5. THE DOTNETSIM MODELLING ENVIRONMENT: THE STENCIL COMPONENT

CHAPTER OVERVIEW

DotNetSim's modelling environment prototypes a software application for modelling DES systems. It uses the Event Graph paradigm [107, 108] and widely-used Microsoft packages as the basis for a diagrammatical modelling environment, within which DES systems are modelled for an event-based simulation. The prototype shows how different components can be linked to produce a modelling system.

DotNetSim links different Microsoft applications to draw Event Graphs and to collect the necessary modelling data so that the model can be executed over time by a simulation engine. It captures the application logic of the models in a diagrammatic form and stores the data that describe their dynamic behaviour. Event Graphs were chosen because they offer a minimalist notation to represent a DES system as a set of interrelated events which trigger state changes upon the system. The composition of the interrelated events represents the logical structure of the system whilst the state transition represents its dynamic or temporal behaviour. This conceptual framework is well suited to event-scheduling simulations.

The implementation of the DotNetSim modelling environment consists of specially-written VBA components that create an Event Graph stencil of shapes which provides the notation for capturing the logic and the dynamics of DE models in diagrams and relational databases.

This chapter introduces the DotNetSim modelling environment and sets its objectives and conceptual framework. Then, it goes on to describe and comment on the implementation of the DotNetSim prototype by focusing on the stencil component. The stencil component of the DotNetSim modelling environment implements the Event Graph extended modelling notation in Microsoft Visio™, defining the properties and the behaviour of abstract shapes which represent the events and the edges of that diagrammatical notation. It also guides the user through a menu of commands that perform the main operations on a stencil file.

Finally, some comments are put forward on the implementation and reusability of the stencil component.

5.1. OBJECTIVES OF THE DOTNETSIM MODELLING ENVIRONMENT

The DotNetSim modelling environment is used to explore how widely-used Microsoft development tools may be integrated to support the development of software for discrete event modelling. Our interests focus on the derivation of a modelling environment by the integration of generic software functionality in a 'generalise-specialise' development strategy (see section 4.2). Thus, this modelling environment is prototyped to investigate the customisation and integration of a diversity of applications for the purpose of developing tools for DES modelling. It is also used to study the possibilities for increasing the transparency of black boxed commercial simulation software by resorting to general purpose programming languages.

The DotNetSim modelling environment is a prototype customisable software component for building diagrammatical representations of DE models. The prototype uses the Automation provided by the Microsoft interface to exchange data across different applications, so as to allow simulation models to be built from data managed by different packages and, then, to display the modelling data in different packages. A set of VBA programs were written to exchange data among Microsoft packages by applying the principles of OOP, i.e. one application instantiates the classes of other applications and manipulates these objects simply by invoking the appropriate methods.

Fig. 5.1 is an overview of the DotNetSim modelling environment functional structure. The system to be simulated is modelled into an Event Graph by resorting to the modelling notation provided by the stencil component. Diagrams are drawn directly onscreen by the user who drags and drops the events and edges from the stencil onto the Visio drawing page. Alternatively, they may be drawn automatically

by Visio™ which reads the model's descriptive list from a file based in Excel™ or other application that supports Automation. Additional modelling data are collected and attached to the diagram so that a ready-to-run description of the model is placed into a relational database. The model description can be displayed in Excel™ or other applications. Once complete, a simulation engine written in a more powerful programming language reads the model by instantiating the Visio object model and invoking the appropriate methods. An event-based simulation engine written in C#, with some VB.NET components, is described in detail in chapter 7.

