Interactivity in Active Objects Model

Dorian Gorgan, Vasile Sebastian Cornea
Computer Science Department
Technical University of Cluj-Napoca
G. Baritiu st. 28, RO-400027 Cluj-Napoca
Romania
dorian.gorgan@cs.utcluj.ro

Abstract – **Interactive application based on adaptive behavior, parallelism, cooperation, synchronization and distribution are extremely difficult to be design, developed and maintained. The active objects based modeling is one of the effective solutions. The Active Objects Model (AOM) introduces and experiments the concept of interactor. This paper highlights the interaction techniques supported by the active objects in complex applications. The research concerns on the user interaction by mouse and keyboard, the real time based processing, the operations on local files and directories, the Internet based operations, and the real process interface.**

I. INTRODUCTION

The development of computational models has progressed through phases of procedural model, object oriented, rule based, logic, and constraint based. The procedural model achieves specific tasks by algorithms and data structures. Any procedure has particular inputs and outputs. Data and procedures are public and no any protection against unauthorized access. The object oriented model is based on classes, objects and their relationships. Data is a set of objects as instance of classes. Computationally, the object oriented and procedural model are similar unless the software development performance.

All these models are static and inappropriate for dynamical simulation, animation, visual programming, multimedia, concurrency, parallelism, and adaptive behavior. The active object based model supports dynamic structure and behavior that obviously, simulates the reality better than the static model.

The research work presented by this paper concerns on:

- The interaction techniques supported by the active objects in the user interface through:
  a. Caption and processing of the keyboard and mouse events
  b. Programming and processing of the real time based events
- The access of the active objects to the computer resources such as files, directories, audio and video, and Internet.
- The active objects based control on the real process.

The main concepts and techniques highlighted in the paper are studied and exemplified through the Active Objects Model [1], described briefly in the next section.

II. ACTIVE OBJECTS MODEL

The Active Objects Model (AOM) supports the development of direct manipulation based techniques [2], [3]. The model associates a virtual location to any model entity: active object, static and dynamic variable, behavior, trajectory, virtual position, action, rule and expression. The active objects perform their functionality by a set of associated processors such as behavior, server, presenter, and interactor. The interactor incapsulates the interaction between model, user and computer resources.

The developer describes the application either as Active Objects Modeling Language (AOML) program or as AOM model in the operative memory. The program describes by AOML language the specific structures and evolutions and then is loaded and executed as AOM model (Fig. 1). The model consists of instances of active objects, behaviors, trajectories [4], actions, positions, rules, etc. Furthermore, the executable model can be saved in the AOML form as well. The AOM platform implements the entity structure and functionality (i.e. active objects, behaviors, positions, etc) and mechanisms for message based communication, synchronization, bounding, etc. [1]

The research on the AOM model focuses toward the following main goals:

- Develop a model that supports the description of dynamical behavior through the visual programming techniques [3]
- Experiment and state a consistent set of entities that supports the description of complex and consistent programming tasks [2]
- Experiment the AOM platform to support the interactivity, communication, cooperation, adaptive evolution, distribution, concurrency, parallelism, hyperstructures, visualization, hyperspace navigation, animation, visual programming techniques, and multimedia presentation (graphics, audio, video)

III. INTERACTORS

The interactor is the processor associated to the active object, which performs the user and object interaction, the

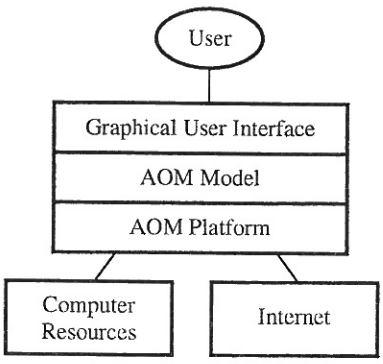

Fig. 1. AOM functional levels

real time based control on the object, and the access of the object to the system files. The interactor gets a location within the bidimensional screen space and provides to the associated object the particular interaction ability. The interactor defines a list of pairs - event and appropriate sequence of actions. When the event occurs the interactor performs the particular sequence of actions. There are four types of events the interactor concerns: mouse, keyboard, timer and stream:

```
Interactor I1{
      position (175,230);
      mouse_event M1 {...},M2{...};
      keyb_event K1{...};
      timer T1 {...};
      stream S1 {...};
};
```

A. Mouse Events

In the AOM model a mouse event means the user executes a gesture by mouse device within a specific area declared as active zone. All mouse based gestures are related to the active zone (e.g. LEFTBDOWN) excepting the MOUSEMOVE gesture. When a user gesture occurs inside the active zone the associated active objects performs a sequence of rules described for an Explicit Trajectory Position (ETP). Any active object can interrogate the interactor about the mouse event through the boolean attributes: *isover*, *isleftbdown*, *ismidlbdown*, *isleftbdow*. Moreover, the caller can get the x and y coordinates of the mouse cursor position.

