# Visual Programming Techniques

**Dorian Gorgan**

Dept. of Computer Science, Technical University of Cluj-Napoca
George Bariţiu 26-28, RO-3400 Cluj-Napoca
Tel. +40-64-194834 ext.165
**Dorian.Gorgan@cs.utcluj.ro**
**http://bavaria.utcluj.ro/~gorgan**

**ABSTRACT.** *The paper emphasizes active entities based modelling of visual programming techniques. Active Objects Model (AOM) and AGent Modelling Language (AGML) are defined and investigated in order to develop flexible and dynamic functionality of interactive components. Therefore the visual programming techniques, distributed knowledge, fuzzy inferences, programming paradigms are analyzed through the concepts and techniques provided by the AOM model. Fuzzy learning and cooperative tasks allow the modelling of the artificial intelligence based approaches.*
**Keywords** *Active objects, rule based behaviour, visual programming, interactive techniques, and fuzzy inference*.

## 1 INTRODUCTION

The last few years have highlighted the importance of interactivity in distributed applications. Especially the development of the web based applications has demanded new interaction techniques, development methodologies adapted to various hardware and software platforms, improved graphical user interface layout and functionality, new languages and models to build up and navigate throughout the virtual web space.

According to this tendency the paper presents the visual programming techniques as a feature of the Active Objects Model (AOM). The main objectives of the AOM based research work are such as: (a) to provide a better user control on both the interface and application entities, (b) to develop active entity based structure and functionality, (c) to develop and investigate visual programming techniques, (d) to provide flexible ways for navigation throughout the distributed virtual space, e.g. databases, web, virtual world, (e) to present the active world by photorealistic graphics, (f) to investigate the distributed knowledge modelling and cooperative tasks performing.

The paper explores the structures of the new web interactive applications and their impact on the graphical interaction techniques. The graphical user interface concepts are briefly presented to emphasise the requirements for new visual techniques and consequently visual programming techniques. The AOM model is presented (Figure 1), as a possible approach to implement the visual programming techniques. Then a few features of AOM are

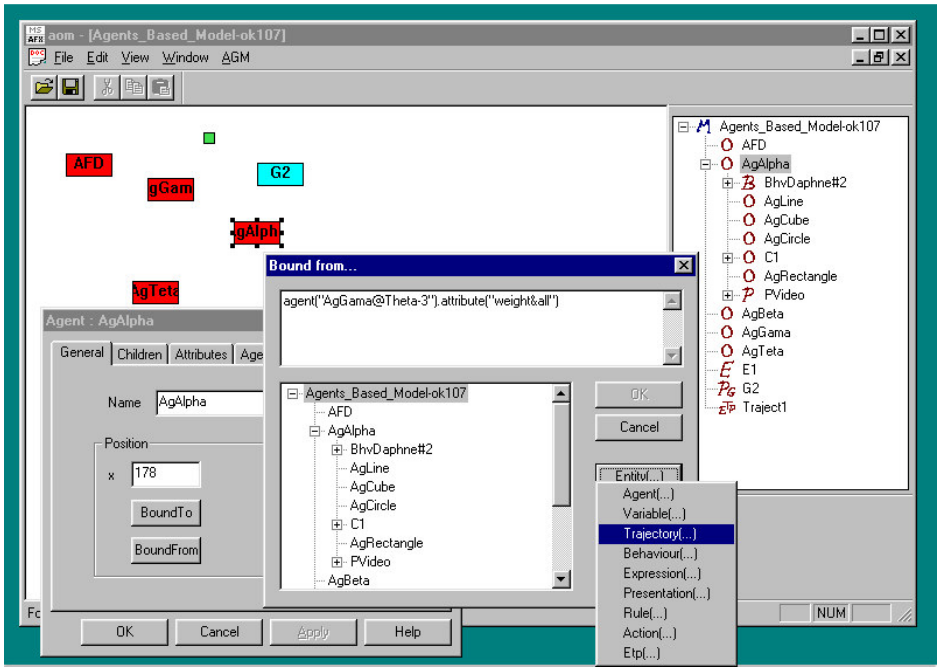


**Figure 1. The AOM model implementation (Visual C++ version)**

exemplified - task modelling, flow control, rule-based modelling, object oriented, fuzzy knowledge and fuzzy inferences, and cooperative tasks. The AOM model uses the AGML language to describe the structure and the functionality of the virtual entities. The AOM model has been investigated through a few implementations, but much more have to be achieved according to the research topics and applications that finally are outlined (see section 10).

