# 10. CONCLUSIONS AND FURTHER RESEARCH



## CHAPTER OVERVIEW

This thesis explores a component-based development strategy that aims to pull the DE modelling and simulation software further towards object-oriented components that cross programming languages and packages, linking them in a single application. This is done by developing the DotNetSim project which prototypes DE simulation software across the entire requirements of a simulation application package.

This chapter summarises this thesis by reviewing the arguments, the underlying research and its major contributions to an alternative development strategy for DES

software. It presents some lessons for the development of commercial DES software, inferred from the strengths and weaknesses that the implementation of the DotNetSim prototype has revealed.

Some issues for future research are proposed. This includes research on frameworks for dependencies, assumptions and standards within which libraries of simulation components can be developed and deployed as web services. Finally, it raises and leaves to future discussion the question on what value the new generation of DE simulation software may have to narrow the gap between the theoretical acceptance of simulation and its practical use as a tool for exploring complex systems.

## 10.1. SUMMARY OF THE THESIS

Set against the expansion of the mass customisation and the corresponding on-demand paradigm to all sectors of the economy, this thesis explores an alternative approach for the development of the DES software which may facilitate the customisation of simulation solutions.

The on-demand consumption behaviour induced by mass customisation is also extending to software products, namely to simulation software. In a similar way to other consumers, simulation software users – developers, solution builders and end-users – are increasingly demanding software products which are customisable to their specific needs. It seems that packages that fit all problems at the expense of large amounts of learning and use of resources are losing the interest of the users in favour of software solutions.

Currently, simulation software users take for granted that software fulfils its designated purposes, i.e. it is dependable, portable, friendly, recoverable, extensible, etc. As with other products, it is plausible that they expect, to a certain extent, to

select the functionality the software provides to match the specific needs of the current problem. Their expectations and desires may even look forward to instant customisation of software solutions.

On-demand simulation, i.e. the instant selection, customisation and assembly of only the functionality each simulation model requires, is not yet attainable. The technological issues that the on-demand software raises are under ongoing investigation in computing science, aiming to enable effective and efficient selection, negotiation, delivery and binding of functionality at runtime. Meanwhile, simulation software can benefit from the recent computing advances in developing paradigms and integration frameworks to progress towards on-demand assembly of simulation solutions in an open market of components, possibly based on the Internet.

Conventional developers of simulation software have responded to the demand for customised solutions by adding new features and tools to their existing simulation packages. This has led to huge, monolithic applications with functionalities that are constantly extended in a 'generalising-customising-generalising' development cycle. Existing packages allow some degree of customisation but, in general, they were not designed to be customisable. Hence, if customisation is possible, it is difficult.

Nevertheless, there are already a few exceptions that attempt to break this mould by integrating simulation and generic software tools to facilitate customisation. This highlights an emerging understanding that the mode of development for much of the contemporary simulation software may not be well-suited to meet the demand for extensible simulation solutions.

### 10.1.1. MAJOR ARGUMENTS IN THE THESIS

This thesis investigates the composition of DES software from components developed within different applications in order to lead the way eventually to the ideal on-

demand simulation. Thus, DES software solutions may be composed from components that the users select, customise and assemble in order to suit each simulation model. In addition, simulation components may be developed within widely-used packages on top of their generic built-in features in a progressive object-oriented specialisation of subcomponents and their component and layered-oriented composition. Customisation is then eased by the use of familiar developments tools to derive self-contained and self-healing components that integrate vertically in a 'generalise-specialise' development strategy.

Object, component and layered-oriented paradigms and integration mechanisms are therefore crucial for this development strategy. Amid recent software integration mechanisms is the Microsoft .NET Framework. It is a fully object-oriented development platform which provides the wiring mechanisms to enable software artefacts to integrate across different environments.

This thesis develops the DotNetSim project to explore how far the integration of the .NET Framework with Microsoft Office applications can pull the DE modelling and simulation software towards fully object-oriented components that cross programming languages and packages, linking in a single application. A DES software is prototyped across the entire requirements of a simulation application package including graphical modelling environments, simulation executives and output analyses.

