A VISUAL ENVIRONMENT FOR MINE WARFARE MODELLING AND SIMULATION

SULEYMAN SEVINC

TECHNICAL REPORT 481
MARCH 1994

BASSER DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF SYDNEY
A VISUAL ENVIRONMENT FOR MINE WARFARE MODELLING AND SIMULATION

Suleyman Sevinc
Basser Department of Computer Science
University of Sydney NSW 2006
Australia

ABSTRACT

This article describes a research and development effort in computer modelling and simulation in support of naval mine warfare countermeasure activities. The significance of mine warfare has become especially apparent after recent military experiences gained in Middle East. The issue is even more important for countries like Australia whose well-being is particularly dependent on sea trade. The research described in this article has been conducted at the University of Sydney in collaboration with Maritime Operations Division of Materials Research Laboratory in Melbourne.

The emphasis of the research is to provide a visual environment supporting object-orientation using which models for entities involved in mine warfare can be developed. As it will be apparent in later sections, mine warfare involves many levels of hierarchies of models ranging from strategic and tactical models to individual component models. Therefore the solution proposed should address the interworking of models at these various but inter-related levels.

The visual nature of the development environment is aimed at improving the productivity of the modellers so that they can focus on the models rather than idiosyncrasies of the model development environment. It also allows visual verification and presentation of simulation results for the end users of the models developed using the environment. With these things in mind, Microsoft Visual Basic (VB) was selected to be the environment for it minimizes the amount of work needed to develop such an environment, interfaces very well with other modern software applications such as databases, spreadsheets, etc. and for its event-driven programming model. This was possible because VB is designed to be an extendable environment. Since VB and Microsoft Visual C++ can share the same extensions, the extensions made under this project are also usable by Visual C++ thus open the way for a full-scale object-oriented support. In this article, however, we will focus on VB extensions and their use within VB environment.

---

1This research is supported by Maritime Operations Division, Materials Research Laboratory, Defence Science and Technology Organisation in Melbourne, Australia.
BACKGROUND

Mine warfare has regained its importance after recent military experiences in Middle East. The issue is even more sensitive for countries like Australia whose economic well-being heavily depends on sea trade. Recognition of this fact is apparent in the establishment of a dedicated Mine Warfare Systems Centre (MWSC) by the Royal Australian Navy (RAN) to provide the prime focus for all operational aspects of RAN's mine warfare activities. Modelling and simulation forms an important part of the MSWC's decision support systems required to enable optimum use of limited Mine Countermeasure assets (Watson and Ryan, 93).

Mine warfare models are used for purposes of planning, evaluation and training. In principle these models may be analytical models, discrete event algorithmic/procedural models or logical models (Sevink 91a). Analytical models have their weaknesses in that they are highly abstract and ignore many details even for moderately complicated scenarios in order to produce a closed form formula. Logical models are usually behavioristic and are usually substantially more abstract than discrete event models. Discrete event models though, can represent the reality in an amazing level of depth but may be slow. The environment described in this article emphasizes discrete event models, though it is capable of facilitating other kinds of models.

In general, there are two phases of mine clearance; locating the mines and neutralizing them. Locating mines involves use of detection technologies such as sonars and neutralising mines may involve remotely operated vehicles and diver teams. It is worthwhile to note that not every mine detected is a real threat to assets as some mines are false and are placed for confusion. The overall operations require coordinated activities of assets possibly in more than one geographical area at a time.

(Lawson and Richardson 1994) describes mine sweeping in a graphical way designed as a game between a player and an Umpire. The sweeping phase of mine clearance involves passes over an area contaminated by mines (see Figure 1). The distance between passes, mine detection probabilities, mine activation probabilities, navigation errors and even the environmental conditions may affect the efficacy of the sweeping operation. The game designed by Lawson and Richardson is aimed at training skills of a player in mine sweeping given various mine types, locations, probabilities, etc. The Umpire places mines in a random fashion whereas the role of the player is to design a sweeping strategy for better results.

Mine Warfare Model Structure (see Figure 2) involves four levels of models. The first level of models involve individual components used in mine warfare. These include mines, remotely operated vehicles, divers, ships and sonars. Since these individual component models act independently from one another in reality, the simulation models should preserve this autonomy. For example, a mine is a standalone entity which comes into contact with a ship at an unknown future time. Similarly a ship is an entity which does not necessarily know about the existence of mines within its vicinity. From the
simulation point of view this requires support for synchronizing the activities of these individual models and yet keeping them independent of one another. This issue will further be examined in later chapters as the structure of our simulation environment becomes clearer.