## 2 INTERACTIVE APPLICATIONS

The conceptual structure of an interactive application consists of two main components - the conceptual (functional, application) component, and the interactive (user interface, communication) component. The same structure is suitable for any type of application such as local, distributed, client-server, and web. Nevertheless the type of the interactive application influences the distribution of functionality and data structures between the two components and among modules of the computer system. Likewise the user interface can be distributed evolving the same or complementary functionality.

According to the type of the interactive web applications (i.e. static HTML files, dynamic HTML pages, Java assisted editing, and dynamic Java) the functionality of the graphical user interface can be implemented at different levels such as client (e.g. browser, controls, applets, ActiveX, plug-in, etc), web server (e.g. applets, HTML files, CGI, ASP, Java Beans, etc), application server (e.g. scripts, Java programs, etc), and even database server (e.g. stored procedures).

The AOM model has been developed in order to provide for a flexible and sound user control over the application entities. Its concepts support the extension of compatibility with the new software technology, e.g. web applications, Internet browsing, graphics standards, VRML, XML, 3DXML, HTML, OpenGL, Java, ActiveX, and so on. Flexibility is achieved through visual techniques (e.g. direct manipulation, graphical syntax editing, visual representation of entities within the virtual space, etc). The ability of the graphical user interface is extended by visual programming techniques, actually the rule based behaviour developed by visual techniques.

## 3 GRAPHICAL USER INTERFACES

The actual interactive applications are built up around the graphical user interface or they are users oriented. That means the flow control of the conceptual component is directed from the graphical interface that is actually built up around the user's events (i.e. user inputs). Therefore the user control is a key feature of the interactive applications, as well of the web applications. The visual programming techniques assist the user control.

Conceptually the user interface consists of interface objects and interface operations. The interface entities are connected to application objects and operations through internal dialogue and to user through external dialogue. In distributed interfaces the interface entities are themselves distributed too. The user interface design process uses concepts such as dialogue independence, structural modelling, representation techniques, interaction techniques, rapid prototyping, methodology, and control structures, see e.g. [7] and [8].

According to these concepts the AOM model proposes an active object based modelling approach. The interface entities are active objects. Their behaviour through virtual space models the entity's functionality.

## 4 ACTIVE OBJECTS MODEL

The AOM model (see e.g. [5] and [6]), has been developed in order to support: (1) flexible and sound user control on both the interface and application entities, (2) modelling of active entity based structure, (3) development and investigation of the visual programming techniques, (4) virtual space navigation (e.g. throughout the web), (5) photorealistic presentation of the model's entities (e.g. shadows, lights, hidden surfaces, textures, etc), (6) distributed functionality (i.e. of the interface and the conceptual component, clients and servers, etc), and (7) parallel processing (e.g. multithreads, multiagents, multiprocessors).

The design and the implementation of the AOM model has been achieved according to a few structural and functional principles: (a) simple and consistent structure, (b) rule based behaviour, (c) topological and graphical information, (d) metaphorical techniques for visual programming paradigms, (e) dynamic, parallel, concurrent and interactive behaviour, (f) distributed knowledge and cooperative tasks, (g) incremental development, (h) providing for web applications, and (i) integration with web technologies.

Conceptually the model consists of *active* and *passive entities*. In [6] there is an exhaustive presentation of the conceptual AOM model. There are two types of active entities: *active objects* (called alive objects, or *aliobs*), and *variables*. The active entity has an associated behaviour throughout the virtual space.

The passive entities are resources used by one or more active entities. There are the following types of passive entities: *behaviours, trajectories, explicit trajectory positions, rules, expressions, actions (create, delete, instantiate, append, get, set, assign, call, jmp),* and *presentations*. The behaviour describes the aliob's evolution within the virtual space. The aliob advances along the trajectory through the sequence of the explicit trajectory positions (ETP). At each ETP the aliob processes a set of conditioned rules and actions. A set of nine types of actions has been defined and investigated. The aliob performs its own actions or delegates other aliobs to complete the task. The model behaves dynamically. At runtime the aliob can change both the active and passive entities. It may create and modify the structure and the functionality of any other entity.

The model entities have visual presentation in order to support direct manipulation based editing and metaphorical presentation of the application. Topological information embodied into the model's entities supports the visual techniques. Both the structure and the behaviour of aliobs are visually developed.