The DotNetSim prototype focuses on two main development issues:

(i) Data exchange between distinct software development environments through the instantiation of objects to apply the object-oriented programming paradigm and replace the current creation of intermediate files and associated format conversions.

(ii) The integration of powerful simulation engines with widely-used packages in an architecture that supports the straightforward modification of modelling and output analyses environments.

### 10.1.2. RESEARCH THAT HAS LED TO THE THESIS

Bearing in mind the contextual environment and the major arguments, the research which has led to this thesis has proceeded as follows:

- Reviewing and defining relevant concepts of modelling, simulation and computing fields to clarify the relationship between these three disciplines, to understand the current trends of the DE simulation software and devise its future developments. The co-evolution of modelling, simulation and computing is reflected in the simulation software which keeps the pace with the advances of these three disciplines.
- Reasoning on the development strategy of much contemporary simulation software, which aims to ensure that the simulation packages have sufficient functionality to suit a large number of problems. DES packages are consequently bloated with constant additions of wizards, templates and add-ons in a 'generalising-customising-generalising' development cycle. This strategy has been successful until now, but the resulting huge monolithic simulation packages are increasingly difficult to customise.
- Discussing the current experiences of a small number of simulation software developers who have recently undertaken steps towards new development approaches. This suggests that contemporary development strategy may not be well suited to meet the demand for customised simulation solutions. The Promodel Process Simulator™, Micro Saint Sharp™ and the HighMast™ Modeling and Simulation Toolkit are examples of those applications that

have resorted to the integration of simulation and generic software tools to facilitate customisation.

- Devising an alternative vision for simulation software that centres the development strategy in a series of 'generalise-specialise' cycles. DE Simulation software should ideally be vertically architectured by successive derivation of specific frameworks and libraries of simulation.
- Defining the DotNetSim project which addresses Discrete Event Modelling, Integration Technologies and Discrete Simulation and their interrelationships to investigate how far the Microsoft integration philosophy can propel DES software towards the ideal scenario. This materialises in the development of the DotNetSim prototype as an example of what Microsoft integration philosophy can do for DES software.
- Developing the DotNetSim prototype for discrete event modelling and simulation, linking in a single application, components developed within various widely-used Microsoft applications in different programming languages. Three coarse-grained components, which implement a modelling environment, a simulation engine and an output analysis environment, were composed into the DotNetSim prototype (see Fig. 10.1).

  The modelling environment emulates Schruben's Event Graph methodology for simulation modelling. It consists of two Visio-based components:

  (i) The stencil component implements the Event Graph's extended modelling notation. It defines the properties and the behaviour of abstract shapes representing the events and the edges of that diagrammatical notation.

  (ii) The modelling component implements the functionality required to

capture the logic and the dynamics of DES systems modelled as Event Graphs.

