

8. THE DOTNETSIM OUTPUT ANALYSIS

Chapter Overview	170
8.1. Objectives of the DotNetSim output analysis	172
8.2. The DotNetSim: Output analysis component	173
8.2.1. Customising Excel TM	174
8.2.2. Analysing the system's changes of state	177
8.2.2.1. Tracing state changes	177
8.2.2.2. Plotting system's states	178
8.2.2.3. Statistical add-ins	179
8.2.3. Reporting the simulation analysis	180
8.3. Comments on the implementation of the DotNetSim output analysis	181
8.4. The chapter in context	183

CHAPTER OVERVIEW

DotNetSim's output analysis prototypes an Excel-based environment for analysing and reporting the results of running multiple replications of a DE model over time. It customises Excel's built-in data analysis and reporting tools and allows for the possibilities of further specialisation to serve the purposes of additional requirements for an event-based simulation of a DE model. Worksheets are appropriately formatted to receive the results produced by the DotNetSim simulation engine and some built-in Excel tools are customised as examples to serve simulation output analysis. The DotNetSim output analysis mainly stresses that the tools demanded by many

simulations are likely to be among the huge number of data analysis Excel add-ins which are currently available. These can be temporarily plugged to the output analysis component if and when needed.

Thus, the DotNetSim output analysis component consists of a number of VBA components, gathered in a template, to format a set of worksheets and to customise and specialise Excel data analysis and reporting tools in order to suit the needs of the event-based simulation of DE models. Also, it automates user-commands for plugging and unplugging new and existing add-ins. The simulation tools are displayed on a menu of commands, the Simulation Output Analysis menu, which is automatically generated and appended to the Excel menu bar on instantiating this Excel template.

Like the other two coarse-grained components of the DotNetSim prototype, the development of this component stresses its integration with other applications and the customisation of the generic application within which it was developed. Upstream, the output analysis component integrates with the simulation engine that instantiates this template and manipulates the Excel object model to place the simulation results. Downstream, it integrates with other Microsoft applications, by object-oriented manipulation of their object models, to report the analysis of the simulation results. It also integrates with a wide range of prefabricated analytical and graphical tools constituted as add-ins and pluggable to ExcelTM when needed.

This chapter describes the functionality built on top of ExcelTM to illustrate the use of generic software capabilities to derive a simulation's layer of software tools which integrates with other modelling and simulation components of the DotNetSim prototype.

Finally, some comments are made on the implementation and extension of these Excel-based simulation output analysis software tools. These comments lead to further

discussion on the value of the integration of different Microsoft applications for modelling DE systems.

8.1. OBJECTIVES OF THE DOTNETSIM OUTPUT ANALYSIS

The DotNetSim's output analysis prototypes an Excel-based data analysis and reporting tool for DE systems. It illustrates the customisation of the Excel's built-in data analysis and reporting capabilities to trace, analyse and report the state changes that a DE system passes through during the simulated runs. From upstream, it integrates with the DotNetSim simulation engine, from which it receives the simulation results and, from downstream, it integrates with other applications to report the analysis of the outputs.

A number of VBA components are gathered in an Excel template to format the worksheets of a workbook and to offer a menu-driven set of tools which exemplifies the customisation and the extension of ExcelTM to serve a discrete event simulation. Specific VBA components and pluggable add-ins are developed to derive simulation tools from the generic analytical and graphical ExcelTM capabilities and to facilitate the exchange of data between the ExcelTM and other applications.

Fig 8.1 depicts the DotNetSim prototype's architecture, zooming in the output analysis component. The DotNetSim simulation engine instantiates the output analysis template and places the results of the simulation runs in the new workbook. That is, it saves the stage which the system reaches at the end of each replication along with the corresponding sequence of the time-stamped events which were executed. In this workbook it also places the state transition matrix, i.e. the changes triggered on each state variable by each event. This data is then available for analysis and reporting. Beyond the ExcelTM built-in generic and data analysis tools, DotNetSim output

analysis offers, as examples, specific simulation tools, namely those for customising Excel™ so that the user may trace analytically and graphically the state variables for each simulation run and may report tables and charts via Word documents and PowerPoint presentations.