## 5 VISUAL TECHNIQUES, LANGUAGE, AND PROGRAMMING

The notions of visual and visualization are used in different contexts both in commercial and experimental software products. The *program visualization* is often used in conjunction with the visualization of code, data, and algorithm in a static or dynamic manner, see e.g. [8]. In graphical user interfaces the visual interaction techniques are frequently used. The *visual interaction techniques* imply interactivity through direct manipulation on entities that have a graphical presentation within the user interface. *Visual language* unlike textual language allows the user to specify a program in two (or more) dimensional fashions, see e.g. [9]. The *visual programming* notion means the developer or programmer uses visually built up expressions in the programming process. The user develops by visual approaches the semantics, and generally the structure and the functionality of the program. For example, the syntactic forms are built by picking up the terms from a graphics scene. If the syntactic forms and generally all program entities (i.e. statements, expressions, data structures, flow control structures, and so on) have visual presentations, then the programming language is a *visual programming language*. Finally, the *visual programming techniques* are methods and tools that provide for the development, the execution and the visualization of the program through visual techniques.

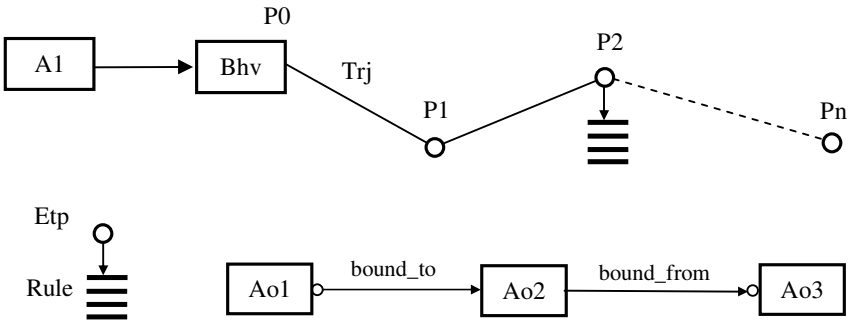


**Figure 2. Visual language of the Active Objects Model**

The AOM model develops a set of elements (i.e. behaviour, trajectory, explicit trajectory position, rules, etc), that may be used to construct the visual programming techniques. For instance, modelling functionality similar with control flow diagram, paradigms such as rule-based behaviour, object oriented programming, simple and complex data structure, state transition based programming, and machine learning techniques.

The model involves elements that provide for the implementation of the visual programming techniques. Therefore the model is analyzed by the three features that define the visual programming techniques - visual language, direct manipulation, and visual programming. A few examples of the programming paradigms argue the capability of the AOM model as visual programming environment.

## 6 VISUAL LANGUAGE

The visual language developed by AOM model consists of the model entities and the relations among them. There is not an obvious one-to-one matching between elements of programming languages and the AOM entities (e.g. such that between C++ and Java). Actually there is equivalence between concepts. For example, the entity behaviour (Bhv) or the entity ETP (Etp) can model the concept of procedure and function (Figure 2). The rule and the sequence of rules (Rule) can model the sequence of instructions. The action models the concept of statement. The evolution over the explicit positions P0, …, Pn of the trajectory Trj models the control flow of the program. The behaviour, and in fact the trajectory, can be linked to the aliob (e.g. A1). Also the entity aliob (e.g. A1, Ao1, Ao2, and Ao3), and the entity variable, and their components model the concept of data structure, i.e. table and list. Likewise, the bounding mechanism models the event-based data flow paradigm. For example, an element of the Ao1 aliob is connected by a bound_to link to an

element of the Ao2 aliob. Similarly, an element of the Ao3 aliob is connected by a bound_from link to an element of the Ao2 aliob, see e.g. [6].

The model entities have visual presentation both in the design and execution stage. The visual language is available only in the design process. The entities are placed into the virtual space and manipulated by the developer using their graphical presentation.

In order to develop the syntactic forms and the semantic relations the editor combines the direct manipulation based selection and form oriented editing.

## 7 DIRECT MANIPULATION

There are two stages of the model development: design and execution. The first involves the editing of the structure and behaviour of the model entities. This stage mainly implies direct manipulation based techniques. The second stage executes the model, which has a metaphorical presentation. The user can interact with the model in accordance with the