The second level of mine warfare involves coordinated activities involving the individual models of level 1. For example, a set of divers form a team which is coordinated in neutralizing mines. Another example would be mine hunting which requires sonar model outputs to be analyzed to locate mines and then hunt them down using either remotely operated vehicles or other techniques. Other second level activities involve a collection of level one entities. A mine field is such an entity comprising of many mines, false and real, distributed according to a pattern to a geographical area. Such mine fields sometimes act like a single entity by either tuning the parameters of individual mines or using other techniques.

The third level of activities involve higher level functions such as mine clearance force-threat models. At this level, the outputs of the level two entities are coordinated. For example, knowledge of mine sweeping models may be used along with the minefield models to model a convoy running through a minefield. Retrospectively, level three might be interpreted comprising of models where many-to-many interactions between models are common place. Meanwhile level two might be interpreted as collection of models which interact in a one-to-many manner.

Level 4 is the top of the hierarchy and represents strategic models. Strategic models should allow representation of tactical decisions and resource allocation. The ultimate goal of building strategic models is to be able to ask "what if .." questions to the system for specific scenarios (Sevne 93). Strategic models are representations of situations and command responses to those situations. For example choosing a mine field and specifying the number of tracks for mine sweeping is a tactical decision. Similarly the command may choose to use convoy models and compare the two approaches for that specific scenario.

**Modelling and Simulation**

Discrete event modelling and simulation is a cheap and effective way of representing real or imaginary systems on computers. It may be said that any real system has its equivalent computer program which acts like it. The equivalent computer program is said to be a model of its real counterpart. So, the art in modelling and simulation is to device that program for a given real system. The challenge, therefore, for simulation methodologists like myself is to provide environments in which such models can be created, modified, stored and experimented with.

Much work has been done in modelling and simulation (see Tanir and Sevne 94). Many environments have been developed usually as an extension of some programming language (Zeigler and others 1989, Sevne and Tanir 90, Sevne 91b). More recently
object-oriented models have gained popularity even though the concept was part of Simula for some 25 years (Birtwistle and others 1979). We ourselves have developed an object-oriented model based on Common LISP Object System (Sevinc 91b). The point of that environment was to device a simulation engine which was provably correct. The theoretical framework chosen was that of Zeigler’s approach. Even though such environments are rather powerful in modularizing system models and organizing them hierarchically, they necessarily fell short in their support for model development as they were not ‘visual’ development environments.

In this research we have taken the view that visual development environments would provide modellers with the ability to focus on their model development rather than having to deal with idiosyncracies of the underlying programming language.

In simple terms, there are three types of people involved in modelling and simulation. Methodologists are people like us who design a model development environment. Modellers are people like engineers who actually write models using the methodologies and environments developed by methodologists. End users are people who use those simulation models to assess a particular situation. For example in mine warfare modelling and simulation project we are describing in this article we are the methodologists and we program in C++ to provide the modelling environment to be described. What we develop becomes a part of VB and engineers from the Department of Defence become the modellers developing models for war scenarios using the environment we develop for them. The modellers in this case can develop their models using either VB or Visual C++ using the framework we provide them with. The anticipated users of this system are naval commanders in charge of mine warfare or trainees who will become in charge of such operations. Their concern is primarily with the current situation that they have to deal with and do not necessarily know about the details of the models they deal with.

Since the object of this special issue is visual development environments we refer the audience to (Zeigler 1976) for more information on simulation.

INTRODUCTION

Microsoft Visual Basic:

First released in 1991, VB has become a popular visual program development environment for Microsoft Windows. VB (see Figure 3) has many features for program development; graphical debuggers, extensive help facilities, project file management features, options for customizing screens, text search and substitution facilities and many other modern features to support a good programming environment (See Microsoft VB manuals). It is probably worthwhile to note that VB allows production of standalone executable files.
What makes VB interesting is its programming model which is basically called event-driven programming in Microsoft terminology. VB supports event-driven programming using a set of objects which users are provided with. Users are provided with buttons for example (see Figure 4) which can respond to a set of predefined events such as mouse click, double click, dragging the button, etc. Examples of other objects include check boxes, option buttons, text boxes, picture boxes and many others. Each one of these objects can respond to a set of pre-defined events depending on their type. Text boxes can respond to key press events, option buttons can respond to click events, picture boxes can respond to resize events, etc. Programmers typically create these objects visually and by double clicking on the objects they are given an edit window where they provide the code for the behaviour of the object by filling in pre-defined procedures for each event that the object can respond to.

What makes VB even more interesting is that it is an expandable framework. Many third party developed objects are available in the market place and use of those objects are exactly the same as the use of the standard objects of VB. Such objects are developed for database integration, image processing, networking, voice and almost anything that today's computers are used for. The third party objects can only be developed using C and C++.