On exiting the Excel workbook, the execution returns to the simulation engine and finally back to the Visio-based modelling environment.

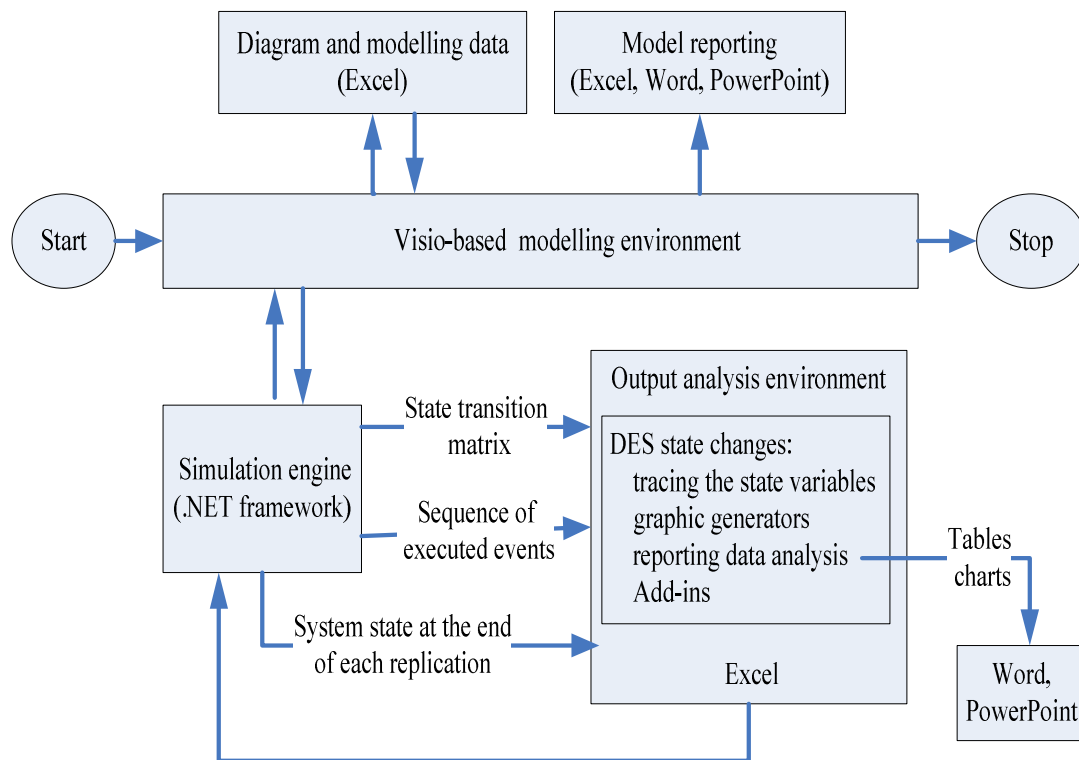


Fig. 8.1: Zooming the output analysis component in the DotNetSim’s architecture. The execution of the DotNetSim starts and ends within the Visio-based modelling environment, passing through the stages ‘Modelling – Simulation – Output analysis’

8.2. THE DOTNETSIM: OUTPUT ANALYSIS COMPONENT

The DotNetSim output analysis environment is implemented in an Excel template, the SimOutAna.xlt, which contains a number of VBA modules to format the worksheets and offer simulation tools for the following categories of functionality:

- Customising Excel™ to display only the functionality required for analysing

and reporting the simulation results

- Analysing the state changes of the DE system analytically and graphically, including tracing the chronological sequence of executed events in each simulation run
- Reporting the simulation analysis to Word documents and PowerPoint sets of slides.

These categories are displayed in the Simulation Output Analysis menu shown in Fig. 8.2. This menu is generated and appended to the Excel's menu bar at runtime.