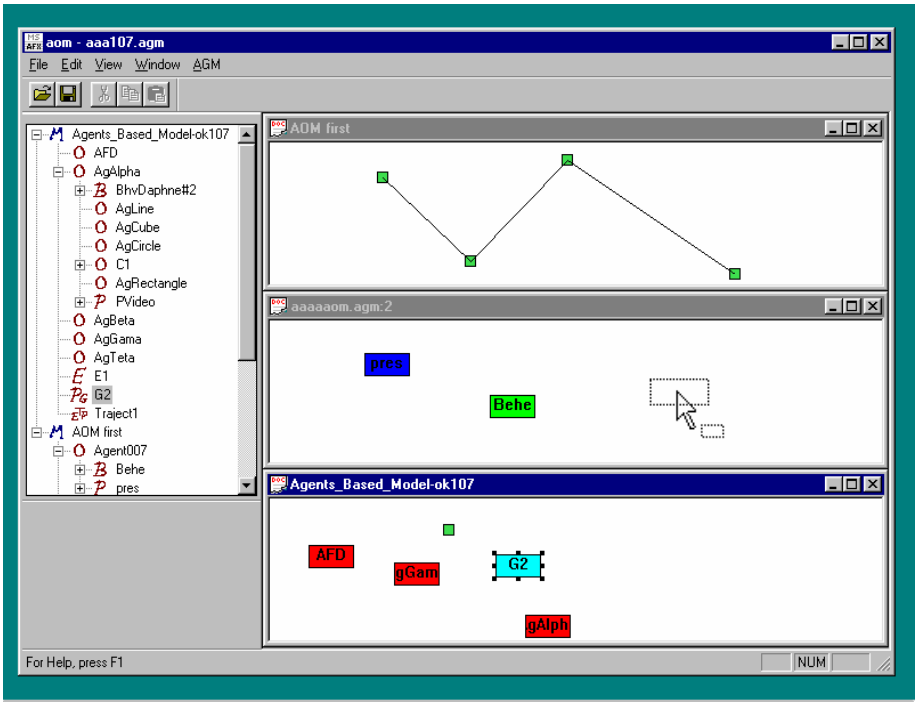


**Figure 3. Direct manipulation techniques in the AOM model. Complex entities may be drag and dropped from one into another model.**

peculiar interactivity described by the behaviour of each aliob. The associated behaviour has been described by direct manipulation in the design stage.

During the design time, the model is stopped. There are no running processors and the entities are displayed using their design time images. The user can have multiple views of the same model at different levels in the model object hierarchy. A modification in the model will be reflected in all views, in order to maintain the consistence at image level. The user also is able to edit several models in the same time. Each model will have one or more views. The user can copy intuitively simple or complex entities by simply dragging and dropping them into the destination model.

In the design stage the user develops by direct manipulation all entities of the model such as aliob, variable, graphical presentation, behaviour, trajectory, expression, rule, and action. The user may create a new entity, instantiate an already created entity, and delete any entity and element.

Let us see an example. The expression is a basic entity in the model. Any action parameter is specified by an expression. The *bound_to* and *bound_from* connections are defined in terms of expressions. An expression consists of a set of logic and arithmetic operators, together with names from the model: entity attributes, fields and constants (Figure 1). The expression editor uses visual techniques in order to construct a valid expression. The legal operators are picked up from a combo box, the entities are chosen from the hierarchy tree and the result is kept in an edit field. A term in the expression can contain also an entity, which is not yet created in the model, but it will be created during the model execution.

## 8 VISUAL PROGRAMMING PARADIGMS

Lets us analyse the AOM model based modelling of the programming paradigms, see e.g. [3], [4] and [9]. Task modelling, control-flow, data-flow, rule based programming, object orientation, data structure definition, fuzzy knowledge and cooperative tasks are a few paradigms presented in the next sections.

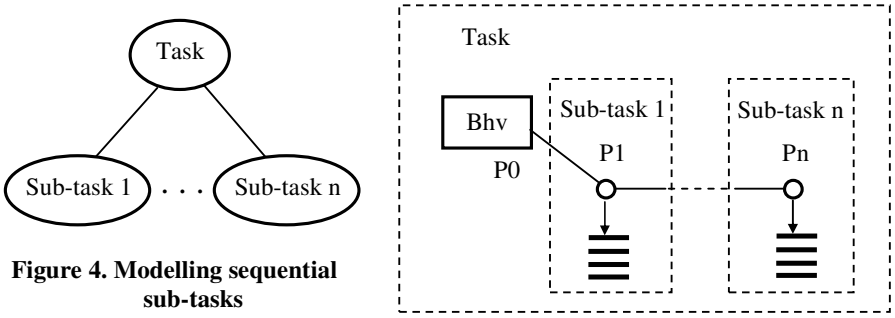


**Figure 4. Modelling sequential sub-tasks**

### 8.1 Task Modelling

Conceptually a task is decomposed into a set of sub-tasks using known methods (e.g. GOMS, CLG, TAG, SSOA, etc). The AOM model allows the modelling of the task in various ways. For example, a sequence of n sub-tasks can be modelled as a sequence of n explicit positions within a trajectory (Figure 4). A set of parallel sub-tasks can be modelled as behaviours associated to the aliob components of an aliob aggregate (Figure 5).

