EPILOGUE

E.1. CUSTOMISATION OF DES SIMULATION SOFTWARE

As in many other application software domains, DES software needs development strategies which promote the ease and speed of customisation in order to respond effectively and efficiently to the expansion of the mass customisation economy. This thesis and the underlying research employed object, component and layered-oriented development paradigms to enable and support this customisation. This approach was selected because it supports the development of software solutions in successive 'generalise-specialise' cycles. That is, the software is vertically architectured by successive derivation of frameworks and libraries of functionalities from base classes to meet the specific needs of each model. Hence, this research attempts to link these three paradigms to allow this successive derivation.

It seems wise to ground speculative ideas by attempting their implementation within at least one domain and the application domain within which this has been attempted is the provision of DES software. However, the lessons learned from this research can be extrapolated to other application domains and, in subsequent research, might be one element in the creation of new development paradigms.

That is, in this thesis, discrete event modelling, integration technologies and discrete simulation and their interrelationships are addressed in order to explore a fully object-oriented component and layered-based development strategy. This differs from a conventional component-based simulation as it targets, eventually, full object-orientation and web-orientation. That is, components are objects which are manageable from within other components and may be spread across the heterogeneous Internet. Thus components may be remotely selected, negotiated, delivered, specialised and bound at compile and runtime. As it is clear from the earlier chapters, the DotNetSim prototype system does not permit composition over the Internet, though with adaptation this should be possible. Its main feature is the integration of heterogeneous components to allow the customisation of discrete event simulation software. Moreover, the components used in DotNetSim are composed into layers of functionality and these, in turn, can be composed into software solutions.

E.2. A STEPPING-STONE TO THE IDEAL ON-DEMAND SCENARIO

The long term objective for this research is captured in the vision for the development of simulation software described in chapter four. This is for on-demand software but, as yet, this cannot be fully realised despite the intense on-going computing research referred to in chapter one. Hence, this thesis looks at the nearer future and focuses on the object-oriented composition of components written in different programming languages, within different applications and from different technological generations into a single DES application. Hence, one view of this thesis is that it presents a stepping stone to the ideal on-demand simulation software; a short-term objective within a longer term vision. It focuses on a disarmingly simple question: how can discrete event simulation (DES) software be composed from object-oriented components that are built on top of generic applications? Further how can this be achieved regardless of the programming languages, packages and technological generations within which the components are developed? To examine these questions, the thesis describes the development of a prototype DES application.

The Event Graph modelling paradigm, the event-based simulation worldview and .NET Framework provided a convenient means to experiment with the composition of simulation applications from components developed within different environments which invoke each others objects as if they were their own. This led to the DotNetSim prototype which uses wholly object-oriented principles to integrate three coarse-grained components. The idea was to demonstrate how a simulation application can be composed from prefabricated components that cross programming languages, packages and technological generations to form a single application.

Were this research to be extended, it would be worthwhile investigating the substitution of other modelling methods and simulation worldviews, provided as components, with DotNetSim. They could also be integrated on top of other fully object-oriented frameworks. That is, components may be substituted and, there seems no reason why frameworks other than .NET should not be used, and these would be worthy subjects for further investigation.

E.3. ASSESSING THE SUCCESS OF THIS RESEARCH SO FAR

Chapter four described a vision for the development of simulation software and, in this section, we consider the issues raised in that chapter, with a view to assessing the contribution made by the research discussed in this thesis. Figure E.1 summarises those issues and indicates their layered relationships.