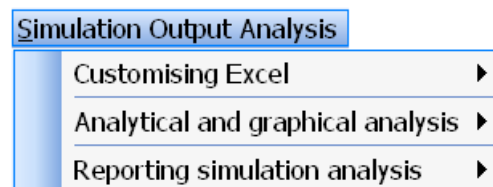


Fig. 8.2: Simulation Output Analysis menu

8.2.1. CUSTOMISING EXCEL™

On instantiating the SimOutAna template, a workbook is automatically created with six worksheets named Events, Replications, Transition, Analytical, Graphical, and Customisation. The first three sheets are used by the simulation engine to place the simulation results:

- The Replications worksheet records the state of the DE system at the end of each replication. Each record is a set of fields that identify the replication, specify its duration in terms of system time and simulation time and store the values of the state variables at the end of the replication
- The Events worksheet records, as the simulation replications run, the executed events and their corresponding execution times. Each executed

event is recorded in two fields - the simulation time and the event number - hence, the events executed during each replication are listed in two adjacent columns of the Events Worksheet.

- The Transition worksheet is used to place the event's state transition, i.e. the state changes triggered by the events as described within the DotNetSim's modelling environment.

The last three worksheets are used to output the results of the analytical and graphical analysis and to customise Excel™ to serve the purposes of the simulation output analysis component.

Also, and like the customisation of Visio™ described in chapter 6, to increase the simplicity of the graphical user interface, the generic tools of Excel™ can be limited to those needed by the simulation output analysis. Thus, the SimOutAna template starts by automatically unloading or hiding the menus, toolbars and add-ins that might not be necessary for the analysis and reporting of the simulation results. However, at any time, these tools can be activated as shown in Fig.8.3.

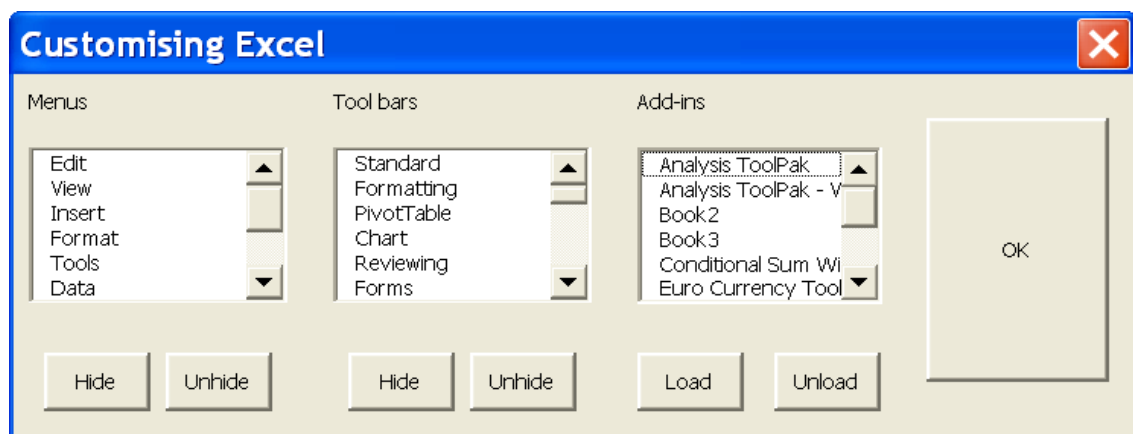


Fig. 8.3: The boxes list the menus, tool bars and add-ins available at the current Excel™ installation

Also, the Simulation Output Analysis menu that is generated and appended to the Excel menu bar on instantiating the SimOutAna template is customisable in order to

display only the functionality required by the current DE simulation. Thus, the Simulation Output Analysis menu can be generated from the list of commands stored in the Customisation worksheet as shown in Fig. 8.4.