The AOM concepts allow various implementations of a given task decomposition scheme. The previous task tree could be modelled at different level of AOM entities: rule, explicit trajectory positions, trajectory, behaviour, aliob, and model. Is up to developer of application to select the proper modelling scheme. For example, the sequence of sub-tasks can be modelled as a sequence of rules within the same ETP.

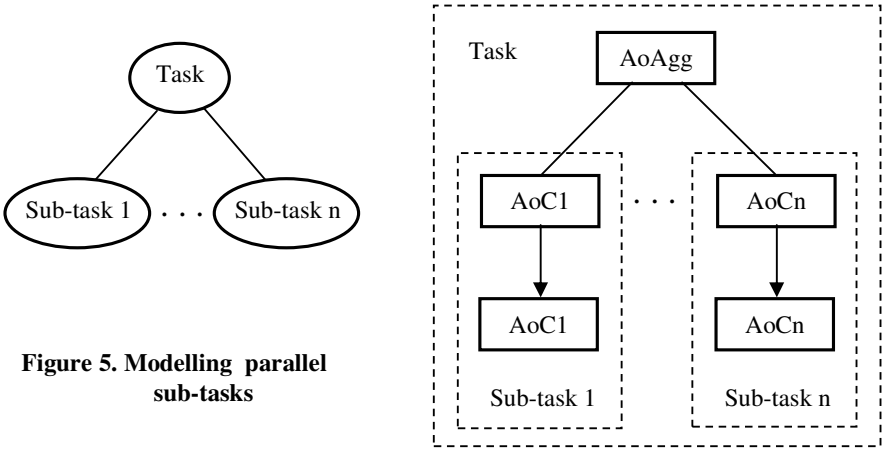


**Figure 5. Modelling parallel sub-tasks**

### 8.2 Control-Flow

The well-known control-flow diagram (Figure 6), actually involves control structures such as repetition (e.g. while_do, repeat_until), iteration (e.g. for_do), branching (e.g. if_then_else, switch_case), and procedure call. The matching between the flow of control and AOM concepts can be accomplished in various ways.

A simple modelling way consists of the using of ETP type, see e.g. [5] and [6]. The ETP entity is defined by type (i.e. uncond, precond, postcond, cond, and iterated), and an associated expression. The type gives indications about the execution of rules embodied within the ETP (e.g. iterated - the evaluated expression returns a positive integer, which specifies the number of loops the ETP executes the set of rules). In figure 6, the ETP positions PS1, PS2, … model the diagram states S1, S2, … The condition "G < alfa" maps onto the condition E that is embodied within the trajectory position PS1. If the expression E

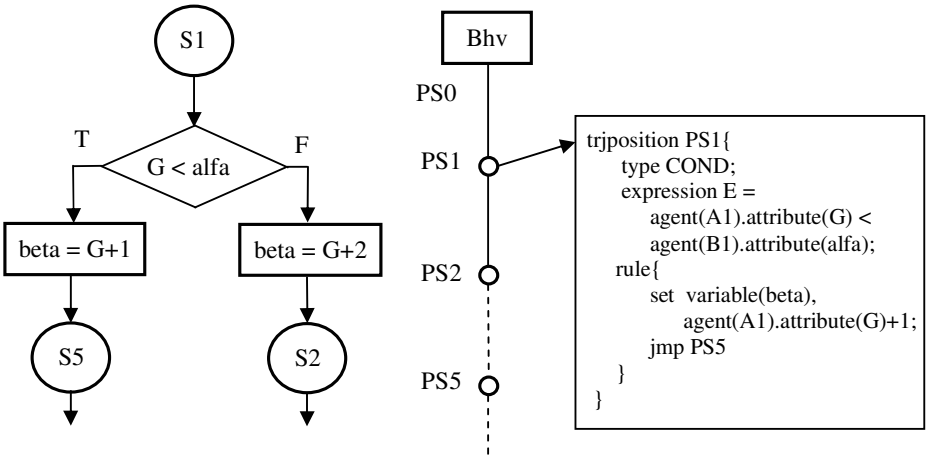


**Figure 6. Modelling a control-flow diagram**

is true the actions set and jmp are executed (see the EGML language in [5] and [6]). Else the next executed trajectory position is PS2.

