natural events (food appears). The model is implemented on a platform that distributed agents across several computing hosts.

The environment of the multiagent system is a cell-like world where agents can move from one cell to a neighboring cell if there is no agent already in that cell. In this environment, food can appear in all but one of these cells. The special cell, in which no food can appear, is considered as a warehouse where the agents can bring and collect their food. An agent can observe if there is food in the cell it is currently visiting. Initially, food is placed in some randomly selected cell. During the execution, additional food can appear dynamically in randomly selected cells except

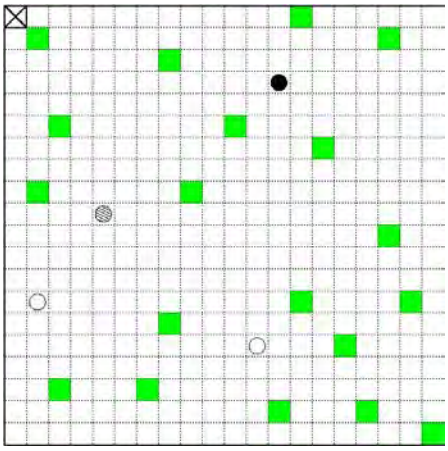

Figure 5. Example model. Different agents are represented by circles, cells with food are represented by remarked cells and warehouse is represented by the X cell

the warehouse cell. The agents may play different roles (such as explorer or collector), communicate and cooperate in order to find and collect food in an efficient and effective way[2].

To develop this multiagent system, some gSpaces and gFipaOS components are used in order to build models. These components are shown and explained in what follows.

gSpaces is a module of the GALATEA platform for simulating agent mobility in urban or architectonic designs. The spatial characteristics of the system are added to or deleted from the model at execution time, allowing the simulation of events that change the structure. This module allows for the development of complex models of architectonic or urban spaces with several types of mobile agents in them. For users who are not familiarized with GALATEA is relatively easy to create models using tools like dxf2g, thus avoiding its spatial aspects, as well as programming details of the model.

This space is divided in two `Spaces`: `Warehouse`, and `Patio`. In the model they are declared as follows:

```
public static Space Warehouse =
   new Space("Warehouse", 0, 1.0, 20.0,
             1.0, 0.1,"Quiet");
public static Space Patio =
   new Space("Patio", 4, 8.0, 6.0,
             1.0, 0.1,"MovementRule");
```

where, the `Space` methods' arguments are: name of the space, number of mobile agents at starting time, an internal point coordinates, spatial resolution, and time step. In the `Warehouse` declaration there is one more argument normally used to select the movement function in the space. In this case `"Quiet"` means that no agent can stand on this space.

Figure 6, shows snapshots of the status for four different simulation times in the same execution. Tree different types of mobile agents are represented with different circles in black (visitor agent) grey (explorer agent) or white (collector agent). Each color represents a particular type of agent, characterized by its own goals and rules of behavior, also regarded as beliefs. We are, of course, distributing the agents across several hosts profiting on FIPA's support. The actual encoding of this is as follows:

```
...
Patio.addAgMovil(2,white);
Patio.addAgMovil(1,grey);
Patio.addAgMovil(1,black);
...
Class MovementRule{
public double [] move(Move e, Message m,
                      Cell c){
  double [] nCoord = new double[4];
  in type = m.getIntValue("type");
  switch(type){
    case white:
      nCoord = collector(m);
      break;
    case grey:
      nCoord = explorer(m);
      break;
    case black:
      nCoord = visit(m);
      break;
  }
  return e.checking(m,c,nCoord);
}
...
}
  ...
```

Figure 6(a) shows the initial state of the simulation, with the four agents (two white, one black and one grey) randomly distributed, as it is also the case with the nineteen cells with food.

Figure 6(b) displays a state, no long after the initial state, in which the first agent (`ag[4]`) find food. The remaining food-cells are, therefore, the same.

In Fig. 6(c) more agents have found and taken food to the warehouse (which explains the lower number of food-cells). One of the agents is shown while carrying food to the warehouse, but not quite there.

Figure 6(d) shows an agent leaving food in the warehouse.

Finally, the standard GALATEA code to complete the model is:

```
Glider.setTsim(100);
Glider.act(creator,0);
Glider.act(Warehouse.getMove(),0);
Glider.act(Patio.getMove(),0);
```

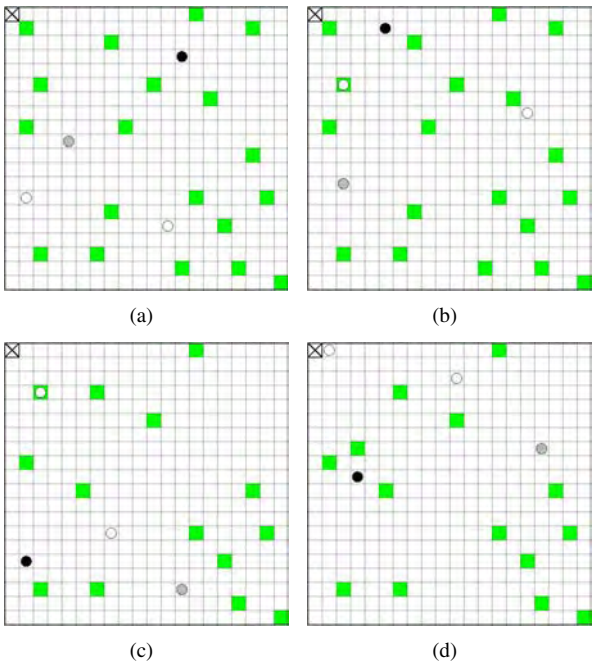

Figure 6. Snapshots of the simulation of the example model. a) represents the initial state of the simulation, b) shows an instant in which an agent finds food, c) shows a state in which an agent is leaving food in the warehouse.

