Distributed Active Object Model

Mihaela Ordean
Computer Science Department
Technical University of Cluj-Napoca
Baritiu 26-28 Cluj-Napoca
Romania
Mihaela.Ordean@cs.utcluj.ro

Dorian Gorgan
Computer Science Department
Technical University of Cluj-Napoca
Baritiu 26-28 Cluj-Napoca
Romania
Dorian.Gorgan@cs.utcluj.ro

Abstract **– This paper proposes and experiments the communication and synchronization in the distributed Active Object Model (AOM) as an environment for the implementation and demonstration of a distributed communication in the description of a graphical scene. AOM model is composed of many types of collaborative entities. Is described a graphical scene in AGML language. The article includes also the AGML extension for distributed computing.**

I. INTRODUCTION

The object oriented systems have been developed in '90, because the growing problems complexity could not be satisfied anymore by the structured programming paradigms.

The structure programming is quite inappropriate for domains like simulation systems, animation, adaptive systems, visual programming and graphical user interfaces.

The active object based model seems to satisfy better such requirements. The Active Objects Model AOM model has been developed as an experimental version at the University of Cluj-Napoca and at the Rutherford Appleton Laboratory, UK.

In the domain of graphical animation, simulation, CAD, virtual reality or multimedia are needed some graphical dynamic models to reflect as much as possible the object scenes of the real world, where those objects structure and behaviour are influenced by any action executed on an object.

The AOM model allows the specification of some dynamic evolutions of constituent entities, each component having the possibility to interact with each other component and modifying their structure and/or behaviour. Here, the behaviour is defined as a relationship between the object and the model.

This document provides an extension of the AOM in the space of the distributed systems.

The object model description is realized by the means of AGML language (AGent Modeling Language) which is a high level specification of the AOM concepts and assures the models portability for different system operation platforms (Windows, Unix, Linux, etc.).

II. FUNDAMENTAL NOTIONS

Active object model is made up both from active and passive entities [1]. The active entities are represented by object and variables (fig.1) and the passive entities are represented by behaviours, trajectories, rules, actions (fig. 2).

The objects are active entities with private behaviour. The behaviour consists of a set of rules, each rule meaning a condition and a set of actions [4]. The evolution implies the execution of the associated behaviour on a specific trajectory.

OBJECT

state
active object components
behaviour
presentation

PROCESSORS

Behaviour processor
Server processor
Presentation processor
Interactor processor

Fig.1. Object communication model

The behaviour execution of the active objects is represented by a set of successive actions realized by the current entity or by the delegated entities. The entity behaviour is given by the set of associated conditioned actions. The entities of the model communicate by the mean of messages.

The trajectory is defined by a set of parameters, a set of positions and a set of actions which are executed by the object during its evolution. The trajectory allows the possibility of direct manipulation of the entities and their elements. Also, the trajectory is characterized by a geometric model, direction, quantification (the number of steps to be passed between two explicit trajectory positions).

The model is represented by a set of active objects having an asynchronous behaviour. Application developer (generally, the user), builds the model using the entity direct manipulation approach. The developer creates, deletes and instanciates model entities and defines their behaviour. During the system running, the active object behaviours are executed. Depending on the evaluation result, the defined actions are executed (there are taken into consideration: the state of the model, the object attributes, positions, parameters, so on).

The model state can be changed by the intrinsic evolution of the model or by external

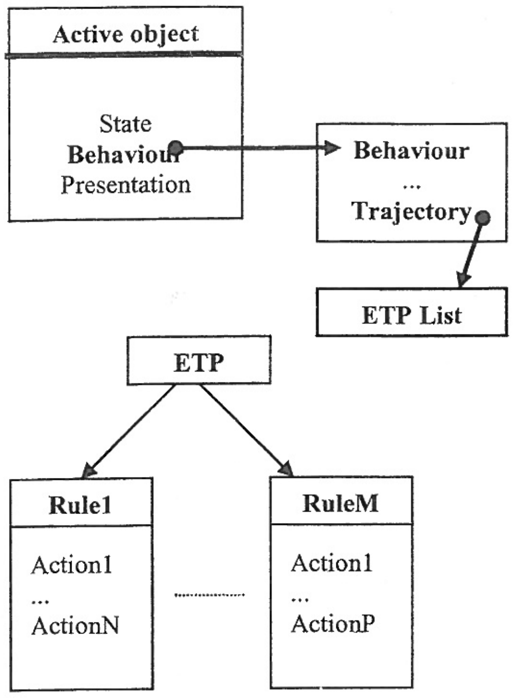

Fig.2. AOM Components

events. The external events act upon the model by means of interface between the user and the model.

Interface events are managed by interactors, a special type of active objects, which check the signals from/to external modules and send/receive messages to/from the model. The interactors transform the synchronous communication between the model and external modules, into a communication based on messages in the model.