A direct modelling of the if_then control can be at the level of rule entity by the conditioned actions. Actually the set of actions embodied into a rule is executed only if the rule condition is true. Similar control structures could be modelled at upper levels in order to control the execution of tasks and sub-tasks (see the section 8.1).

Another concept that provides for the modelling of the flow control is the aliob's state (i.e. play, wait, stop, and pause). The state attribute describes the current aliob's state over the associated trajectory. Any aliob can modify the another aliob's state attribute synchronously by actions and asynchronously by the bounding mechanism.

A similar mechanism combines the data-flow and control-flow. The mechanism modifies the passive resources which are linked to the aliob (i.e. behaviour, trajectory, ETP, etc). For example, an aliob changes the name of another aliob's behaviour. Consequently, the affected aliob changes the control flow of its evolution within the virtual space.

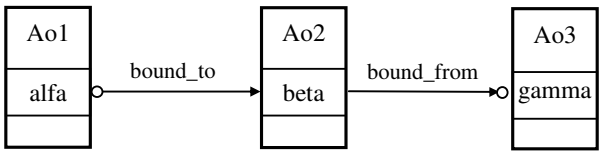


**Figure 7. Data-flow by the bounding mechanism.**

### 8.3 Data-Flow

Data-flow diagrams are most commonly used to represent those dependencies in visual programming languages, see e.g. [1], [2] and [3]. In AOM model the data flows in and out to and from the model entities by two ways. The first method consists of the explicit set and the implicit get actions. The second way is the bounding mechanism, see e.g. [6].

For instance, by the action

set agent(AG2).behaviour, Bhv+agent(AG1).attribute(index)

the name of behaviour of the current aliob is set to the string that concatenates the name "Bhv" with the value of the AG1 aliob's "index" attribute.

The second approach consists of the bound_from and bound_to mechanisms (Figure 7). The attribute alfa of the aliob Ao1 is connected to the attribute beta of the aliob Ao2. Then the attribute gamma of the aliob Ao3 is bound_from to the same attribute beta. It is an event-based functionality. If the attribute alfa changes its value the attribute beta automatically changes its value too, by the bound_to mechanism. If the value of the attribute gamma is requested by a get action it reads the value of the attribute beta through the bound_from mechanism.

### 8.4 Rule Based Programming

The behaviour of the model entities is described as a set of rules. As simple definition the rule is a conditioned sequence of actions. Only if the condition is true the sequence is executed. The effectiveness of such a notion consists in the ability to define generic behaviour. The pattern of behaviour may be instantiated and associated to various aliobs. According to the generic rules new active and passive entities can be dynamically created, instantiated and modified. For example, an aliob can change its behaviour completely creating a new one.

In the fuzzy logic based version of the model the aliob can rationalize according to a set of fuzzy inferences, actually fuzzy rules, see e.g. [6]. The aliobs can improve their behaviour by collecting and enriching knowledge, and by creating new rules (see sections 8.7 and 9).

### 8.5 Object Oriented

The AOM model has been designed and implemented in an object oriented manner. The model itself is an object that can be instantiated. Any entity is an object indeed. An entity can be created, instantiated, and deleted. An instance becomes an autonomous entity. It may inherit the structure and the behaviour of its ancestor or may change any component element (i.e. attributes, behaviour, presentation, etc).

The aliob can be a simple or complex entity. A simple aliob does not have any aliob component. A complex aliob, called aggregate, has a set of aliob components. Any component can be an aggregate. Each component has its private behaviour throughout the virtual space, but its position is relative to that of its parent aliob.

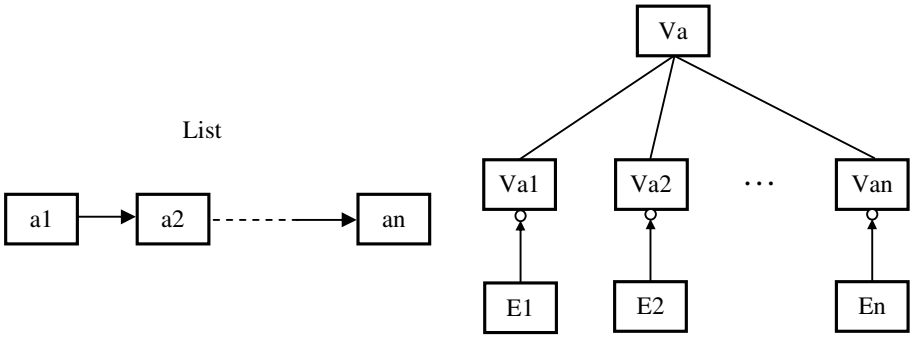


**Figure 8. Data structure modelling. The example of a list.**

### 8.6 Data Structure Definition

The fundamental data type accepted in the AOM model are *integer, float, string, Boolean*, and *fuzzy*. The data may be stored into the aliob's attribute and into the variable's value. The data item may be accessed through two ways: (a) the explicit actions set, get and delete, and (b) automatically. The bounding mechanism, and the dynamic updating of the variable's value, and the evaluation of an expression uses the automatic access.