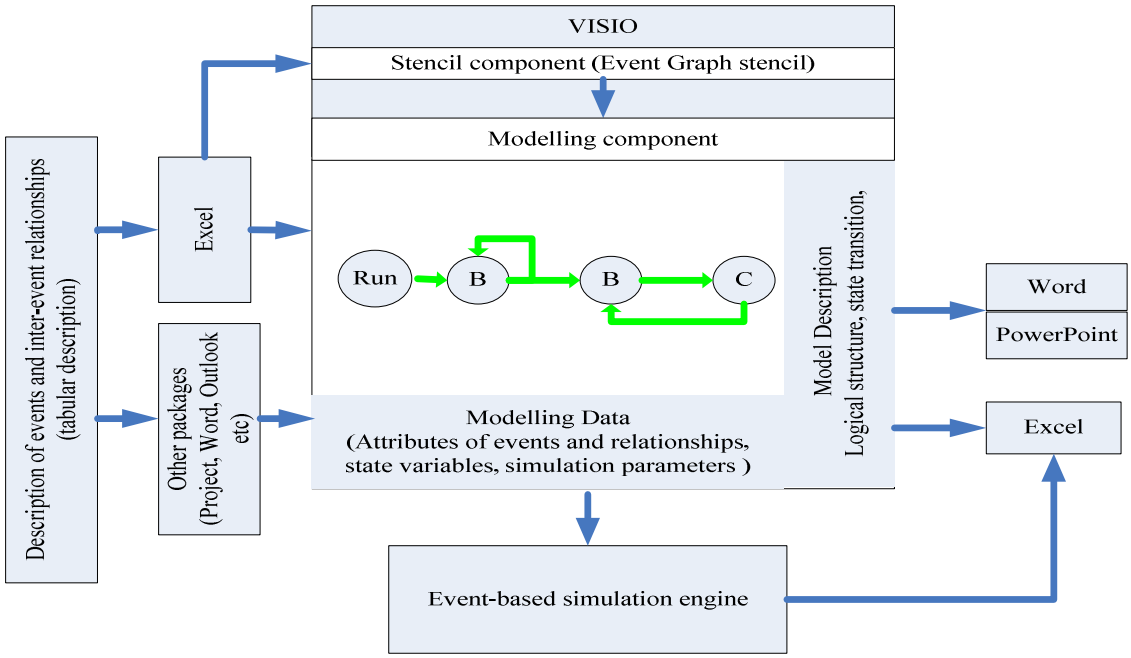

Fig. 5.1: Overview of the functionality of the DotNetSim modelling environment

Though DotNetSim prototypes an Event Graph modelling environment and an event-based simulation engine, other schemas can be substituted, i.e. other graphical modelling techniques can be implemented and run by simulation engines based on other worldviews.

5.2. THE DOTNETSIM CONCEPTUAL FRAMEWORK: EVENT GRAPHS

The DotNetSim modelling environment uses the Event Graph paradigm [107, 108, 17] introduced by Lee Schruben (1983) to model the dynamics of discrete event simulation systems. Event Graphs represent the logical structure of a DES system and its temporal behaviour by abstracting the events which trigger the state transition. Events are networked in scheduling relationships, i.e. causal relationships that are temporised, so that events occur in the future, some time after the occurrence of their logic precedents. Many of the scheduling relationships are conditional and might be cancelled depending on the state of the system, e.g. on the availability of resources.

Event Graphs are a minimalist representation of these event relationships, which have been shown to have enough generality to correctly emulate any discrete event system [103]. Despite the high level of abstraction required by the relationships between events, when compared to the movements along successive processes, Event Graphs are easy to use and generally facilitate the comprehension and representation of discrete event systems [17, 18].

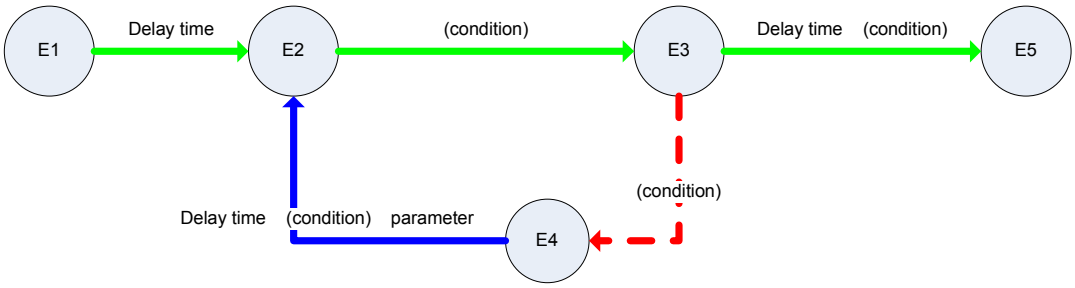

Fig. 5.2: Scheduling and cancelling relationships between events in an Event Graph

Fig. 5.2. illustrates how pair-wise relationships are represented in Event Graphs. A set of events (vertices) and their causal relationships (arcs or edges) are networked to represent a simulation model. Arcs represent scheduling or cancelling relationships

between events and are a means to transport data such as head conditions, delay times and parameters between pairs of events.

5.2.1. BASIC CONCEPTS

The basic modelling concepts of DotNetSim derive from the definition of a DES system (see section 2.1.3.) as a set of states which transit dynamically in response to the occurrence of time-stamped events. States are described by a set of variables which represent those attributes of the system that best serve the objectives of the simulation study. The system state changes in response to events, hence events functionally map onto state variables. An event changes at least one of the state variables. As an example, Table 5.1 lists the events and the corresponding state transitions in a discrete event simulation of a multiple server queue. The run event corresponds to the initialisation, in which the number of customers who arrive at the system (N) and the number of customers queuing (Q) are set to 0 and the number of idle servers (S) is set to the number of existing servers.

	State variables		
Event	N- Number of arrivals	Q- Length of the single queue	S- Number of idle servers
Run	0	0	4
Arrival	N=N+1	Q=Q+1	
Start the service		Q=Q-1	S=S-1
End the service			S=S+1

Table 5.1: Changes of state in a four-server queue

When an arrival occurs, one more customer joins the queue so that N and Q are incremented by one; when a service starts there is one less customer queuing and the server becomes busy whilst the end of a service increments the number of idle servers.

Other variables may be derived from those state variables to collect statistics on the performance of the system as time-varying statistics.

Not only do the events change the state variables, but they also schedule their logical successors: the start of a service, for example, schedules its end. Consequently, given a basic modelling notation [107, 108, 17], a system can be mapped onto a network of events, each of which maps onto a particular state of the system. Events and inter-event relationships are, respectively, the vertices and the edges of the network. Events and state transitions are instantaneous and atomic, i.e. they consume no time and their execution is performed in one indivisible operation. On the other hand, one edge represents the time that elapses between the occurrence of two adjacent events.