Notice that a skeleton of the model can be generated, by means of the dxf2g tool, from a Drawing Interchange Format (DXF) ASCII file, with the drawing of the space. This way, the modelist would only have to establish the resolution of each space and set some parameters like the amount and type of mobile agents in each `Space` as we did before, but also the capacity and delay time associated with `Doors` and `Walls`; the movement rules of the different mobile agents types; and the simulation time.

5. Conclusions and Future works

We have used any implementation of the FIPA standard to build a distributed simulator of multiagent systems. In particular, we have used the FIPA-OS agent management system, with fipa's performative acts supporting the interactions between GALATEA agents and GInterface, the main system simulator. Additionally, FIPA-OS is responsible for the distribution of the agents into the different computers of a networked architecture.

gFipaOS's design prescribes a centralized structure for a set of communication-supporting agents (SeudoAgent, GalServer, ServerRemote) and an interface (GInterface) which is responsible for negotiating all the interactions between agents and the simulation model.

GALATEA allows for a seamless integration of gFipaOS[13] with the model generator gSpace[15], which allows for the creation of models of urban spaces with mobile agents.

One of the most important future works to be carried out on GALATEA's distribution platform, is to extend the implementation of the FIPA standards. This way, it could interoperate with others agents management systems that support FIPA protocols, incorporate the use of the different services that they can offer, and, as a whole, increase the interoperability of our agents with existing simulating, agent communities.

6. Acknowledgements

This work was supported in part by a grant from the CDCHT-ULA.

References

[1] O. Biham, A. A. Middleton, and D. Levine. Self-organization and dynamical transition in traffic-flow models. *Phys. Rev. A*, (46), 1992.

[2] CLIMA. The first clima contest, 2005.

[3] J. Dávila. Actilog: An agent activation language. In *PADL2003*, LNCS, New Orleans, USA, 2003.

[4] J. Dávila and M. Uzcátegui. Gloria: An agent's executable specification. *Collegium Logicum. Kurt Gödel Society*, VIII:35–44, 2004. Vien, Austria.

[5] J. A. Dávila. *Agents in Logic Programming*. PhD thesis, Imperial College of Science, Technology and Medicine, London, UK, June 1997.

[6] J. A. Dávila. Openlog: A logic programming language based on abduction. In *PPDP'99*, Lecture Notes in Computer Science. 1702, Paris, France, 1999. Springer.

[7] J. A. Dávila and K. A. Tucci. Towards a logic-based, multiagent simulation theory. *AMSE Special Issue 2000. Association for the advancement of Modelling & Simulation techniques in Enterprises*, pages 37–51, 2002. Lion, France.

[8] J. A. Dávila and M. Uzcátegui. Galatea: A multi-agent simulation platform. *AMSE Special Issue 2000. Association for the advancement of Modelling & Simulation techniques in Enterprises*, pages 52–67, 2002. Lion, France.

[9] DoD DMSO. High level architecture (HLA). Technical report, Departament of Defense. Defense Modeling and Simulation Office, 1995.

[10] C. Domingo. GLIDER, a network oriented simulation language for continuous and discrete event simulation. In *International Conference on Mathematical Models*, Madras, India, August, 11-14 1988.

[11] C. Domingo and M. Hernández. Ideas básicas del lenguaje GLIDER. Technical report, Instituto de Estadística Aplicada en Computación, Universidad de Los Andes. Mérida. Venezuela, October 1985.

[12] FIPA. The foundation for intelligent physical agents.
[13] E. Goméz. Diseño e implementación de la plataforma distribuible para el simulador multiagentes galatea. Master's thesis, Maestría en Modelado y Simulación de Sistemas, Universidad de Los Andes. Mérida. Venezuela, May 2005. Tutor: Tucci, Kay.
[14] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, (407):487–490, 2000.
[15] K. Laffaille. Gspaces, meta-modelo para simular desalojos de espacios urbanos y arquitectónicos basado en galatea. Master's thesis, Maestría en Modelado y Simulación de Sistemas, Universidad de Los Andes. Mérida. Venezuela, 2005. Tutor: Tucci, Kay.
[16] K. Nagel and J. H. Herrmann. Deterministic models for traffic jams. *Physica A*, (199), 1993.
[17] D. E. Wolf, M. Schreckenberg, and A. e. Bachem. *Traffic and Granular Flow*. World Scientific, 1996.
[18] B. Zeigler. *Object-oriented simulation with hierarchical, Modular models (Intelligent agents and endomorphic systems)*. Academic Press, Inc (Harcourt Brace Jovanovich, Publishers), Boston-Sydney, 1990.
[19] B. P. Zeigler. *Theory of modelling and simulation*. Interscience. Jhon Wiley& Sons, New York, 1976.