VB engine supports VB objects using a message passing mechanism. A click event, for example, is captured by the VB engine and forwarded to the event handler of the appropriate object. Third party developed objects operate on the same principle. Basically VB engine passes to them any events that they need to process. It is this ability that makes VB a truely extendable framework.

Figure 5 shows a VB object. Basically an object is defined by three sets of things; events, methods and properties. Visual Basic supports a small set of generic components which on the screen appear as resizable icons. Each control has a number of properties that define objects appearance at a given time such as height, colors, etc. Another set of properties are related to object's purpose such as caption of a command button, list array of a list box, a property showing the selected area of a text button, etc. Some of object properties are available for read only, some available at run time and some may not be available at run time. In this article we will only describe properties that are not part of standard package of VB. Interested readers should consult VB manuals.

Each object can receive a set of events. A procedure exists for each event and can be written by the programmer to handle that event. Default behaviour of the system usually does as little as possible. Click, drag, resize are examples of a few events. Each object receives a different set of events depending on the purpose of the object. In addition a set of procedures, called methods, are defined for some objects. Methods are like event procedures but are initiated from within the code and can not be changed by the programmer whereas event procedures can be defined by the programmer. Move, which moves an object to a given point, and print, which prints on a picture box, are examples
of methods. Many other methods exist in VB resulting in a powerful visual development environment with graphical debugging and editing facilities.

The concept of an object in VB is a good basis for representing models of mines, divers, remotely operated vehicles, ships, etc. A mine model, for example, has a number of event procedures such as `detect_signal`. This event is fired when a ship comes within a distance of a mine model. In addition to specific models to address mine warfare modelling and simulation, a set of generic models are developed which can be used in modelling and simulation of any system.

Development of Third Party Objects

Development of a new object type for VB can not be done using VB. We use Microsoft Visual C++ for this project. Objects that are developed for VB can also be used with Visual C++ thus opening the way for a fully supported object-oriented and visual modelling and simulation. The development of a new object requires a definition of an icon which determines the look of the object. This look can be designed to be changeable and in fact we developed one ship object type which can appear in many different forms; destroyer, submarine, mine hunter, speed boat and a tanker depending on the modeller’s choice. A set of properties can also be defined to characterize the object and its look. Mine object type for example has a property which indicates its actuation probability and one which indicates its strength if it goes off. A set of events can also be defined. Our ship model for example has an event procedure `get_hit` which is called when a mine explodes within its vicinity. This event procedure is intended to be filled in by the modellers depending on what kind of ship they are modelling. To complete the definition of a new VB object, a set of methods may be defined. For example, it is possible to passivate a mine by calling a method called `passivate` which disables the mine.

**VISUAL MODELLING ENVIRONMENT**

Before we describe the implementation of the environment in detail, we would like to describe how to use it.

Let us first develop a simple scenario in which a ship is moving towards a mine (Figure 6). Figure 7 shows the starting screen where the user needs to draw a ship on a VB form representing the window in which the simulation will run. The user can choose an appropriate ship type according to which the appearance of the ship changes (in the figure it is a destroyer). The modeller can resize the ship icon and place it on the form. The second step involves creating a mine object by simply clicking on a mine object and drawing it on the form. Properties of each object can be changed, e.g., a property of mine object indicates whether the mine represents a real mine or a false mine. Other visual appearances are also available, such as removing borders gives moving objects a more natural look.
The second step involves developing models for the scenario. This is as simple as filling in the event procedures. The mine model has a detect_signal event procedure through which it receives signals as a ship comes close to it. It has an internal logic which decides whether to go off or not; in this case it is simple and the mine goes off when the signal level exceeds a certain amplitude. When the mine model goes off, this is detected by the central agent which is not visible to the modeller at this age. The ship model has a sail procedure by which it navigates in whatever direction it is programmed to be. It also has a get_hit procedure which is called by the simulator when a mine within its vicinity goes off. The get_hit procedure decides how much harm is done to the ship depending on the strength of the explosion, the distance from the mine and the type of the ship. The ship and the mine are totally unaware of one another until they come in close contact. The message exchanges between the models are handled by a simulator. Figure 8 shows the event procedures described above in pseudo-VB code.

Using this approach it is possible to develop many scenarios; Figure 9 to figure 11 show three possible scenarios.

Once a scenario has been developed, it is possible to turn off the visual features of the models to accelerate processing. If the visual feature stays on, the ship icons move around the screen consistently with their actual route and they disappear as they hit a mine and explode. The visual features are good for training and verification of models developed.