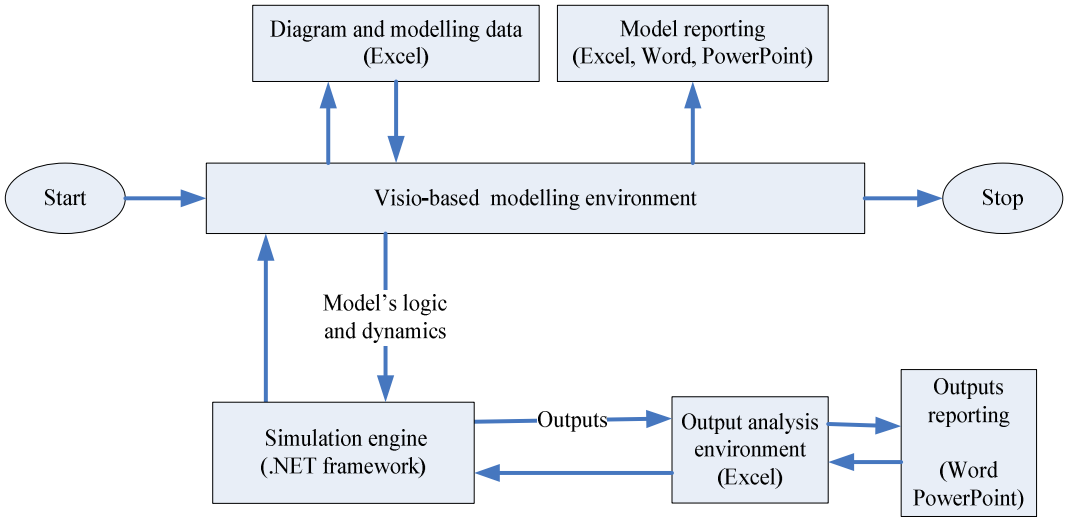


*Fig. 10.1: DotNetSim's composition of components. The execution of the DotNetSim starts and ends within the Visio-based modelling environment, passing by the stages 'Modelling – Simulation – Output analysis'*

The simulation engine prototypes an event-based simulation executive in a number of .NET-based components. It applies the OOP principles to read the DE model's logic and dynamics from the DotNetSim modelling environment, runs the event-based simulation and to write the simulation results into the Excel-based DotNetSim output analysis environment.

The output analysis environment prototypes a data analysis and reporting tool for DES systems. It illustrates the specialisation of the Excel™ generic capabilities for data analysis and reporting to trace, analyse and report the states that a DES system passes through during the simulated runs.

Next, this research reflects on the DotNetSim project, the lessons that were learned from its development and its contributions to an alternative development strategy leading to the next generation of DES software. Finally, we present some issues that might be of interest to other researchers who may wish to move simulation software further towards the ideal vision devised in this thesis.

### 10.1.3. CONTRIBUTION TO THE DEVELOPMENT OF DES SOFTWARE

Through its working prototype, the DotNetSim project shows that DES software may be composed by object-oriented integration of components written in different packages and programming languages. Integration frameworks such as Microsoft .NET Framework encapsulate the software infrastructure that allows different packages to be considered as objects of other packages. Hence, one application is instantiable from within another, to which the former exposes its object models. The Microsoft .NET Framework allows this interoperability also across different technological generations, e.g. COM-based and .NET-based components.

The DotNetSim prototype illustrates the development of a component-based simulation software application that integrates .NET components with COM-based components developed within widely-used Microsoft applications. Its implementation of an object-oriented component architecture built on top of generic software applications highlights the following major contributions to the customisation of DES software:

- Replacement of components

  Each of the three coarse-grained components that the DotNetSim integrates can be substituted by others which implement different modelling paradigms, simulation worldviews and data analysis and reporting techniques. Thus, the Visio-based modelling environment that emulates the Event Graph paradigm can be replaced by other Visio-based graphic modelling environments. Various Visio stencils can be added to support different modelling notations.

  The .NET-based simulation engine, which reads the simulation model by invoking the appropriate methods on the Visio objects, can implement any

simulation worldview. The Excel-based output analysis environment can be replaced by other data analysis components and its functionality can be extended by plugging prefabricated add-ins to the existent component.

- Addition and removal of components

  Likewise, other components can integrate the DotNetSim prototype to provide specific functionality. For example, .NET or COM-based graphic checkers, optimisers or animators can be instantiated by the DotNetSim components which then may manipulate the respective functionality as if it were their own. Components may also be removed without affecting the remainder set of components and the corresponding inter-operation.

- Components sourced in widely-used generic software

  The DotNetSim prototype shows the integration of components written within widely-used generic Microsoft applications. COM-based components written within Microsoft Office integrate with .NET-based components written in fully object-oriented programming languages.

  Thus, the large number of existing generic and simulation specific components that were developed for Microsoft applications can object-oriented integrate as black-boxes with the DotNetSim components. On the other hand, most of these components are open source written within quasi-standard development environments (Microsoft's graphical user interfaces, development tools and popular programming languages), hence white-box modifiable to suit specific needs. Also, the development of simulation components which extend these familiar applications does not require 'rare and expensive' programming expertise.

At the core of DotNetSim's research contribution is the reusability of

prefabricated functionalities, reinforced by the development of simulation components on top of generic widely-used applications. This stimulates the supply of components which target a wider market and a variety of modes of usage that range from black-box to white-box reusability.