5.2.2. BASIC AND EXTENDED MODELLING NOTATION

DotNetSim adopts the minimal modelling notation of Event Graphs. Fig. 5.3 displays the basic modelling constructs, the event and the scheduling edge:

I) Event A

II) Unconditional scheduling edge: event B is due to occur after t units of time upon the occurrence of event A

III) Conditional scheduling edge: event A schedules event B to occur t units of time later if condition (i) holds.

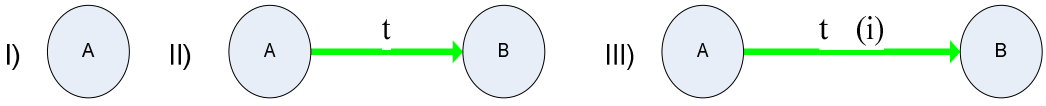

Fig.5.3: DotNetSim basic modelling notation

A DotNetSim diagram of a multiple server system with a single queue and many servers of three types is diagrammatically represented in Fig. 5.4. The state variables are the queue length (Q) and the counters of idle servers of type A (Sa), type B (Sb)

and type C (Sc). The servers are allocated in the order ABC, i.e. first, all servers of type A are allocated, then all servers of type B are allocated and finally all servers of type C are allocated. No delay times were considered and arrivals are generated at a rate ta.

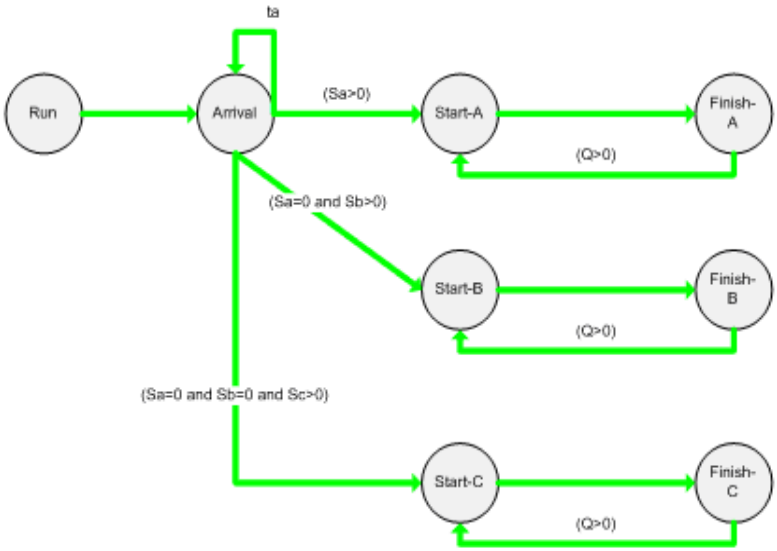

Fig. 5.4: DotNetSim representation of a multiple server system with three types of servers (partly derived from Schruben, 1995 [107])

Since events map onto state changes [107, 108], Fig 5.4 corresponds to the state transition shown in Fig. 5.5.

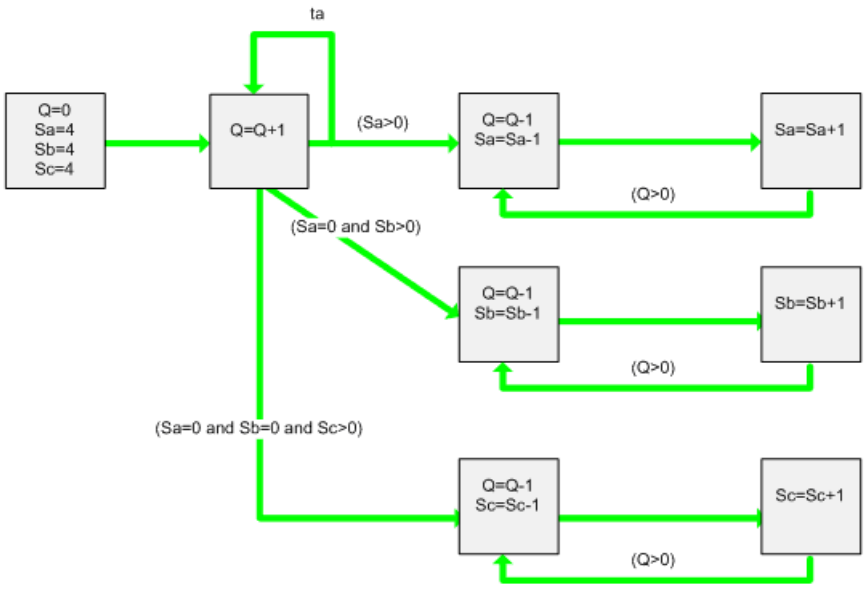

Fig. 5.5: The state transitions associated with the multiple server system represented in Fig.5.4

Each event changes one or more state variables. The run event sets Q to 0 and the counters Sa, Sb and Sc to 4, i.e. the number of servers of each type; the arrival event increments Q by 1; the start events decrement Q and the corresponding counter; the finish events increment the corresponding counters.