	A	B	C
1			
2	Level	User command	Procedure
3	1	Customising Excel	
4	2	Menus, toolbars & Add-ins	CustExcel
5	2	Simulation output analysis menu	CustSim
6	1	Analytical and graphical analysis	
7	2	Tracing state changes	Extracing
8	2	Plotting system's states	
9	3	State variables charts	Gstatevar
10	3	Other charts	gothers
11	2	Statistical add-ins	Staddin
12	1	Reporting simulation analysis	
13	2	Tables	
14	3	To Word	TabToWord
15	3	To PowerPoint	TabToPPT
16	2	Graphs	Reptoolbar

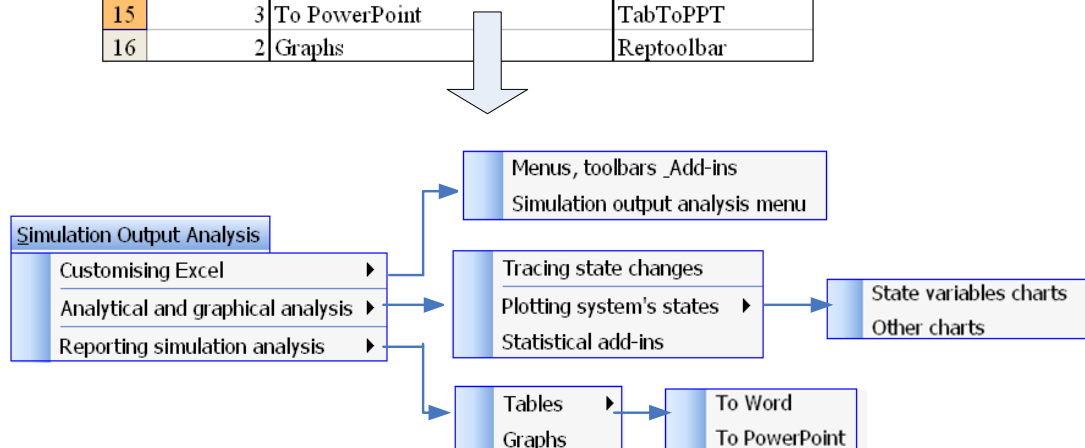


Fig. 8.4: The list of commands typed in the Customisation worksheet is converted into the Simulation Output Analysis menu

While generating the menu, the list of procedures stored in the Customisation worksheet is checked against the VBA procedures of SimOutAna.xlt. Missing procedures are signalled in the cell to the right of their names and replaced by a general Missing procedure which serves to remind the user that the functionalities are not available. Thus, the user may adapt the menu to match the needs of each simulation by adding or removing commands and the corresponding procedures.

The Customising Simulation Menu toolbar, shown in Fig 8.5, is activated

automatically from the Customising Excel command of the Simulation Output Analysis menu. It allows a new menu to be generated or the original one to be restored.

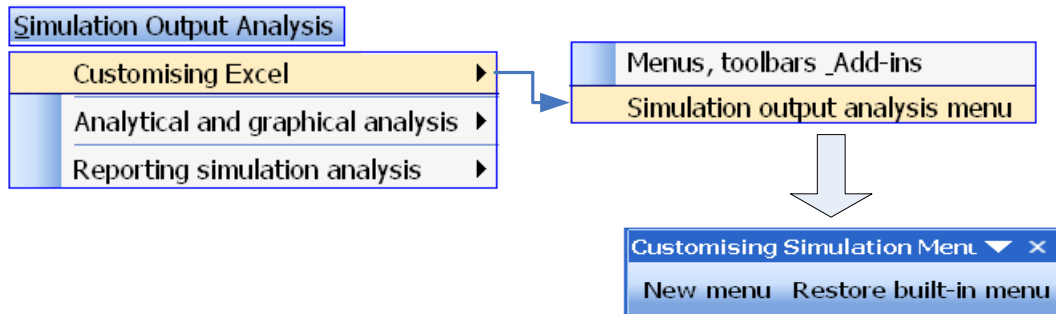


Fig. 8.5: The Customising Simulation Menu toolbar allows the customisation of the Simulation Output Analysis menu