Let us exemplify the mouse event declaration in the AOML language:

```
mouse_event Lbdown{
  type LEFTBDOWN;
  zone Zone1{
    position(32,57);
      visibility 1;
        graphics G1{
          position (10,7);
          select brush,br1{
            color RGB(0,255,0);
          };
          ellipse (0,0,50,50);
        };
  };
  do{
    type UNCOND;
    rule R1{
      action A1{
        set agent(AS).interactor(IM).
            mouse_event(Lbup).
            zone(Zone1).graphics(G1).
            brush(brsh1).color_green,0;
      };
    };
  };
};
```

The Lbdown is looking for a pressing gesture on the mouse left button. When the user presses the mouse left button inside a green circle with radius of 50 screen pixels the model executes unconditionally the set action of the R1 rule. The action sets on 0 the color_green attribute of the graphics presentation G1 of the active area Zone1 associated to the mose event Lbup, of the interactor IM and the active object AS.

B. Keyboard Events

In the model the keyboard event means the pressing or releasing a particular key. When the user activates the key, the interactor gets a message that encodes the gesture type and the key code. If the message matches the expected event the interactor performs the actions defined for the associated ETP position.

Example:

```
keyb_event K1{
  position(1,1);
  type KEYDOWN;
  key ANY;
  do{
    type UNCOND;
    rule R1{
      position(4,4);
      action Act01{
        set agent(Ts).presentation(pr).
            graphics(G1).drawtext,
            "Value: " + agent(Ts).
            interactor(I1).
            keyb_event(K1).value;
      };
    };
  };
};
```

When the user presses any key on the keyboard device the active object Ts displays the text "Value:" followed by the text key code. The text is the drawtext attribute of the graphics presentation G1.

C. Timers

The interactor can be set for real time events within a programmable time window and rate. The event definition specifies the start and end of the time window, the event rate, the cyclic type (e.g. true, false) of the event, and the timer state (e.g. PLAY, STOP, PAUSE). Through the state parameter the timer could be started, stopped, suspended and resumed. By default the state is STOP.

A valid timer event starts the execution of the actions defined for the ETP position.

Example:

```
timer T1{
  position(1,5);
  start 0.0;
  end 2.0;
  steps 0.01;
  loop TRUE;
  do{
    type UNCOND;
    rule R1{
```

```
        position(4,4);
        action Act{
          set agent(Clock).presentation
            (Hour).graphics(G1).drawtext,
            agent(Clock).interactor(I1).
            timer(T1).tmrminute+":"+
            agent(Clock).interactor(I1).
            timer(T1).tmrsecond;
        };
      };
    };
};
```

The timer T1 has the position (1, 5) relative to the parent interactor. When the timer gets the state of PLAY it generates within the time window (0.0, 2.0), between 0 minutes, 0 seconds and 2 minutes, 0 seconds, at each second (0 minutes and 01 seconds) a real time event. Consequently, the interactor I1 performs unconditionally the Act action of the rule R1. The action displays through the graphics presentation of the active object Clock, the current real time of the timer T1: <minutes> : <seconds>. The AOM model can interrogate the timer attributes through the *get* action in order to obtain the relative time: *tmrsecond* and *tmrminute*, and the absolute time: *year, month, day, hour, minute*, and *second*.

D. File Based Operations

As well as the user and timer interaction the AOM model supports the interaction with the computer file system. The interactor declares the stream events the active objects can operate on:

```
stream S1{
  position(1,1);
  type TEXT_FILE;
  location "C:\autoexec.bat";
};
```

The active objects can navigate throughout the local directories and can read and write the text and binary local files. The model operates on the files and directories through the *get* action that returns information about the operation. For instance:

copyto(<dest_file_name_expression>) – returns true whether the AOM model completes successfully the copy operation, and false otherwise.
deletefile – returns true if the delete operation has been completed.
readln – reads a line from the current text file.
writeln(<text_expt>) – returns true if the model has successfully wrote the text into the file.

Other operations the AOM model performs on the files are: *openread, openwrite, close, iseof, chdir, getfirstfile, getnextfile, getprevfile*, and *isdir*.