Event Graphs and DotNetSim both include extended modelling constructs to support the cancellation of already scheduled events and to allow parameters to be passed between events. The former materialise as cancelling edges, which are the inverse of the scheduled edges and may be due, for instance, to a machine failure. They may also occur dependent on a condition, but they have no delay times [107, 108, 17]. Their graphical representation is displayed in Fig. 5.6, in which:

I) Unconditional cancelling edge: event A cancels the first scheduled occurrence of B

II) Conditional cancelling edge: event A cancels the first scheduled occurrence of B if condition (i) holds

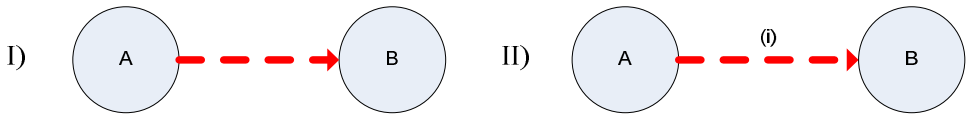

Fig. 5.6: Extended modelling notation (cancelling edges)

The second extension, the parameterisation of events, allows an event-vertex to represent a collection of identical events. It works as if the identical events are elements of an array of events, each one being identified by an index. At a particular point, the event-vertex represents the event whose index is equal to the actual parameter passed by the originating event. For example, Start(n) is the Start event that engages a server of type n. Whenever Start is scheduled, the originating event passes the actual value to n.

Fig. 5.7 shows the constructs for conditional parameterised scheduling and

cancelling edges. Unconditional parameterised edges are also defined.

I) Parameterised scheduling edge: if condition (i) holds, event A schedules event B for t units of time later and passes j to k.

II) Parameterised cancelling edge: if condition (i) holds, event A cancels the event B whose parameter k is equal to j.

Fig. 5.7: Extended modelling notation (parameterised edges)

Parameterised events can be used to model multiple server systems of many types. Fig. 5.8 represents the arrival of spectators at a football match at the security checking system with an inter-arrival time ta. They move sequentially from security barrier 1 to n. There are many servers at each barrier. The state variables are Q(i), representing the number of spectators queuing for barrier i and S(i) which represents the number of idle servers at barrier i. The run event initialises the number of idle servers per barrier, which is an input parameter of the model.

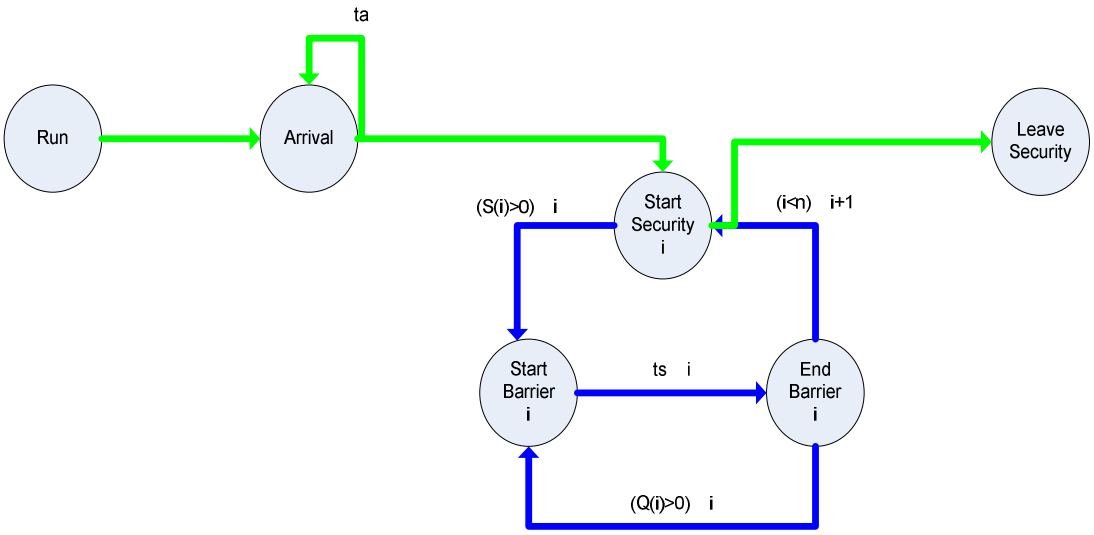

Fig. 5.8: A security checking system consisting of a series of n different barriers. Each barrier is served by many servers and has its own queue

5.3. DOTNETSIM MODELLING ENVIRONMENT: THE STENCIL COMPONENT

The stencil component of the DotNetSim modelling environment defines a stencil with a collection of abstract shapes which implement the Event Graph extended modelling notation. This stencil is saved as a Visio stencil file, the Event-Graph.vss, and it is the only one which is automatically loaded into the DotNetSim's modelling component, hence it contains the only notation that can be used to represent the DE models diagrammatically.

The Event Graph stencil is the abstract definition of the properties and methods of the shapes required by the Event Graphs. It is, therefore, a collection of classes of shapes, or Masters in Visio™ terminology, which are reusable by object instantiation whenever the shapes are dropped into diagrams.