Data may be organized as active entity components into hierarchical and sequential structures (Figure 8). For instance, the component variables Va1,…,Van of the active aggregate Va model a list.

### 8.7 Fuzzy Knowledge and Cooperative Tasks

The aliobs have knowledge about their domain. The aliob's attributes are used to store the knowledge. The aliobs cooperate in order to achieve a task using the private knowledge of other aliobs. Therefore, the cooperative task is accomplished through distributed sub-tasks, which use distributed knowledge.

There are two types of the aliob's attributes: *crisp* and *fuzzy*, see e.g. [6]. The *membership function* models the value of the fuzzy attribute (Figure 9). The fuzzy attributes are accessed through explicit actions (i.e. the actions get and set) and bounding mechanism. For example, complex *fuzzy inferences* can be performed:

(if x is A1 and y is B1 then z is C1) or (if x is A2 and y is B2 then z is C2)

The arguments x, y and z are crisp values, and A1, A2, B1, B2, C1 and C2 are fuzzy attributes. The fuzzy inference returns the crisp value z (i.e. float) that may be used by the model entities (e.g. within fuzzy and non-fuzzy expressions).

In the AOM model the arguments of the previous inference may be the elements of different entities. Likewise, the inference is performed by an aliob that cooperates with others, which are the owners of the arguments. For example in the AOM model the first term of the "or" expression is achieved by the following fuzzy expression:

```
agent(AGC1).attribute(C1).fuzzy(
      min( agent(AGA1).attribute(A1).crisp(x),  agent(AGB1).attribute(B1).crisp(y) )
)
```

A very useful feature of the AOM model is *fuzzy learning* [6]. The fuzzy learning refers to the facility of the AOM model to improve the fuzzy inferences and aliobs' knowledge. The model provides a set of operators and actions that allow to operate on fuzzy knowledge and to improve the learning mechanism. Knowledge improvement is an explicit operation that adds a *knowledge quantum* to an existing knowledge, i.e. fuzzy attribute, and membership function.

The notion of knowledge quantum is defined as an elementary membership function w(Δw, r), where Δw is the maximum of the knowledge quantum. r is a set of weights that define the *propagation rate* of the knowledge quantum to related values from the universe of discourse. The notion of knowledge quantum allows a membership function to be built up through a sequence of set actions, i.e. within the learning process.

Another notion imposed by the *saturation phenomenon* of the learning mechanism is the *forgetting operator*. The AOM model uses as forgetting operator the normalization of membership function. Consequently each saturated fuzzy value is decreased with the same capacity of assimilation Δw.

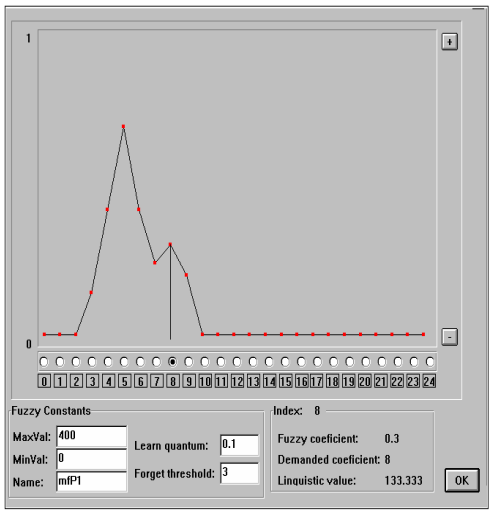


**Figure 9. Direct manipulation based editing of the membership function.**

## 9 IMPLEMENTATION

The AOM model has been developed in an experimental form at the Technical University of Cluj-Napoca (Borland C++, Visual C++, and Java version), and at the Rutherford Appleton Laboratory, UK (Visual C++, the first multi-threads based version). Being a multithreading application, AOM is considerably consuming the system resources. In order

to get acceptable performance, the application must run on powerful machine (e.g. at least a 100 MHz processor, 32MB RAM memory, Windows NT).