The behaviour of an object is a sequence of actions executed by the object or by delegated objects. The actions operate upon the input elements and generate output elements. The processed elements can be object attributes, object behaviours, variable values, trajectory positions, position associated actions, so on. The access to the elements upon which an operation is executed is authorized only by means of the processor associated to the accessed entity. This restriction is imposed by the synchronization between the object behaviours, common resources multiple access, and the flexibility of an implementation and distributed execution of the model. The delegation principle of the server processor is above the object encapsulation principle, used in object oriented programming paradigm.

II. COMMUNICATION SOLUTION

At execution, the behaviour is implemented by a processor which manages the accesses to all local resources of the associated object, generates the intermediate positions between two consecutive ETPs (Explicit Trajectory Position) and responds the requests coming from other behaviour processors.

Distributed execution assumes that the agents which are running on different machines can be visualized on one computer.

The distributed control might be achieved in many ways:

1. *Centralized*: there is one application managing the running agents, application standing on one machine allowing the communication between the agents. The application maintains a table with all running agents. This table may be updated by adding new running agents at the moment of their start, by deleting the agents from the list at the moment of their stop.
   Advantage: easy agent management.
   Disadvantage: the centralized application has to know the types of all entities exchanged between the agents.
2. *Distributed*: on each machine storing an agent is running an application, a process which realizes the communication between the agents. The agents table has to be kept on each machine.
   Advantage: the communication between the agents is a direct one, without a centralized application.
   Disadvantage: the agents' evidence is hardly realized.
3. *Combined* (a combination of centralized and distributed). There is a centralized application maintaining the agents' evidence but the communication between the agents is not by the mean of this application but directly.

The third variant was chosen for implementation because it eliminates the disadvantages of the first two solutions.

III. IMPLEMENTATION SOLUTION

Windows was used as the operating system. This OS includes many mechanisms to support distributed computing.

Typically, distributed computed means a task split into two or more components. The firs component runs on a client machine and needs minimal resources and the other component runs on the server and uses high resource capacity for data processing and a specialized hardware.

Another type of distributed computing splits the work between many computers. In both cases a computer connection at the process level allows data exchange bidirectionally.

Windows includes the following inter-process communication mechanisms: Windows Sockets, Remote Procedure Call (RPC), NetBios, named pipe files and Distributed Component Object Model (DCOM). From these solutions we chose the socket communication model and as the implementation language Visual C++ (after some attempts in some other solutions it was proven that this one fits better the requirements). The used class hierarchies are those illustrated in fig.3.

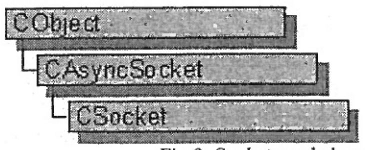

Fig.3. Socket used class hierarchy

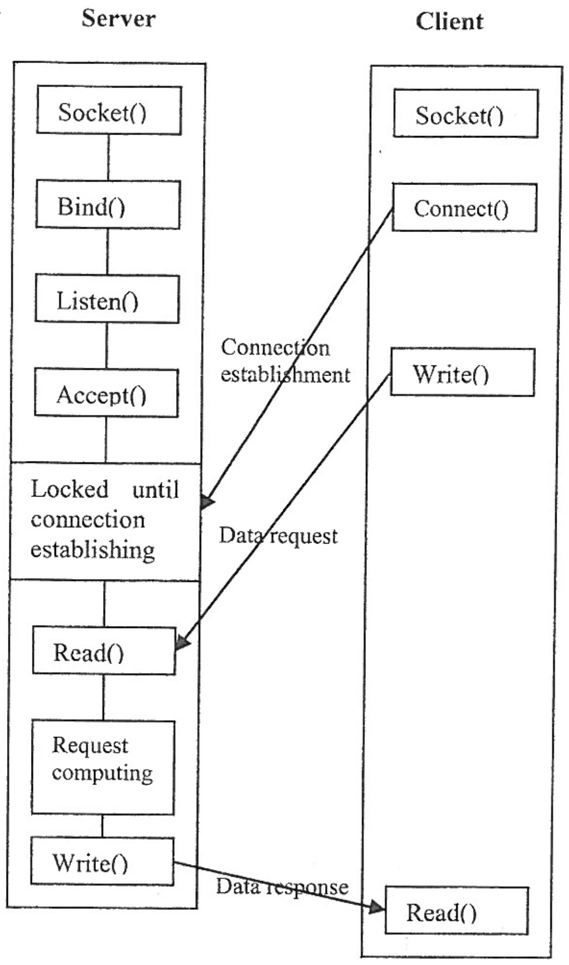

Fig.4. Connection oriented protocol phases

The application uses the connection oriented protocol (fig.4).