The stencil component is a set of programs written for DotNetSim in VBA for Visio™ and saved in a Visio drawing file. It performs the main operations on a stencil file, namely the creation of a new stencil, its customisation by adding and removing masters, and opening and closing operations. It also allows the user to set the folder path, where this stencil can be located by other Visio files.

5.3.1. THE STENCIL MENU

A stencil menu lists the commands that perform the basic operations on a stencil file. It is generated and appended to the Visio menu bar at runtime. It is worthwhile referring to the implementation of this menu as it is our first step towards the customisation of Visio solutions by exchanging data between Visio™ and Excel™. Fig. 5.9 lists the VBA statements for getting a reference to the cell A1 of sheet 1 of the

active Excel workbook from within the Visio™. This assumes that a reference to Microsoft Excel™ 11.0 Object Library has been set to the Visio project.

```
Set xlapp = GetObject(, "Excel.Application")
Set wb = xlapp.ActiveWorkbook
Set R = wb.sheets(1).Range("A1")
```

Fig. 5.9: VBA statements to get references to the active Excel instance, the active workbook and range A1 of sheet 1 from within Visio™

The implementation of the stencil menu also tests the implementation of event handlers triggered by actions performed on Visio shapes. Fig. 5.10 shows a VBA handler for double clicking a Visio shape. Double clicking the first shape of the active page of the current active Visio drawing document invokes the procedure ThisDocument.MenuiteminExcel.

```
Sub doublingclick()
  Set sh = ActivePage.Shapes(1)
  thisproc = "RUNMACRO(""ThisDocument.MenuitemsinExcel")"
  sh.CellsSRC(visSectionObject, visRowEvent, visEvtCellDblClick).FormulaU= thisproc
End Sub
```

Fig. 5.10: VBA handler for double clicking the first shape of the active page of the current active Visio drawing document

On opening the EGstencil.vsd, an event handler displays two possible sources for the list of commands that will be converted into the Visio Stencil menu. On double clicking the Visio box, the Stencil menu is automatically generated from a list of commands stored in a Visio array; on double clicking the Excel box, the list of commands stored in a range of an Excel worksheet is converted into the Stencil menu, as illustrated in Fig. 5.11. Hence, editing the appropriate Excel worksheet leads to a different menu list.

	A	B	C
1	Level	User command	Procedure
2	1	Set path to stencil	SetpathtoStencil
3	1	New Stencil	
4	2	Create stencil	CreateStencil
5	2	Add masters	
6	3	Event	Eve
7	3	Schedule	Sch
8	3	Cancel	Can
9	3	Parameterised schedule	Parsch
10	3	Parametrised Cancel	Parcan
11	2	Remove masters	RemShapes
12	1	Open stencil	OpenStencil
13	1	Close stencil	CloseStencil

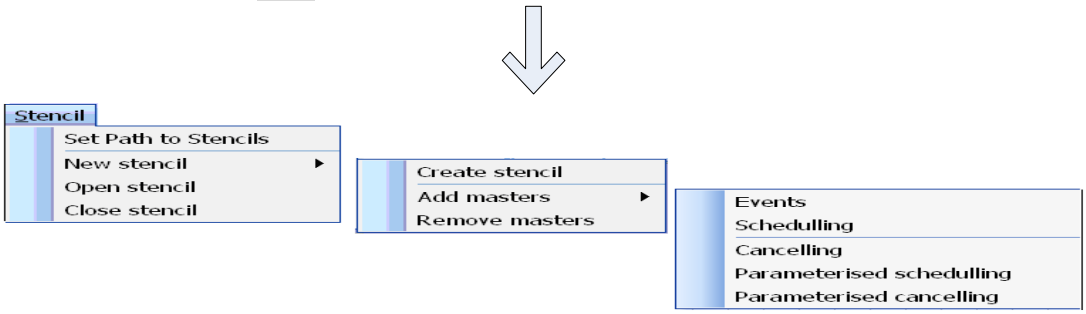

Fig. 5.11: A list of commands placed in Excel™ is converted into a Visio menu of commands

While generating the menu, the list of procedures stored in Excel™ is checked against the VBA procedures of EGstencil.vsd. Those which are missing are listed to remind the user that the corresponding functionality is not available. Thus, the user may adapt the menu to match the needs of each solution by selecting the commands and the corresponding procedures. As they are stored as text in an Excel file, the list of commands can, for example, be re-written so that they are expressed in the user's native language.

5.3.2. MASTERS AND SHAPES

The Event Graph stencil consists of five master shapes, the Event Graph masters,

which correspond to the event, the two scheduling and the two cancelling edges of the extended notation of the Event Graphs as described above. Fig. 5.12 shows the masters that comprise the Event Graph stencil.