The DotNetSim prototype demonstrates that simulation software, like other software domains, can keep up with computing advances and evolve from a monolithic developing strategy, which is mainly procedural, towards a component-based strategy which is framed by object programming paradigms. Object-oriented simulation components and integration frameworks support the composition of simulation solutions from prefabricated components. Components can be added, replaced and removed from a core application in order to compose particular simulation software solutions. This places simulation software within the mass customisation economic context.

## 10.2. LESSONS LEARNED

The DotNetSim project and its working prototype illustrate how the integration-friendly development frameworks, such as the Microsoft .NET Framework, can be used to develop a component-based DES software. The full object-orientation of these integration mechanisms eases the integration of components written within different software applications and programming languages by handling components as objects regardless of their development environments. Thus, .NET components written in different programming languages can instantiate Microsoft Office-based components (COM components) and handle their object models by invoking the appropriate methods. The integration operates at a higher level than machine level, hiding from application developers and solution builders the infrastructure on top of which

components sourced within different packages may handle others' objects as if they were their own. This was experimented with the DotNetSim prototype by integrating components sourced in widely-used applications with a .NET-based simulation engine.

### 10.2.1. STRENGTHS AND WEAKNESSES REVEALED BY THE DOTNETSIM PROTOTYPE

The strengths of the DotNetSim DES software prototype derive from the object-oriented composition of three coarse-grained components developed within different programming environments. These components integrate by application of the OOP principles, which means that data flows among the components not by file transfer and conversion of formats but by proper instantiation of the object models of each component and invocation of the corresponding methods. The Microsoft Office applications are objects to other applications. Hence, the functionality of each application extends by setting references to other applications which allow data to flow as if there were no borders between applications.

The DotNetSim prototype highlights the contribution of this object-oriented integration of components to the development of component-based DES software. It also shows how Microsoft Office applications can be extended to provide DE modelling and simulation functionality from derivation of their built-in generic features. DES software is then component compose-able and based on quasi-standard Microsoft GUIs which increase its ease of learning, ease of use and ease of customising.

However, the implementation of the DotNetSim prototype has also uncovered the weaknesses and difficulties which derive from manipulating objects using Visual Basic for Applications, a procedural programming language with some object-oriented primitives. Also, the integration of the powerful .NET languages with VBA has

emphasised the syntactic and semantic differences which required the development of wrappers to homogenise their use within a single component.

### 10.2.2. OPPORTUNITIES AND THREATS TOWARDS A COMMERCIAL VERSION

The DotNetSim prototype draws interest to the development libraries of modelling and simulation components within generic software environments, in a variety of programming languages and compliant with the 'generalise-specialise' development paradigm. Software solutions can then be assembled by selecting the libraries and components that better suit the specific DE simulations.

The DotNetSim prototype highlights the opportunity for developing generic and simulation components on top of widely-used applications that object-oriented integrate with other components regardless of the programming language and development application. Specially developed components extend the built-in capabilities of the applications and they may be written from outside in more powerful programming languages. Components, for example, which extend the modelling capabilities of Visio™ may be written within the Microsoft .NET Framework in a combination of .NET languages. It also highlights the opportunity of composing different combinations of components, which increases the flexibility of the simulation software applications. A modelling component that emulates Petri nets or activity cycle diagrams, for example, can be composed with the C#-based DotNetSim simulation engine. Also, a component that traverses an Access database can be composed with the Event Graph modelling component of DotNetSim. Components are to be reused within different compositions by different categories of simulation users. This calls for distinct level of usage, hence, distinct levels of exposition of the components' cohesion, coupling and context dependencies.

In short, it would be worth considering the disintegration of three coarse-grained

components of the DotNetSim prototype into a number of fine-grained components in order to increase the flexibility of the composition of solutions. The DotNetSim modelling component, for example, should be de-composed in finer-grained functionality, so as to allow a solution builder to add a diagram checker or to allow end-users to alter the toolbars, menus or the diagram notation. The composition of a DES software solution should start with the GUI-based selection of components and the transference of the respective source code in order to facilitate its customisation.