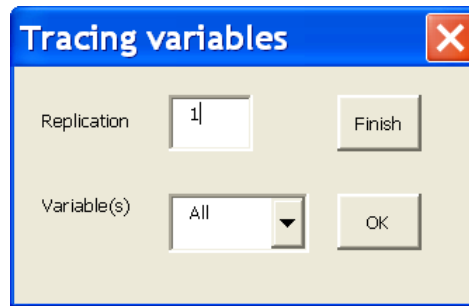
8.2.2. ANALYSING THE SYSTEM'S CHANGES OF STATE

The simulation results written by the DotNetSim simulation engine into a workbook instantiated from the SimOutAna template (see section 7.2.3.) may now be analysed by resorting to the analytical and the graphical ExcelTM tools. This prototype component stresses that specific data analysis tools can be constituted as add-ins and plugged to the SimOutAna-based workbook when and if they are needed. Some VBA modules were developed to extend ExcelTM with examples of tools for event-based simulation.

8.2.2.1. TRACING STATE CHANGES

Based on the list of executed events output by the simulation engine while running a replication and the matrix of the event's state transition, state variables can be traced to observe their performance during a simulation run. Each executed event and chosen state variables are looked up in the state transition's matrix and updated accordingly.

If, as shown in Fig. 8.6, all state variables are selected to be traced, a list of the state changes which the DE system passed through during a replication is displayed.



The dialog box titled "Tracing variables" has a blue header with a close button (X). It contains two rows of controls. The first row has a label "Replication" followed by a text input field containing "1" and a "Finish" button. The second row has a label "Variable(s)" followed by a dropdown menu showing "All" and an "OK" button.

Fig. 8.6: Input form to list the state changes which the DE system passed through during the first replication

The final values assigned to the state variables can then be cross-checked with the state the DES system reached at the ending of the corresponding replication output by the simulation engine to the Replications worksheet.

8.2.2.2. PLOTTING SYSTEM'S STATES

The state changes which the DES system passes through during each replication and the corresponding final states may be graphically analysed in order to evaluate the system performance and improve the knowledge of the system.

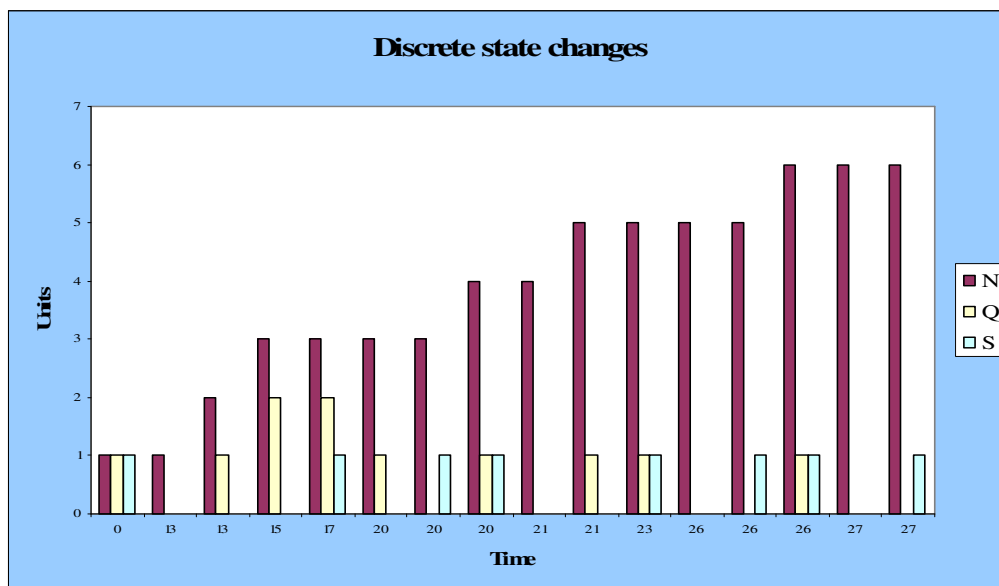


Fig. 8.7: DE system's states at certain points of time

Charts of the state variables may be generated automatically by looking up the ranges that contain the traced variables in the Transition worksheet. The charts may be placed in a separate sheet, the Graphical worksheet, in order to facilitate the

comparison of several charts. Fig. 8.7 depicts graphically the state transition of a single server queue system, which was modelled within the DotNetSim modelling environment and then simulated by the DotNetSim simulation engine. The successive values of the state variables N, Q and S (respectively the number of customers that arrive at the system, the number of customers queuing, and the status of the server {busy=0, idle=1}) were traced for a simulation run and placed in the Transition worksheet and then automatically plotted in a bar chart.