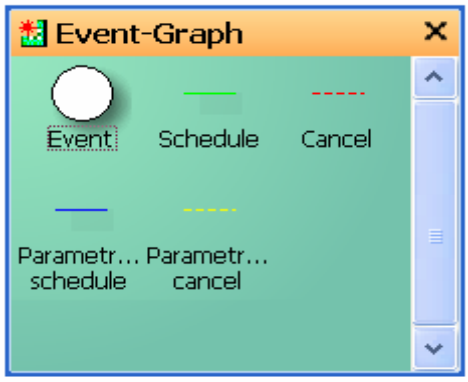

Fig. 5.12: Snapshot of the Event Graph stencil

A master is a class that defines the properties and the methods of a shape. The event master, for example, defines properties such as the circular shape, the diameter of the circle, the fill pattern and the name of the master. Its definition also includes the methods that can be invoked on it, such as those which define the attributes of the events shaped out of this master. For example, some attributes of the event 'Arrival of customers' are defined by the corresponding master and set when the event is placed in a diagram. The Event Graph masters derive from the definitions of the Circle and the Dynamic Connector in the Basic Shapes stencil (Basic_M.vss), built-in in Visio™. From the base masters they inherit two groups of properties and methods: those of the master, which characterise each master within the stencil and those of the shape, which characterise the shape that the master takes.

VBA modules were coded to drop the base masters onto a Visio drawing page, to set the values of the relevant inherited properties and to add the definition of the properties which characterise the events and the edges of the Event Graphs.

Master properties such as the master's name or icon identify each master within the stencil. The Event Graph masters inherit these properties from the base master. The names, indices and prompts of the masters were set to suit the Event Graph modelling notation. Table 5.2 lists the values assigned to those properties.

MASTER	NAME	INDEX	PROMPT
Event	Event	1	State transition
Scheduling	Schedule	2	Scheduling the connected event to occur after a delay time if condition holds
Cancelling	Cancel	3	Cancelling the first occurrence of the connected event if condition holds
Parameterised scheduling	Parameterised Schedule	4	Passing an argument while scheduling the connected event
Parameterised cancelling	Parameterised Cancel	5	Cancelling the occurrence of connected event whose parameter matches the passed argument if the condition holds.

Table 5.2: Names, indices and prompts of each Event Graph master

Accessor and mutator methods, implemented as Let and Get properties, are provided for reading and setting the values of these properties. Other properties, such

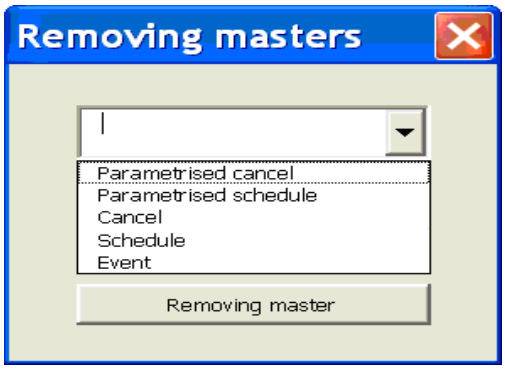

Fig. 5.13: Removal of the Event Graph masters

as the icon size and shape and searching keywords can be overridden. Drop and Remove methods can be invoked on each master to add or remove them from the

stencil. Fig. 5.13 shows the form that allows the selection of a master to be removed.

Shape properties, such as the height and the width of a shape, the fill pattern, x and y coordinates and connection points, among many others, characterise the shape that a master takes. The Event Graph masters inherit these properties from the Circle and the Dynamic Connector of the base stencil. The properties of a shape are listed in the shape sheet, which is a set of two-dimensional tables displaying the predefined properties and default values of the selected shape. The Event Graph masters override the default values of some of them, namely the diameter and fill pattern of the event master, the connector's ending arrow and the line's attributes of the edges masters. Fig. 5.14 shows part of the shape sheet of the event master.

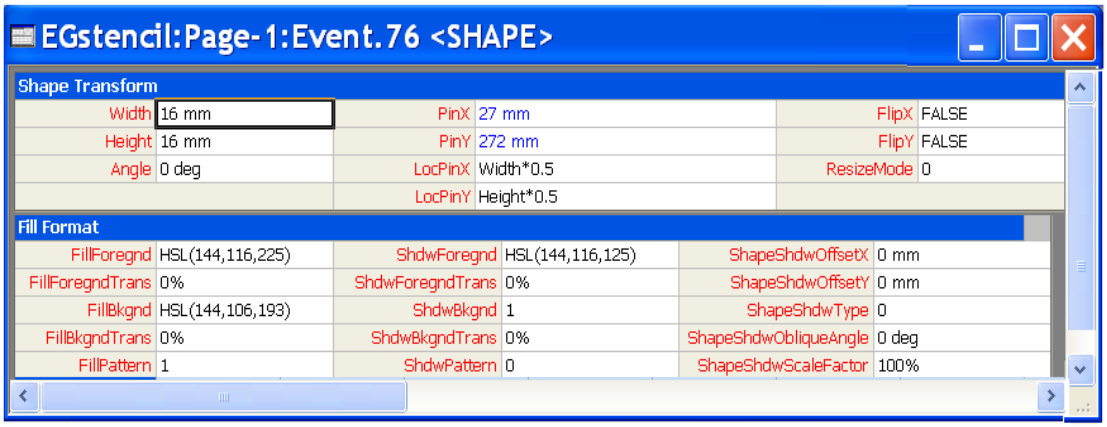

Fig. 5.14: The width and the height properties in the Shape Transform section set the diameter of the circle node and FillPatern in Fill Format section was set to a grey shade

Although the built-in properties that apply to the event and to the edges masters are different, Visio™ does not allow the structure of the shape sheet to be changed significantly. Few sections are removable and the columns per section remain the same, irrespective of the property or the shape. Thus, all the Event Graph masters inherit a large number of predefined properties that are superfluous and should be removed, were this permitted in Visio™. A few, however, can be made invisible in the shape sheet, even though they cannot actually be removed.