IV. EXPERIMENTS

The next subsections highlight some features of interactivity in the AOM model.

A. Display the Current Mouse Cursor Location

This AOM model based application exemplifies the mouse move event, displaying the current location of the mouse cursor (Fig. 2). The application consists of two active objects: Afis and Soricel. The first object has a static location, on a simple trajectory of only one ETP position. Its graphics presenter just displays the cursor coordinates. The second object Soricel, includes the MousePad interactor and the Move mouse event. The object is visible to enable the interactor, but for simplicity of the exemplification, does not have any behavior and presenter.

```
agent Soricel{
  position (4,4);
  state PLAY;
  visibility 1;
  interactor MousePad{
    position (1,1);
    #mouse event entity
    mouse_event Move{
      type MOUSEMOVE;
      do {
        type UNCOND;
        rule R1{
        #action A1 and A2 modifies
        #the text of the presenter
        #by x and y mouse coordinates
        action A1{
         set agent(Afis).
          presentation(pres).
          presentation(MX).
          graphics(GX).drawtext,
          agent(Soricelu).interactor
          (MousePad).mouse_event(Move).
          positionx;
        },A2{
          set agent(Afis).
          presentation(pres).
          presentation(MY).
          graphics(GY).drawtext,
          agent(Soricelu).interactor
```

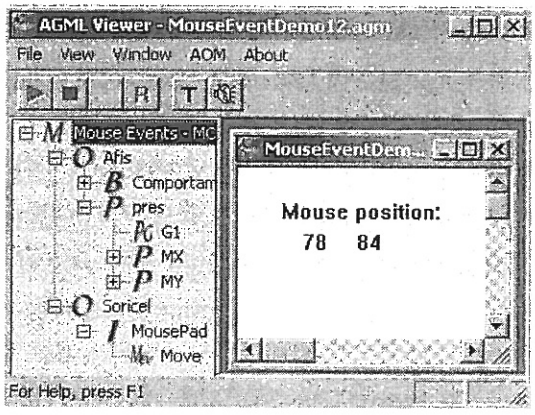

Fig. 2. AOM based application exemplifying the mouse move event.

```
            (MousePad).mouse_event(Move).
            positiony;
            };
          };
        };
      };
    };
  };
};
```

The Afis active object has a graphics presentation (pres) that consists of the graphics (GText) displaying the text "Mouse Pozition", and two subpresentations MX and MY that display the graphics GX and respectively GY by two texts dynamically modified by the Soricel object's behavior.

```
agent Afis{
  ...
  presentation pres{
    graphics GText{
      position (26,29);
      drawtext("Mouse Pozition",
        0,10,110,60,DT_CENTER);
    };
    presentation MX{
      graphics GX{
        drawtext("MX",
          10,30,40,90,DT_CENTER);
      };
    },MY{
      graphics GY{
        drawtext("MY",
          45,30,75,90,DT_CENTER);
      };
    };
  };
};
```

B. Digital Clock

The example highlights the access of active objects to the real time of the system (Fig. 3). The active object (Ceasul) has defined through its interactor (I1), a timer event (T1) that modifies the text of the graphics presenter (Ora) each second. Similarly, it modifies the graphics presenter (pres) to show the date. The graphics presenter consists also of three ellipses and three rectangles to draw the clock body.

For instance, the time seting is achieved by the code:

```
interactor I1{
 ...
 timer T1{
  do {
   rule R1{
    action A1{
     set
      agent(Ceasul).presentation(pres).
      presentation(Ora).graphics(G1).
      drawtext,
      agent(Ceasul).interactor(I1).
      timer(T1).hour+":"+agent(Ceasul).
      interactor(I1).timer(T1).
      minute+":"+agent(Ceasul).
      interactor(I1).timer(T1).second;
    };
   };
  };
 };
};
```

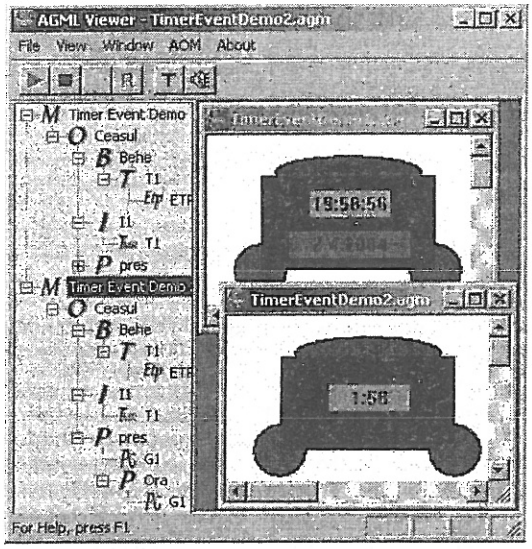

Fig. 3. Digital Clock and Chronometer as active objects in the AOM model.