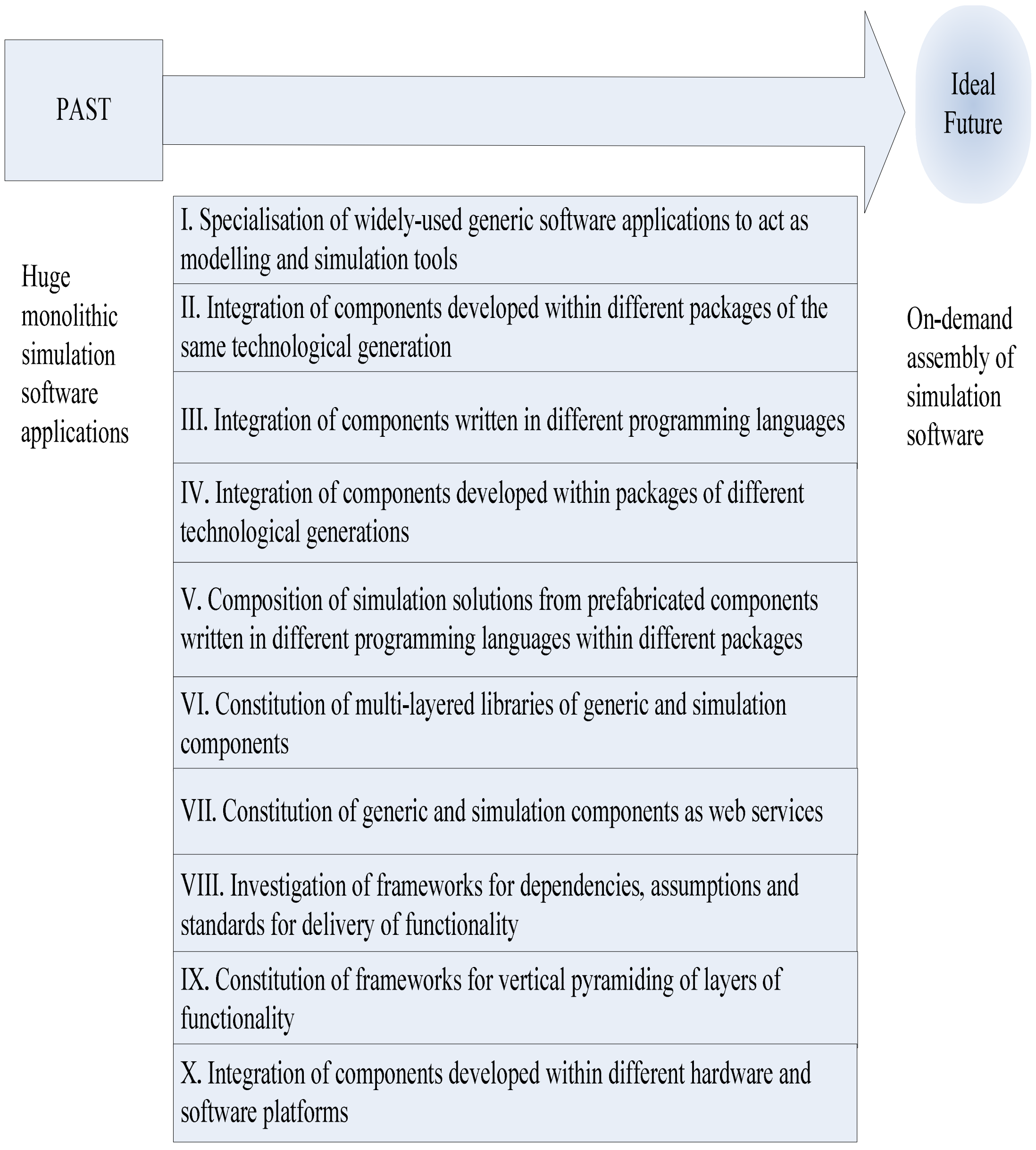
PAST
Ideal Future
Huge monolithic simulation software applications
On-demand assembly of simulation software
I. Specialisation of widely-used generic software applications to act as modelling and simulation tools
II. Integration of components developed within different packages of the same technological generation
III. Integration of components written in different programming languages
IV. Integration of components developed within packages of different technological generations
V. Composition of simulation solutions from prefabricated components written in different programming languages within different packages
VI. Constitution of multi-layered libraries of generic and simulation components
VII. Constitution of generic and simulation components as web services
VIII. Investigation of frameworks for dependencies, assumptions and standards for delivery of functionality
IX. Constitution of frameworks for vertical pyramiding of layers of functionality
X. Integration of components developed within different hardware and software platforms

Fig. E.1: From monolithic simulation software applications towards on-demand simulation software solutions.

Each of these issues is discussed below, which illustrates that component integration is multi-dimensional, depending on what needs to be integrated with what. This research project successfully showed how issues I to V could be resolved. It seems likely that the same principles, given more time, would allow resolution of issues VI and VII. The remaining three would require much more extensive work.

I. SPECIALISATION OF WIDELY-USED GENERIC SOFTWARE APPLICATIONS TO ACT AS MODELLING AND SIMULATION TOOLS

That generic software applications can be used as the basis of modelling and simulation tools is no new insight. In the work described in this thesis, the simulation tools were not just Visio and Excel applications, they were extensions of these applications to provide a basis for DE modelling and simulation. That is, they exemplify the object-oriented approaches that are needed if the vision underlying this thesis is to be realised.