This example could be improved by users who resort to the built-in Charts functionality of Excel™. Other VBA programs can be written to draw additional charts on the DES system's state changes.

8.2.2.3. STATISTICAL ADD-INS

There are many Excel-based tools available to support statistical analysis, hence, the DotNetSim prototype in this concern is limited to load and unload appropriate add-ins and to automatically generate charts of the state variables previously traced.

The Excel-pluggable add-ins can be chosen from the list box displayed in Fig. 8.8 and loaded or unloaded for each simulation solution. New add-ins, which are not listed, have to be identified by their full names in order to be loaded.

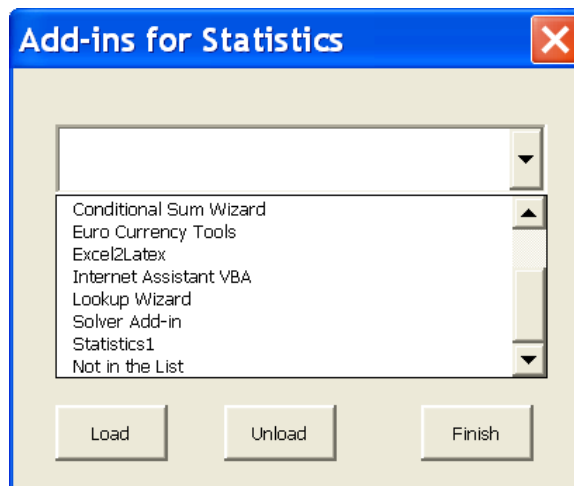


Fig. 8.8: Loading and unloading add-ins to analyse the simulation results

8.2.3. REPORTING THE SIMULATION ANALYSIS

The simulation results and the subsequent data analysis may be inserted in textual reports and slide presentations by manipulating the objects of other Microsoft applications from within Excel™. The SimOutAna template contains VBA procedures that automatically insert Excel tables and charts in Word documents and PowerPoint presentations.

- Copying Excel tables to Word documents and PowerPoint presentations

Ranges of Excel worksheets are copied and pasted either into new or existing Word files and PowerPoint files. The user's selection of ranges to be copied is input to the form shown in Fig. 8.9.

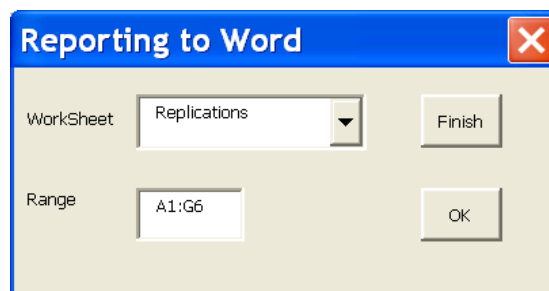


Fig. 8.9: User interface to input the range to be copied into a Word document

The tables which the user selects are placed in subsequent paragraphs or slides.

- Copying Excel charts to Word documents and PowerPoint presentations

The Reporting Graphs toolbar is shown in Fig. 8.10. It is activated automatically from the Reporting simulation analysis - Charts sequence of commands of the Simulation Output Analysis menu. It allows the selected chart to be copied and pasted to Word documents. Similarly (with the appropriate VBA statements), the charts can be pasted to PowerPoint presentations.