Individual Models, Concepts

As discussed in earlier chapters, level one of mine warfare modelling and simulation hierarchy involves individual models such as models for mine types, ship types, divers, remotely operated vehicles and alike. In the environment described in this article these models are implemented as extended VB objects. A typical modelling study starts with identification of object types involved, drawing these objects on the screen, setting object properties related to their visual appearance and actual types and writing event procedures representing objects' behaviour. Level two of mine warfare modelling and simulation hierarchy involves coordination of these components. For example the detect_signal event procedure of a mine should be activated when a ship enters in its area, or a get_hit event procedure should be activated for a ship model if it happens to be intruding into a real mine's area and that the mine goes off. In this section we would like to describe these two levels more closely.

Level 1 Models:

Level 1 models are atomic in nature and are designed to be standalone entities. They are abstracted using a set of properties, a set of event procedures and a set of methods. Each model has a visual appearance which may be changed by the modellers. Figure 5 shows such a generic component. We will describe ship model and mine model types in a simplified way to help a better understanding of the environment.
A mine model has a number of properties. Some of those are related to the visual appearance of a mine such as whether it has borders or not, background color, etc. Other properties are aimed at defining the behaviour of a mine model. For example a property Mine_Definition states whether the mine is a true mine or a false mine. Another property called Amplitude is the strength of the explosion if a mine goes off. Yet another property Distance indicates the distance in which the mine will cause some damage to the passing sea traffic.

In the simplified mine model, we have defined one event procedure; detect_signal. Detect_signal receives a signal each time it's called and assesses the signal to decide whether to go off or not. A method Explode is defined which causes a mine model to generate a shock wave and then disappear. Figure 12 shows a mine model its properties and event procedures.

A simple ship model has a ship type property which allows a number of choices including a tanker, destroyer, speed boat and a mine sweeper. The Speed property sets the maximum speed of a ship. It has two event procedures; Sail and Get_Hit. Sail event procedure is designed so that the ship can calculate its route when it is needed. Get_Hit event handles the situation when there is a mine exploding within the vicinity of the ship. A method Destroy totally damages the ship and the ship disappears from the simulation models. Figure 13 shows a simple ship model.

In addition to these we developed other models for mine warfare modelling and simulation including divers, remotely operated vehicles, sonars, etc. We also developed a set of generic components which can be used for general modelling and simulation. These generic models are designed in such a way that their appearance can be set by the modellers depending on the nature of the modelling exercise. We will not describe these models here as the intention of the article is to present the mine warfare modelling and simulation.

Level 2 Models

The generation of first level models is a starting point to facilitate simulation. Now these models must be synchronized in time and space. For example, a mine model should start receiving signals from a ship exactly when it enters its area or a ship should be hit by a mine exploding only if it is at the right place at the right time. There are indeed serious problems with the computational efficiency of handling such synchronization in such an environment where models move freely and come in contact with one another sometimes momentarily. An example of such a momentary contact is when a ship briefly enters a mine's effective area and immediately leaves before the mine decides to explode. Some mines apparently possess the intelligence of delaying their activity in anticipation of a convoy or a larger ship.
The synchronization of our models are done by a simulator which keeps account of the location of each object. Discrete event modelling dictates that there are times of no activity, therefore, the global clock can advance safely to the next time. This is not strictly accurate for this case. The route through which the objects move is also important as if the route is contaminated by mines the object may never be able to make it to its final destination. Therefore a mechanism beyond a simple event scheduling and management is required. First we will describe the simple case of managing events and more advanced issues will be addressed later.

Event Scheduling and Event Management

Event scheduling is done by a simulator (Figure 14) which maintains a priority queue structure based on next event times of individual components. The simulator grabs the first from the event queue and places a message to the object which placed that request. When the object receives this message it calls the appropriate event procedures to process it. When the object finishes its processing it reregisters itself with the simulator by sending a message back comprising of the next event time it needs to be activated, the event procedure that needs to be called and its identity so that the simulator can send it a message at an appropriate time. This message is put in the event queue by the simulator.

This simple mechanism will work in general but not in every case. Let us assume we are dealing with a ship model which needs to get from point A to point B. Typically the ship model may choose to compute how long it will take to make the trip and place itself on the event queue to be waken up at the time that it is supposed to reach point B. Clearly the ship has no knowledge of the state of the ocean between points A and B. It may be that the area is contaminated by mines and that the ship can never get to point B without substantial delays to say the least. So we need a mechanism which will enable us to deal with this situation.