Specially-written VBA components were developed to derive modelling and simulation features that exploit the built-in object models of Visio and Excel, i.e. the generic object models of these applications were instantiated to produce modelling and simulation functionality. In particular, Microsoft Visio ™ and Microsoft Excel™ were specialised to form the basis for an Event Graph modelling environment and the simulation output analysis template. The graphical user interfaces of Microsoft Visio™ and Microsoft Excel™ were also customised in order to automatically unload or hide any features which are not directly used by the event graphs and the simulation output analysis.

In a like manner, other widely-used applications of the Microsoft Office suite, e.g. Access™ and Project™, could be appropriately specialised to provide user input and output interfaces for DES modelling and simulation, though this has not been attempted in the work described in this thesis.

II. INTEGRATION OF COMPONENTS DEVELOPED WITHIN DIFFERENT PACKAGES OF THE SAME TECHNOLOGICAL GENERATION

As well as creating simulation support tools by the object-oriented extension of commonly used application packages, once created these must be integrated if they are to be of use. In this thesis, this is exemplified by the bi-directional exchange of data between components sourced in Microsoft Visio^{TM}, Microsoft Excel^{TM}, Microsoft Word^{TM} and Microsoft PowerPoint^{TM} in order to model discrete event systems as event graphs and report the modelling data and simulation results. These data exchanges operate through the instantiation of objects and the application of an object-oriented programming paradigm.

As long as other packages allow access to their underlying object models then any components sourced in other packages which support automation could be similarly integrated by manipulating other packages' object models as if they were their own. This has not been attempted in this research, but could be a project for future research.

III. INTEGRATION OF COMPONENTS WRITTEN IN DIFFERENT PROGRAMMING LANGUAGES

Within the .Net framework it should be possible to link components, written in different programming languages, into a single application – providing, of course, that the languages are .Net compliant. This cross-lingual integration was illustrated by the composition of the simulation engine from C# and VB.NET components that inter-operate as if they were written in the same programming language. Thus, for example, the class *randoms* which generates random numbers from the Uniform and Negative Exponential distributions, is written in VB.NET and is instantiated by the C# component which schedules the following events. This illustrates that other applications could be created from such cross-lingual composition.

However, it is important to consider when this would be worthwhile and there seems to be at least one obvious instance. This is when existing components need to inter-operate with other newly created components that may, for one reason or another, be written in a different source code.

IV. INTEGRATION OF COMPONENTS DEVELOPED WITHIN PACKAGES OF DIFFERENT TECHNOLOGICAL GENERATIONS

As with issues II and III above, this relates mainly to component re-use, but also to issues of convenience. That is, though it would be possible to recode components so that they are all of the same technological generation, this is not necessary, though some work is still required to support their inter-operation. This is illustrated in the integration of the Microsoft Office-based components, i.e., the modelling environment and the output analysis template with the .NET-based simulation engine. This was achieved by resorting to the PIAs (see chapter four) which allow .NET applications to bind Office components at compile and runtime so that the CLR can marshal them across contexts, processes or machines.

It is important to note that this integration required some considerable work, even when the principles of the PIA and CLR were well-understood. In particular, wrappers were developed to overcome the semantic and syntactic differences between .NET languages and the Visual Basic for Applications used to extend the Microsoft Office applications. However, this is hardly surprising. Though in earlier generations some applications were developed with the intention of linking them by hard coding to some specific applications, they did not aimed object-oriented integration to applications.

V. COMPOSITION OF SIMULATION SOLUTIONS FROM PREFABRICATED COMPONENTS WRITTEN IN DIFFERENT PROGRAMMING LANGUAGES WITHIN DIFFERENT PACKAGES.

Finally, the DotNetSim prototype is composed by the integration of three coarse-grained components developed within Microsoft Office applications and .NET framework. This demonstrates the cross-lingual and cross-package integration of object-oriented components into a single DES simulation software application. Each component crosses the upstream and downstream borders to manipulate the objects of other components as if they were their own regardless of the programming languages in which they were written and packages within which they were developed.

The components used in DotNetSim could be substituted by others that provide the same functionality but are based on other modelling and simulation paradigms. However, due to time constraints, this has not been done in this research since further prototyping is necessary to demonstrate the replacement of components.

In addition, as the DotNetSim prototype integrates coarse-grained components, further investigation into the granularity of the components and the usability of the composition is required. These issues link to the reusability of components and their integration in the creation of specific simulation solutions. The gains in reusability and flexibility from finer grained components should be contrasted with the increasing complexity of integrating a larger number of inter-related components. Though it is unclear at present where this balance lies, it seems important to investigate it.