5.3.3. CUSTOM PROPERTIES AND METHODS OF THE MASTERS

Custom properties can be attached to a shape, alongside its built-in properties. The custom properties of the Event Graph masters are the attributes of the events and the edges which are relevant for the description of the model. For example, as a scheduling edge links the event that originates the scheduling to the event to be scheduled, data on these two events are of interest to the scheduling edge and may be constituted as custom properties.

The user is prompted for some custom properties whenever a master is dropped onto a drawing, e.g. the name of an event or the test condition of a scheduling edge, others are generated automatically from the model diagram, e.g. the origin and destination events of a cancelling edge are automatically filled according to the diagram's structure. Fig. 5.15 illustrates the set of custom properties used in DotNetSim for the event master. Number is a sequential integer automatically assigned to the event as the diagram is read; the name and the succinct description of the event are prompted to the user when the master is dropped onto the drawing page; the parameter name is also prompted to the user on dropping the master.

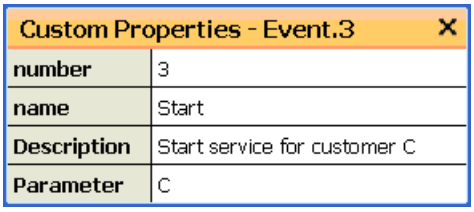

Fig. 5.15: Custom properties of the event master

DotNetSim edge masters display custom properties associated with the causal relationships between events and the time that passes through the edge. Some are common to all edge masters, others only to scheduling edges and others to parameterised edges.

The following custom properties are shared by all the edges:

- Number: The sequential integer assigned to each edge while reading the model diagram
- Origin event: The number of the event that is connected to the beginning of the edge
- Destination event: The number of the event that is connected to the end of the edge
- Condition: The condition that determines the scheduling or the cancelling of the destination event
- Priority: The priority assigned to each edge to serialise the execution of simultaneous events.

The edge number, the origin and destination events are assigned automatically when the diagram sequence is determined; the condition and the priority are prompted to the user when the edge is dropped onto the drawing page.

The delay time and its distribution are custom properties shared by the scheduling and parameterised scheduling edges:

- Delay time: The name of the variable representing the time that elapses between the occurrence of two successive events
- Delay time distribution: The delay time may be deterministic or randomly distributed. The DotNetSim prototype limits the distributions to Uniform and Negative Exponential, but others can easily be added if substituted.

Both are prompted to the user on dropping the master onto the drawing page. Fig. 5.16 shows the custom properties that are prompted to the user on dropping a scheduling master onto the drawing page.

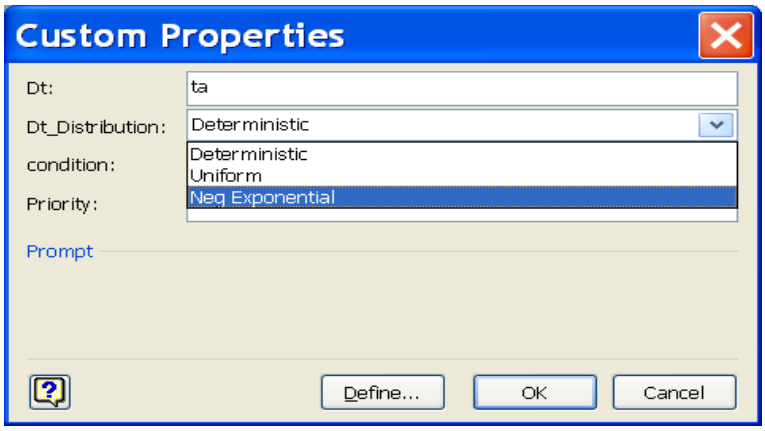

Fig. 5.16: Custom properties prompted on dropping a scheduling master onto the drawing page

Parameter and maximum parameter are custom properties shared by the parameterised scheduling and parameterised cancelling edges:

- Parameter: The name of the parameter that passes through the edge to be assigned to the formal parameter of the event to be scheduled.
- Maximum parameter: The maximum value which the parameter can take.

These are prompted to the user on dropping the parameterised edges. Fig. 5.17 lists the custom properties prompted to the user on dropping a parameterised scheduling edge.

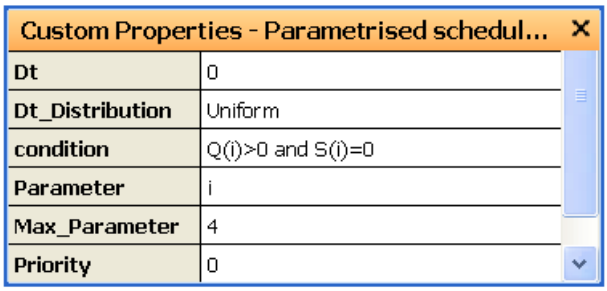

Fig. 5.17: List of the custom properties of a parameterised scheduling edge

The custom properties of the edges suggest that the corresponding masters could be implemented as a hierarchical structure of classes and subclasses. An edge class could define the common properties and pass them over, by inheritance, to two derivative classes, the scheduling and cancelling edges. These, in turn, would define more properties and pass them over to the parameterised scheduling and the

parameterised cancelling. Fig. 5.18 shows a possible tree-shaped structure of edge masters.