C. Menu in Graphical User Interface

The active object MenuBar has the interactor I1 of three buttons to support the selection of objects File, Edit and Help (Fig. 4). The File object's interactor has three menu items as buttons, associated as all other buttons in this example, with the pressing event of the left mouse button.

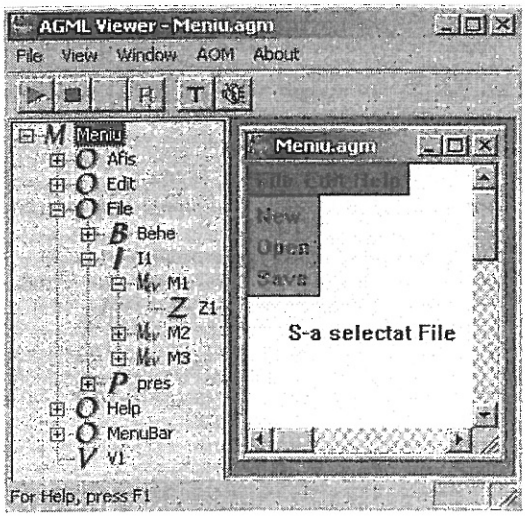

Fig. 4. Active objects based menu interaction.

The agent Afis displays the text related with the selected button.

```
agent MenuBar{
  interactor I1{
    mouse_event Buton1{...},
      Buton2{...}, Buton3{...};
  };
  ...
};

agent File{
 interactor I1{
  mouse_event M1{
   type LEFTBDOWN;
   zone Z1 {
    visibility 1;
    graphics G1{
     textout(5,5,"New");
    };
   };
   do {
    type UNCOND;
    rule R1{
     action A1{
      set agent(Afis).presentation
       (pres).graphics(G1).drawtext,
       "S-a selectat New";
     }, A2{set gent(File).visibility,0;
     }, A3{set variable(V1).value,0;
     };
```

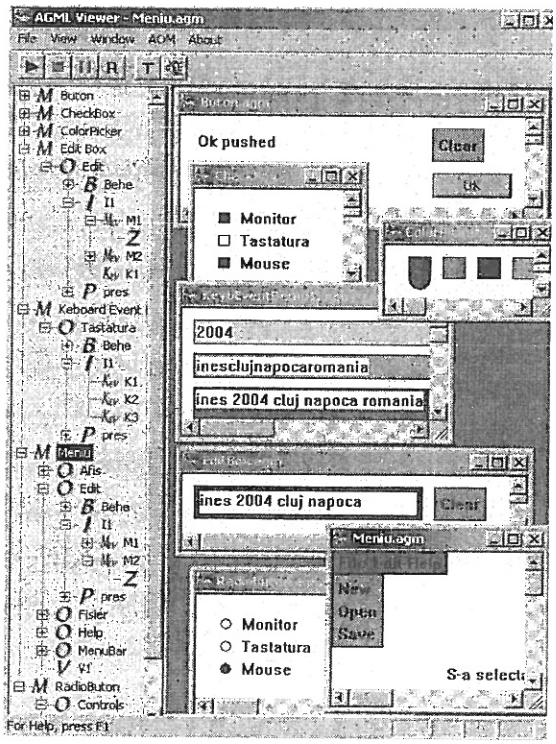

Fig. 5. Examples of various interaction techniques of Graphical User Interfaces.

```
    };
   };
  }, M2{
  # definition of the "Open" menu item
  }, M3{
  # definition of the "Save" menu item
  };
 };
 ...
};

agent Edit{...};
agent Help{...};
agent Afis{...};
```

D. Interaction Techniques in Graphical User Interfaces

The AOM model has experimented (see Fig. 5) the various interaction techniques often used in the Graphical User Interfaces (GUI) such as: push button, edit box, check box, radio button, text area, menu, etc. The AOM model provides for the interface objects the adaptive and flexible structure, appearance and behavior. The model supports the visual programming based development of GUI.

E. Internet Based Applications

Fig. 6 exemplifies an AOM application by which the active object called Internet displays through its presenter four images taken from Internet.