Similarly, it is important to consider different levels of usage for the composition of simulation software solutions. That is, the levels of usage could vary from the black-box composition of appropriate components to the white-box selection, customisation and composition of simulation software solutions. With further work, different levels of usage might be supported in the DotNetSim prototype by integrating a graphical component that profiles the user and sets the appropriate usage. Thus, the first component of DotNetSim could be, for example, a graphical user interface which implements a questionnaire that leads to the customisation and transference of the required functionality in order to assemble specific simulation software solutions.

The two issues

VI. CONSTITUTION OF MULTI-LAYERED LIBRARIES OF GENERIC AND SIMULATION COMPONENTS

VII. CONSTITUTION OF GENERIC AND SIMULATION COMPONENTS INTO WEB SERVICES

were not attempted during this research. They require further prototyping to explore the integration of the .NET Framework with widely-used applications to develop multi-layered libraries of components written in different programming languages and within different applications. Some of these libraries, e.g. the simulation engine and the modelling environment, may be deployed as web services. If this were done, the composition of simulation solutions from resident prefabricated components and web services could be investigated through the prototyping.

The next three issues

VIII. INVESTIGATION OF FRAMEWORKS FOR DEPENDENCIES, ASSUMPTIONS AND STANDARDS FOR THE DELIVERY OF FUNCTIONALITY

IX. CONSTITUTION OF FRAMEWORKS FOR VERTICAL PYRAMIDING OF LAYERS OF FUNCTIONALITY

X. INTEGRATION OF COMPONENTS DEVELOPED WITHIN DIFFERENT HARDWARE AND SOFTWARE PLATFORMS

were also not attempted during this research and must be deferred to the medium and long terms as they probably will benefit from ongoing computing research. If successful, these could finally lead to the on-demand assembly of DES simulation software solutions which is the ideal stage considered in this thesis.

E.4. FURTHER PROTOTYPING TOWARDS ON-DEMAND SIMULATION SOFTWARE

Further prototyping is required to pull the DotNetSim project along the time line towards the ideal of on-demand simulation software. This is a long term objective which may even be achieved within a newer technological framework which will eventually provide specific support for the on-demand programming paradigm. Fig. E.2 places the DotNetSim project on the time line drawn towards the ideal future and summarises the issues that ought to be prototyped.

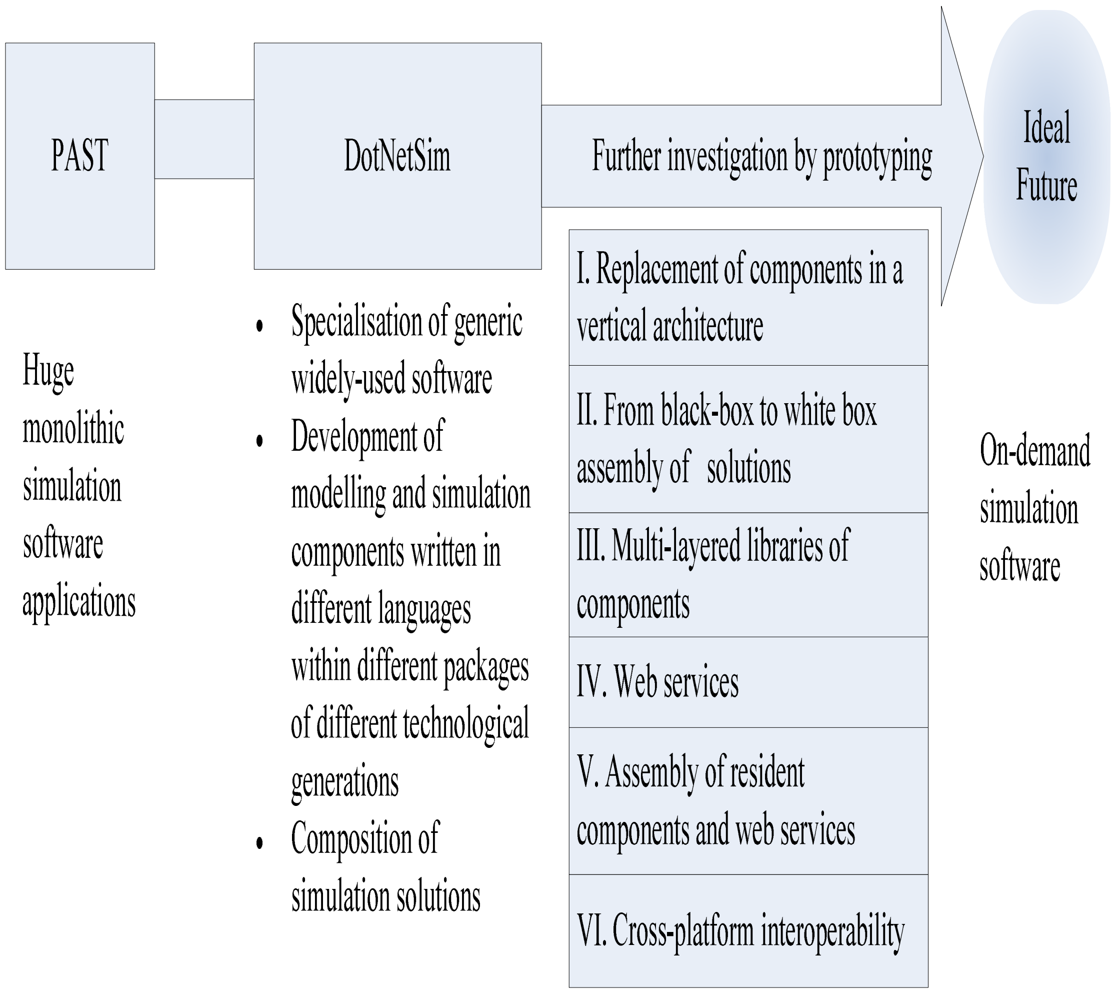
PAST
DotNetSim
Further investigation by prototyping
Ideal Future
Huge monolithic simulation software applications
• Specialisation of generic widely-used software
• Development of modelling and simulation components written in different languages within different packages of different technological generations
• Composition of simulation solutions
I. Replacement of components in a vertical architecture
II. From black-box to white box assembly of solutions
III. Multi-layered libraries of components
IV. Web services
V. Assembly of resident components and web services
VI. Cross-platform interoperability
On-demand simulation software