However, the model performances are acceptable, despite of the entity name-addressing mode used. Addressing by name offers flexibility to the model, the entity access can be realized even from outside the process boundaries.

The graphical performances can be enhanced if techniques like OpenGL or DirectX are used. A better version of the AOM editor should provide a manager for undo/redo operations. The undo manager allows the user to go back along the chain of operations performed, up to the initial model configuration.

## 10 RESEARCH TOPICS AND APPLICATIONS

The AOM model is intending to be a framework that supports the development and the investigation of concepts, notions, techniques, and methodologies in the domain of distributed interactive applications.

The previous developments and investigations argue the capability of the AOM concepts. The model is convenient to be used in the research in the following fields of interest:

- Photorealistic presentation of animated virtual world
- Flexible behaviour of the interface objects (e.g. controls, windows)
- User control over application entities
- 3D interfaces to applications such as databases, internet navigation, scientific visualisation, simulation, animation
- Distributed interfaces, distributed applications
- Aliob based multi agents systems
- AOM mobility throughout the internet
- Distributed task modelling
- Cooperative learning
- Fuzzy knowledge
- Aliob based authoring tools
- Visual programming techniques
- Compatibility with new technologies – standards, formats, and languages

## 11 CONCLUSIONS

Through its achievements the AOM model has managed to investigate and state a set of consistent and encapsulated entities. It is a minimum set necessary to describe the dynamical, parallel and concurrent evolution of the model. The Agent Modelling Language provides for portability over different software and hardware platform. The language allows the extension and compatibility with new web technology. The new tendency of programming environments to the visual techniques and particularly to visual programming has been emphasized since a few years. Therefore the AOM model has investigated the

programming paradigms in order to prove the ability of the developed concepts in this field. Fuzzy knowledge and learning techniques has been used to model distributed knowledge and cooperative tasks. The consistent set of actions, operators, data structures, and AGML extension has been developed related to the fuzzy logic concepts.

In spite of the above mentioned achievements the model has to be extended and tested through more investigations. An important deficiency is the control of user's action at runtime. This goal is more difficult if it is an animated model within a photorealistic presentation of the virtual world. Likewise, more programming paradigms have to be developed such as recursiveness, management of parameters, program modularization, definition of generic functionality, mobility over distributed systems, etc.

## 12 ACKNOWLEDGMENTS

Part of the work described in this paper was carried out at Rutherford Appleton Laboratory, UK. I am grateful to Prof. David A. Duce for advising the conceptual design of the AOM model. I thank Cătălin Druță, Sergiu Dunca, Sorin Sâmplăceanu, Ioan Velea, and Mihai Paul who worked on the implementation and assessment of the concepts.

## 13 REFERENCES

1. Baroth, E., Hartsough, C. (1995) Visual Programming in the Real World, in *Visual Object-Oriented Programming, Concepts and Environments*. Burnett, M., Goldberg, A., Lewis, T. (eds.) Manning, 21-42.
2. Bird, R., Wadler, P. (1988) *Introduction to Functional Programming*, Prentice Hall, NY.
3. Burnett, M. (1994) A Classification System for Visual Programming Languages*, IEEE Visual Programming*.
4. Glaser, H., Smedley, T. (1995) PSH - the next generation of command line interface, in *Proceedings of the 11th International Symposium on Visual Languages. VL'95*. IEEE Computer Society Press, 29-36.
5. Gorgan, D., Duce, D.A. (1997) The Notion of Trajectory in Graphical User Interfaces. *Proceedings of the DSV-IS'97* workshop, pp. 289-305, Granada, Spain, 4-6 June 1997, and in the *Design, Specification and Verification of Interactive Systems '97*, Harrison, M.D., Torres, J.C. (eds.), SpringerWienNewYork, 257-272.
6. Gorgan, D., Duce, D.A. (1997) Fuzzy Learning in Multi-Agents Based Interactive Systems. Rutherford Appleton Laboratory, Research Report, January 1997, 1-43.
7. Hartson, H.R., and Hix, D. (1989) Human-Computer Interface Development: Concepts and Systems for its Management. *ACM Computing Surveys*, Vol. 21(1), March 1989. 5-92.
8. Myers, B.A., Zanden, B.V., Dannenberg, R.B. (1990) Creating Graphical Interactive Application Objects by Demonstration. *The Garnet Compendium*: Collected Papers, 1989-1990. Carnegie-Mellon Univ., Aug. 1990. 95-114.
9. Rekers, J. (1995) *Visual Languages*. Course Notes. Informatics Department. Leiden University. 1-274.