One possible solution to this problem is to do discrete time simulation rather than event simulation. In this approach every object would have to reassess its state every so often such as every minute depending on the sensitivity of the modelling study. This is an expensive way of dealing with this problem but has the capability of producing accurate results (see Sevine 90 for a discussion on model simplification). The approach we have taken is a mixed-mode approach.

Each object is defined to be a point in the space with a radius indicating the area which it is enabled to monitor. In other words, each object has a circle in which it is effective. When such a circle moves on plane, it produces a channel which can be approximated by straight rectangular paths (see Figure 15). The requirement on the models' part is that they are responsible for informing the simulator of their routes between the last event they process and the next event they will be producing. If there is an intersection between the routes of any two objects, ship-to-ship or ship-to-mine, each object is informed of the potential collision and asked to provide the timing details of the route. If the simulator determines that the objects will have to have intersecting routes on the time axis as well,
it changes their time of next event time to just before the coincidence of routes starts and
the simulation reverts to a discrete time mode. In other words, a maximum limit is placed
on the next event time for the objects involved and they are only allowed to proceed
within certain interval. This addresses the problem we cited in a more efficient way than
using pure discrete time based techniques.

Another kind of interaction between objects requires communication channels be
established between objects. For example two ships communicating with one another is
such a case. The environment supports this kind of intercommunication between models.

Strategies and Model Storage and Retrieval

Strategic models sit right at the top of the mine warfare modelling and simulation
hierarchy. A strategy is perceived to be a scenerio describing a situation. The way the
strategic models are handled in the environment is that the scenerios are unrolled into
levels of models interacting with one another.

The handling of scenarios are still at the planning stage. We are experimenting with
various representation schemes such as natural language expressions and relational-like
models. In this approach we assume many thousands of cases are studied during peace
times and are loaded on a disk. Each case addresses a scenerio with certain parameters to
be set. The generic description of a scenerio helps choosing the right case which then is
followed by setting the parameters. If say 1000 cases are loaded on a disk, with the
potential parameter value settings, it may handle millions of different situations thus
resulting in a very powerful environment for decision making. The cases we refer to here
are stored as VB executable files and they are self sufficient to run. See (Sevick and
Zeigler 1988 for a model storage/retrieval technique).

CONCLUSIONS

The article described a visual environment for mine warfare modelling and simulation.
The environment is built by extending Visual Basic environment of Microsoft. VB is
chosen because of its strengths in interfacing to other applications, ease of program
development, graphical debugging and editing facilities and its event-based approach.

ACKNOWLEDGEMENTS

I like to express my gratitude to Drs. Don Richardson, Peter Ryan and Richard Watson
for their valuable input to this research. I like to thank other DSTO staff involved in mine
warfare modelling for their valuable comments on this research. I also like to thank my
research assisstant Dr. Kamil Bagde who does the code development for most parts of the
above project.
REFERENCES


Lawson K.R., Richardson, D. The Australian Mine Sweeping Game, MRL TEchnical Note MRL-TN-659, Department of Defence DSTO Materials Research Laboratory, Melbourne.

Microsoft Visual Basic Manuals, Microsoft Press.


Figure 1. Mine Sweeping Passes over an Area
Figure 2. Levels of Mine Warfare Models

- Individual Models
- Many-to-Many Models
- Force-At Threat Models
- Strategic Models
Figure 3. The Opening Screen of VB
Figure 4. A Command Button's Properties and Events
Figure 5: A Generic Object

Properties

Methods

Events

Object
Figure 6. A Ship Model Moving Towards a Mine Model
Model in Figure 6

Figure 7. Starting Point in the Development of the

Mine Warfare Related Objects

[Diagram showing a grid with a small area marked]
Figure 8. Event Procedures for the Scenario Described in Figure 6

<table>
<thead>
<tr>
<th>If Signal &gt; Threshold then</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mine Detect Signal</td>
</tr>
<tr>
<td>Mine Passive</td>
</tr>
<tr>
<td>Mine Explode</td>
</tr>
<tr>
<td>Ship Destroy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Do Damage Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ship Get Hit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Move on a Straight Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ship Sail</td>
</tr>
</tbody>
</table>
Figure 9. A Submarine Model is Moving into a Mine Field
In a Coordinated Way

Figure 10. A Number of Models Moving Into a Minefield
Figure 11. A Speed Boat is Opening a Channel Thorough A Minefield
Figure 12. Simpilified Mine Model

**METHODS**

- EXPLODE
- DETECT SIGNAL

**EVENTS**: 
Figure 13. Simplified Ship Object in Various Forms
Can Exchange Messages

Figure 14. VB Engine Passes Messages to Objects. Objects send messages to VB Engine.
Figure 15. Simplification of a Route of a Moving Object