Fig. E.2: DotNetSim on the time line towards the on-demand simulation software.

As briefly discussed above, a new core component should also be prototyped within the .NET framework in order to investigate how different categories of users – developers, solution builders and end-users - may inter-relate with different grades of customisation. This component could be based on a server of functionality or a server of source-code and a code generator. That is, depending on the requirements of each simulation model and the skilfulness of the user, this component:

- integrates into the user's solution the functionality the user specifies. If the user, for example, requires a random number generator to sample from a Poisson distribution, the server will add to the user's solution a reference to the corresponding class of a library of components
- transfers to the user's solution the source code of the functionality the user specifies. If the user requires a variety of the Poisson distribution, the server will transfer to the user's solution the source code of the corresponding base class of the library of components. The user may then customise the distribution to meet his/her specific needs.
- generates the source code of the functionality which the user graphical or textually describes. If the user requires a specific functionality, the server generates the source code for a base class that provides the described functionality.

In this way, the user could be given the capability to select, modify and assemble the functionality required in each simulation model. However, achieving this is still a long way off.

E.5. BROADENING THE SCOPE OF THIS RESEARCH

The intense ongoing computing research on integration and interoperability is likely to affect a wide range of applications, other than discrete event simulation. This is because the increased demand for mass customisation is wide-spread. There may well be an increased demand for customised solutions and new integration technologies to support these. The research discussed here investigates the discrete event simulation domain and the Microsoft .NET Framework, but other domains and other frameworks may also be of interest. Such component-based composition has long been of interest in software engineering.

The exploration of this alternative approach to the development of DES software lays the foundation for the investigation of a paradigm oriented for the vertical architecturing of other software solutions. Rules, principles, assumptions and standards are needed for framing the vertical composition of software solutions from prefabricated functionality that the users select, negotiate the delivery, modify and assemble just in time. These must cover the development of components, their composition into layers of functionality and, in turn, the constitution of layers of functionality into pyramidal structures of functionality (see chapter three).

Software components, layers and pyramidal layered structures are simultaneously systems and sub-systems (see chapter two) that inter-relate through interfaces (see chapter three) to provide some emerging whole functionality under certain preconditions. The DotNetSim project brought new insights to the 'classic' issues (see chapter three) raised by this interrelationship between systems and sub-systems as it extended the OOP principles to the derivation of components and sub-components and their integration. The lessons learned from this experimentation with object-oriented specialisation of generic object models of various applications into modelling and simulation components and their integration (see chapter ten) could lead to further investigation of the encapsulation of functionalities within components and their delivery as services by means of object-oriented interfaces. That is, the main concepts

- granularity, location, interfaces, reusability and integration - revisited within the OOP principles by the DotNetSim project, lay the foundation for inferring general rules, principles, assumptions and standards for service-oriented programming paradigms. This, in turn, leads to on-demand programming paradigms.