The messages exchanged between agents and server can be in of following types:

- REGISTER - client registration
- UNREGISTER – deleting an agent
- BROWSE_SERVER – listing running agents
- MSG_OK – server response for communication success
- MSG_FAIL – server error response

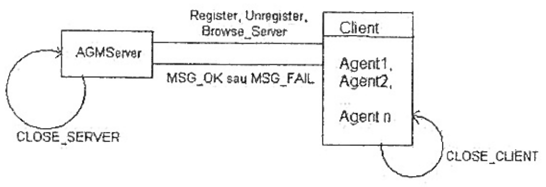
Register, Unregister, Browse_Server
AGMServer
MSG_OK sau MSG_FAIL
Client
Agent1, Agent2, ... Agent n
CLOSE_SERVER
CLOSE_CLIENT

Fig.5. Communication diagram

The AOM model is described in AGML language which was extended in order to support distributed execution facilities. In this sense were introduced some keywords like:

- "register": declares the server machine (AGMServer application).

  Syntax:

  register <machine_name_or_address>

Action mode: when an AGML file is read, the machine name or address is stored in a memory location. This location will be accessed each time an agent is registered with the given server, an agent is discharged or when is needed the machine address in the moment of communication with a running agent.

- "extern": declares the name of the agents located on different machines and referred in AGML code. Syntax:

  extern <agent_name >

Using this keyword has pros and cons.

Pros: In the absence of this keyword the agents are treated without considering if they are local or on different machines. In this case the agents must have unique names in AGML models. Also, each agent should register with a server generating network loading. If it is considered that the local agents are more then distributed agents, the network traffic becomes really high. Another cause of the network traffic is the name server accessing for local agents, because an application would have no idea if an agent is a local one or a remote one. Contrary, when the "extern" keyword is present, the network traffic becomes lower. By default it is considered that the agents are locals, the distributed agents being registered with "extern".

Cons: On each machine using external agents must keep a list with these agents.

Action mode: The reading of the extern agents list is realized by interpreting AGML code. The server is accessed only when a distributed agent is registered or when a communication between two distributed agents took place.

The AOM distributed architecture is shown in the fig.6.

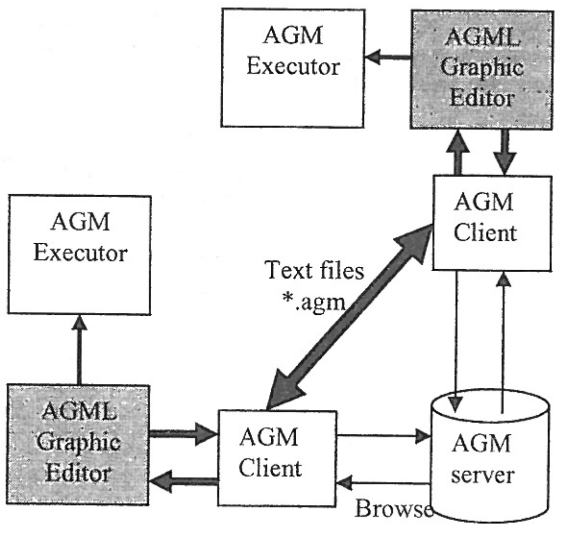

Fig.6 Distributed AOM architecture

The AGMServer application realizes the agents' management in a distributed environment and intermediates the exchanged messages between them. In this context were developed the following application modules:

- AGMServer
  - name server for running agents;
  - intermediates the communication between agents
- AGMPlayer
  - Netscape Navigator plug-in
  - executes distributed AOM in a Web page
- AGMClient
  - responds to the distributed or local agents
- AGMOcx
  - ActiveX control used as a demo to prove the AGMServer application functionality

IV. EXPERIMENTAL TESTS AND RESULTS

The application was locally (server and client on the same machine) and distributed (in a network) tested. For the tests I used an AGML editor with the following distributed facilities implemented:

- register / delete an agent to / from the server
- send / receive messages of type GET, SET, CREATE, JMP, CALL, INSTANCIATE, DELETE.

In order to deal with the errors, the server keeps a history of the agents' requests and of their answers. This report is saved on disk and, when an error appears, might be detected the conditions of this error.

IV. CONCLUSIONS AND FURTHER WORK

The application might be implemented in the future by using DCOM technology. It can be extended to allow many servers communication between themselves in order to maintain updated the list of active agents. This way will increase fiability, the response to errors of the distributed model.

The application might also implement in the future more collaborative servers to deal with the agents. This can bring an increased fiability and better error response.

V. REFERENCES

[1] Gorgan, D.: "Fuzzy Learning and Behaviour in Multi-Agents Based Interactive Systems". *LDAIS Workshop at ECAI96 Conference*, 12-16 August 1996, Budapest.

[2] D. Gorgan: Multiagents Based Modelling in Graphical User Interfaces. In the *Development Consortium of the CHI97 Conference*. Atlanta, USA, 22-28 March, 1997.

[3]http://users.utcluj.ro/~gorgan/res/webdocs/repository/paradigms/Index.htm

[4] D. Gorgan, D.A. Duce: Fuzzy Learning in Multi-Agents Based Interactive Systems. *Rutherford Appleton Laboratory*, Research Report, January, 1997, pp.1-43.

[5] Matei Mihai, Mobile agents distributed execution, Licence Thesis