In addition, to facilitate the integration of COM components (e.g. Microsoft Office-based components) with .NET components, a layer of wrappers may be required to bridge the gap between the .NET programming languages, which are fully object-oriented, and the Visual Basic for Applications, which is a procedural language that implements powerful object models. Also, the PIAS (Primary Interop Assemblies) that enable this integration may have to give way to DotNetSim Interop Assemblies, through which a wider range of objects and methods of the Microsoft Office applications are exposed to .NET components.

The threats to the evolution of the DotNetSim prototype towards a commercial application for DES software are generally associated with the usage of computer resources and the performance indicators.

## 10.3. FUTURE RESEARCH

The achievements of the DotNetSim prototype place the DotNetSim project into the broad area of the component and layered-based composition of software. The recent technological advances in integration and interoperability of software have expanded this often-revisited research area to the composition of application software from prefabricated components regardless of the development and deployment contexts.

The 'classic' issues raised by the component-based paradigms which are related to the topography of architectural composition, the interfaces exposed by the components and the standards that apply to the development of components, interfaces and their composition, have gained new dimensions which derive from the spread of components over the Internet. In addition, issues associated with the granularity, the cohesion and coupling of components and their binding time are also to be revisited due to the transposition of the OOP principles to the derivation of sub-components and their constitution into layered-libraries of functionality.

In all, the achievements of the DotNetSim project are to be continued in setting the frameworks for the constitution of libraries of components for DES software that may be delivered as web services.

### 10.3.1. LIBRARIES OF COMPONENTS FOR DES SOFTWARE

In order to better explore the customisation of the DES software, the extensive functionality provided by the three DotNetSim coarse-grained components have to disintegrate into finer-grained components and grouped into layers of functionality. Then these layers of functionality can be constituted as libraries of components that are plug-able to simulation solutions by setting a reference to the respective containers. Libraries and components may be extended, shrunk or may constitute the base for further specialisations. Hence, software solutions can be incrementally built by selecting and customising the appropriate components. However, this calls for frameworks of assumptions, rules, specifications and standards which have to be investigated so as to convey the functionality of each component, its cohesion, coupling and context dependencies. Frameworks for the development and deployment of layers of functionality that bridge different context environments have also to be investigated. Moreover, it is important to investigate how these frameworks can lead

to metadata that, by reflection, allows the transference of the selected the components, their negotiation, delivery and binding at runtime. This will lead to on demand simulation software.

Other technical issues arise from the logic and computational complexity associated with the granularity of the components and the optimisation of their composition. Finer granularity may increase reusability but it also may diminish the efficiency and the robustness of the composition of the components [118]

### 10.3.2. WEB SERVICES

The libraries of components for DES software may go further into customisation by delivering the functionality across the Internet. Components may be constituted as web services [45] and, thus, expose their functionality across the Internet. Software components are then instantiable and their methods are invoke-able from within DES applications through web browsers.

Components that range from add-ins to Microsoft applications to modelling environments and simulation engines can be constituted into web services and invoked remotely through the Internet. A simulation engine, for example, can be invoked remotely through its URL (Uniform Resource Locator), run the model which is read by parameters passing and place the simulation results where (application or web service) the DES software solution specifies. DES software will eventually be composed of a set of web services [50, 113].

Microsoft .NET Framework implements ORPC (object-oriented remote procedure calls) which hides the transport protocols and the specific formats from the software developer or builder, enabling the integration of web services by references setting (URLs ) and methods' invocation. This asks for simulation software to follow the research computing advances on the web services description languages (e.g. WSDL),

discovery of web Services (e.g. UDDI and DISCO) and the underlying techniques for enabling components to cross development, deployment and execution environments.

## 10.4. FINAL CONCLUSIONS

A new generation of DES software that promotes the ease and speed of selection, customisation and assembly of the functionalities required by each simulation model may be needed to meet the expectations and concerns generated by the mass customisation market. The underlying development strategies have to co-evolve with computing advances to support quick-to-develop, quick-to-change, quick-to-assemble and quick-to-run robust software solutions.

However, will the new generation of DES software contribute to boost the application of simulation as a tool for decision making? What is the value of the new generation DES software for narrowing the gap between the theoretical acceptance of simulation and its practical use [83] as a tool for exploring complex systems? Answering these questions would be the subject of at least one more PhD thesis.