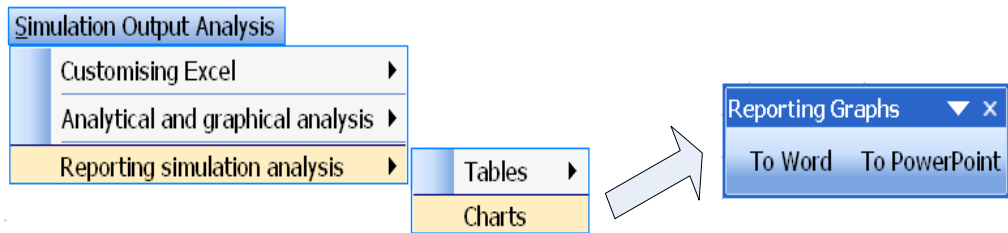


Fig. 8.10: The Reporting Graphs toolbar allows charts to be pasted to Word documents and PowerPoint presentations

Whilst reporting tables and charts, Excel™ instantiates Word™ or PowerPoint™ applications or gets references to active applications (see Fig. 8.11 and Fig. 8.12). Methods are invoked on these objects to instantiate new documents or get references to the active documents.

```
Set wd = GetObject("Word.application")
Set doc = wd.ActiveDocument
```

Fig. 8.11: Statements of VBA for Excel™ to get a reference to the active Word document

```
Set ppapp = CreateObject("PowerPoint.application")
ppapp.Activate
Set np = ppapp.Presentations.Add
```

Fig. 8.12: Statements of VBA for Excel™ to instantiate the PowerPoint application and open a new presentation

Finally the selected tables or charts are pasted into them. This applies to other Microsoft applications.

8.3. COMMENTS ON THE IMPLEMENTATION OF THE DOTNETSIM OUTPUT ANALYSIS

The implementation of the output analysis component of the DotNetSim prototype shows that Microsoft Excel™ can be integrated with other applications to analyse and report the simulation results. The integration is based on the object-oriented paradigm

and, hence, ExcelTM is instantiated by other applications and its objects are manipulated by components developed within different packages. Also, ExcelTM instantiates other applications and manipulates their objects invoking the methods they expose. Thus, the output analysis Excel template is instantiated from within the DotNetSim simulation engine in order to create and manipulate a new workbook in which worksheets' ranges it places the simulation results. The DotNetSim simulation engine also instantiates and manipulates Microsoft WordTM and Microsoft PowerPointTM to insert the tables and charts created for analysing the simulation results. Similarly, other applications, which support Microsoft Automation, can be integrated by applying the underlying object-oriented programming paradigm.

This integration extends to the huge number of available Excel-based add-ins which provide modules for data analysis. Additional add-ins can be written specifically for DotNetSim output analysis. VBA components can then be anchored in ExcelTM Workbooks, constituted as add-ins and plugged to the output analysis component as needed.

This object-oriented integration shows that ExcelTM can be customised for a DES software solution and it also demonstrates the DotNetSim idea of specialising and integrating widely-used packages into simulation packages.

However, the object-oriented limitations of the VBA programming language and the incoherence of the different varieties of VBA referred to in chapter 6 are, as in the DotNetSim modelling environment, an obstacle to the implementation of the output analysis component. All that was said about VBA for VisioTM applies to VBA for ExcelTM. It allows the integration of different packages, but its procedural nature with shallow incursions into OOP imposes huge limitations on the creation and manipulation of user types which could simplify the development of simulation

specific data analysis tools. Also, the syntax incoherence of VBA is time consuming, mainly while invoking identical methods on objects of different applications.

Nonetheless, programming in VBA for Excel™ requires less effort than in VBA for Visio™ above all because of the huge number of users and developers who have been writing and documenting well programs and add-ins for Excel™ since the first versions.

8.4 THE CHAPTER IN CONTEXT

This chapter describes and comments on the implementation of the DotNetSim output analysis environment, built on top of Excel™ to illustrate the use of generic software to derive simulation-specific data analysis and reporting tools. These are integrated with the DotNetSim simulation engine component following OO principles. The results of the simulations are placed on an instance of the output analysis template which is created and manipulated from within the DotNetSim simulation engine. The data is then analysed within the DotNetSim output analysis environment and reported to other Microsoft applications which are also OO manipulated from within this DotNetSim component.

The next chapter demonstrates the use of the DotNetSim prototype.