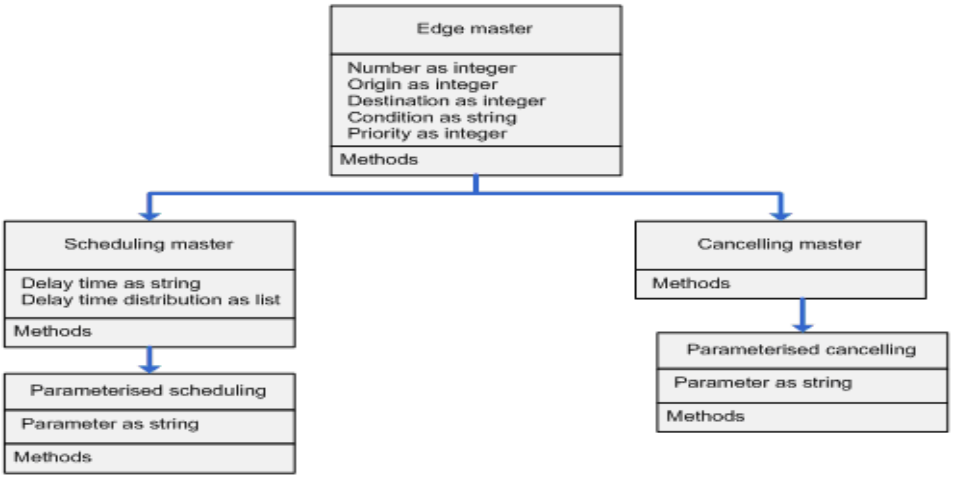

Fig. 5.18: Tree-shaped organisation of the edges masters

However, since VBA for Visio™ does not support class inheritance, the class hierarchy could not propagate beyond the inheritance of the master properties and built-in properties. This led to redundant definitions of the common properties of the Event Graph masters.

The Event Graph masters encapsulate the methods that set and get the current values of the shapes' built-in and custom properties. Also, methods were coded to add new custom properties. Event handlers, such as those that respond to the double clicking of the events or the scheduling edges, are registered in the masters but the developer of the DotNetSim modelling component must implement them in the drawing template.

5.4. COMMENTS ON THE IMPLEMENTATION OF THE STENCIL COMPONENT

The ability to specialise a stencil, by derivation of built-in stencils and from attachment of code to capture specific properties and event handlers of masters' shapes, is of great value for customising Visio™ and promoting code reusability. For

example, the masters of the newly created stencil were endowed with the attributes which fit the Event Graph modelling notation. Also, part of their behaviour was moulded by attaching to the masters' shapes the signatures of the event handlers. The Event Graph stencil is therefore used as a collection of classes that may be instantiated by any DE model.

5.4.1. THE REUSABILITY OF THE EVENT GRAPH STENCIL

Any Visio drawing file that loads the Event Graph stencil grants access to the functionality of the event and the edges masters. The masters' functionality passes to the drawing by instantiation into shapes. Thus, dragging an event master, for example, and dropping it onto an Event Graph corresponds to the creation of an event which inherits all the properties and methods coded in the master. The custom properties are prompted to the user, the corresponding values are stored in the shape sheet and the event handlers respond to actions such as double clicking the shape.

Hence, the functionality of the Event Graph stencil is black-box reusable by all the drawings that load the stencil. They acquire functionality when the stencil is loaded and, by instantiating the masters and providing the input parameters (values of the prompted properties), they acquire shapes with specific functions. On the other hand, the functionality of the stencil is white-box customisable, i.e. new masters can be added and the existing masters can be altered or removed. New properties and new methods may also be added to the existing masters. Altering the stencil functionality has no impact on diagrams previously drawn, since a document stencil is kept for each drawing. The document stencil stores a copy of each master when it is dropped onto the drawing, which means that each drawing deals with a particular instance of the stencil. This stresses the drawing's portability and extends the level of customisation that can be achieved per DE model.

5.5 THE CHAPTER IN CONTEXT

This chapter introduces the graphical modelling environment of the DotNetSim prototype, setting out its objectives and explaining the Event Graphs paradigm which was chosen, as an example, to provide its conceptual framework. The Event Graph's extended modelling notation led to the Event Graph stencil which was implemented by the DotNetSim stencil component. The DotNetSim stencil component is an extension of Visio™ which creates a stencil from abstract shapes specifically defined to represent the events and the edges of that diagrammatical notation. It also implements the main operations on the stencil file.

The next chapter discusses the modelling component of the DotNetSim prototype which uses the Event Graph stencil to implement the functionality required to capture the logic of DE models diagrammatically as Event Graphs.