F. Process Interfaces

The experiment concerns on the implementation of process interfaces through state diagrams (Fig. 7). The execution of the diagram is achieved by three active objects W1, W2 and W3. The execution depends onto an input variable „a", which is 1 (true) if the object W1 has passed over a particular area and false otherwise. Each object reports the states and the actions onto a screen object (Ecran). The diagram execution by the three objects starts when a mouse click occurs on the „Start" state.

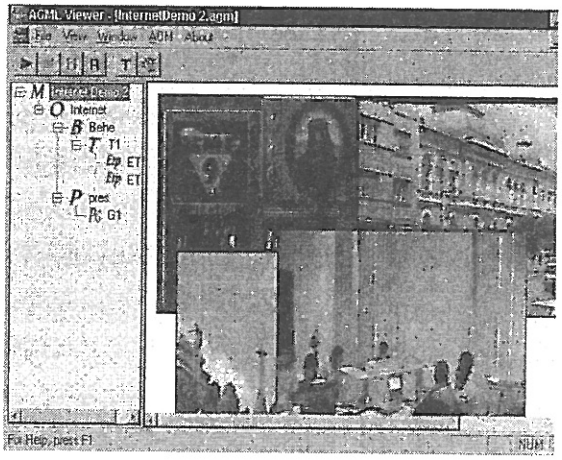

Fig. 6. AOM based application displays four images taken from Internet.

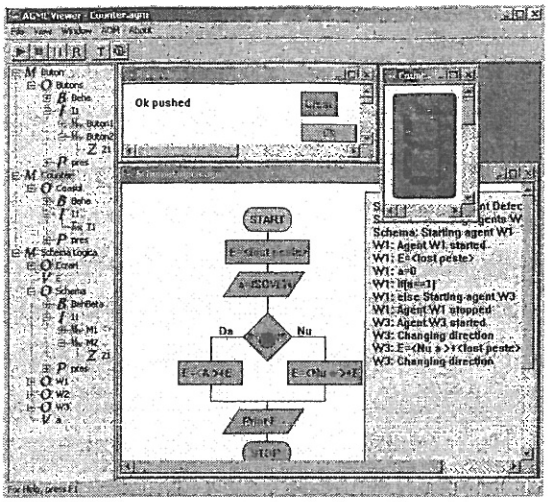

Fig. 7. Active objects support implementation of the process interfaces.

V. CONCLUSIONS

The AOM model provides to the software applications the active objects based interaction. The interactivity is supported through a set of interactors that process the user events carried out by mouse and keyboard, the events given by the programmable timers, and the file based events.

The research work concerns on the implementation of such functionality, the description through the AOML language, and the experiments that prove the concepts and techniques. The work presented in the paper focuses on the concepts and fundamental techniques of interactivity implemented through active objects, which provide concurrency, adaptability, communication, dynamic evolution, visual presentation, distribution, and visual programming based development.

The AOM model has been developed, implemented and experimented in different languages (C, C++, Java, VRML) and software platforms (Windows, Linux and UNIX). The interactor based version has been implemented in C++, under Windows operating system.

The practical results have proved the large potential of active objects model. The functionality, the structure and the behavior is quite easy to be described in AOML language and moreover, developed by visual programming techniques. The development process is rapid and very flexible. The AOM based applications can be modified and experimented by really fast approaches.

The future work will be focused on hierarchical definition of interactors, function specialization, software platform independence, and experiments on various domains of applications.

VI. REFERENCES

[1] "Active Object Model" AOM Documentation and Projects, http://users.utcluj.ro/~gorgan/res /aom /aom.html

[2] D. Gorgan, "Programming Control Structures in Active Objects Model", *Transactions on Automatic Control and Computer Science, vol. 49 (63)*, "Politechnica" University of Timisoara, 2004.

[3] D. Gorgan, "Visual Programming Techniques". *Computer Science Education: Challenges for the New Millenium.* Eds. G. van der Veer, I.A. Letia, Ed. Casa Cartii de Stiinta. pp. 129-142, 1999. http://users.utcluj.ro /~gorgan/res/webdocs/repository/papers/ROC-C99P.zip.

[4] D. Gorgan, D.A. Duce, "The Notion of Trajectory in Graphical User Interfaces". *Design, Specification and Verification of Interactive Systems '97*, M.D.Harrison, J.C.Torres (eds.), SpringerWienNewYork (ISBN 3-211-83055-3), pp.257-272, 1997. http://users.utcluj.ro/~gorgan/res/webdocs/repository/papers/dsv97.zip