1. INTRODUCTION

1.1. CONTEXTUAL ENVIRONMENT

The increasing demand for products and services which match unique and specific needs has driven customisation to the forefront of the economy [36, 93]. Many advances in various technological fields have transformed mass production into successive varieties of mass customisation, which envisions the satisfaction of individual needs when required but at mass production prices.

Consumers are currently used to tailoring a wide range of manufactured products and services, such as subscription to digital information, insurance or financial assets, by topping the base products with the options they select from a menu of derived products and services. It is often even possible to redesign the base products according to individual consumer preferences.

The consumption behaviour and expectancies associated with the customisation economy also extend to software products. As with other products, consumers take for granted that software fulfils its designated purposes, i.e. it is dependable, portable, friendly, recoverable, extensible, etc. and they desire it to be easily and quickly customisable to match their specific needs. Software users – developers, solution

builders and end-users – expect that they can, to a certain extent, select its functionality. They are increasingly uninterested in packages that fit all problems at the expense of large amounts of learning and use of resources. Moreover, they do not want to pay for functionality that is not required.

This leads to the on-demand software paradigm [118] which supported by intensive ongoing computing research, is driving application software from the development of all-inclusive packages towards the building of solutions. Solutions for concrete problems are intended to be assembled on the spot by selecting and negotiating the use of only the required functionality. That is, functionality is rented and paid per use, just-in-time and released immediately afterwards.

The idea is to enable users to build their own software solutions by the dynamic composition of components which are available over computer networks, particularly the Internet. This implicitly assumes that components are self-contained and that their composition results in self-forming and self-healing systems. Ideally, integration mechanisms should enable interoperability across hardware and software platforms, regardless of their technological specifications. Thus, software components, which reside in any fixed or mobile devices, would effectively cross-operate among different development, deployment and execution environments.

The complexity of this integration increases when it has to be done at runtime, as the risks for security and safety also arise. Thus, on-demand software is not yet a common practice.

The technological issues raised by the on-demand software paradigm motivate intense software research to investigate how the selection of the functionality, its negotiation, delivery and binding can be done at runtime. This includes topics such as the development of context-adjustable components, dynamic ascertainment of the

deployment environments, the programmable configuration and reconfiguration of software components and the underlying machine-interpretable metadata. Fig. 1.1 lists the ongoing research areas that sustain the on-demand software paradigm [58, 77, 122].

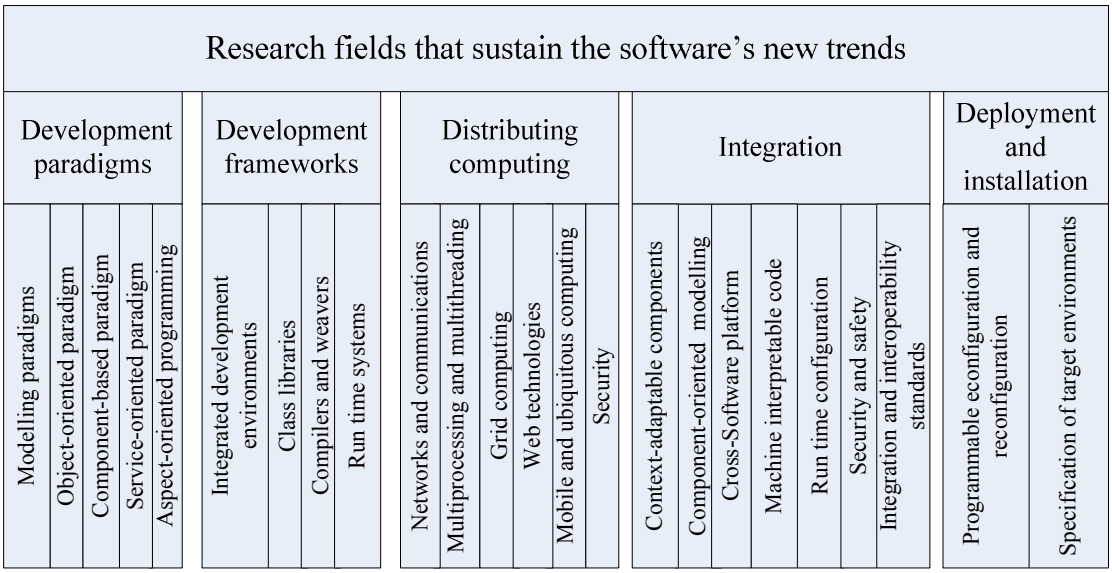

Fig 1.1: Ongoing research which sustains the on-demand software paradigm

1.2. ALTERNATIVE VISION FOR SIMULATION SOFTWARE

Developers of simulation software have responded to the increasing commercial demand for the customised solutions that the on-demand paradigm generates by adding new features and tools to their simulation packages. Whatever functionality the user demands - for example, a new n-dimension graphical interface, a particular optimiser or a code generator - is specifically developed, generalised and glued to the existing simulation packages.

The underlying reasoning is to ensure that the existing simulation packages have sufficient functionality to be customised to a large number of problems. This has led to huge, monolithic applications with functionalities that are constantly extended by addition of wizards, templates and add-ons in a 'generalising-customising-

generalising' development cycle. Though successful so far, this approach may be reaching its limit as these packages are increasingly difficult to customise and, hence, may lead to a slow response to the demand for customisation.

A small number of simulation software developers have already changed their packages, which reveals an emerging understanding that the mode of development of much current simulation software may not be appropriate to face up to the future demand for customised solutions.

An alternative approach is to compose simulation packages from prefabricated components which users may select, modify and assemble so as to acquire the functionality that suits each simulation model. In order to do this, simulation packages should be designed to comply fully with object-oriented, component-oriented and layered-oriented programming paradigms, thus easing customisation by plugging and unplugging components. Components might then be substituted to respond to specific needs at appropriate times. In addition, local or remote components might be selected and plugged at compile or runtime.

Fully object-oriented generic development frameworks, e.g. the Microsoft .NET Framework, might be used to derive progressively specific simulation software components and tools. In this way, the development of simulation software would be vertically architectured by deriving frameworks and libraries of simulation functionalities from the base classes, following a 'generalise-specialise' development cycle. Simulation solutions would consequently be able to reference the components and tools they need at the required level of specialisation and would use them by instantiation or by deriving new tools and ultimately new solutions.

Simulation software could thus keep pace with computing advances and co-evolve with computing towards on-demand software. Although this is still a dream,

simulation software could employ computing advances in component and layer-oriented programming paradigms and integration frameworks to beat a path that leads, eventually, to on-demand assembling of simulation solutions in an open market of components based on the Internet.

This thesis describes the DotNetSim project which explores how far the new Microsoft integration philosophy can pull discrete event simulation (DES) software towards fully object-oriented components that cross programming languages and packages to be linked in a single application and, with further work, might be deployed as web services.

1.2.1. THE DOTNETSIM PROJECT

The DotNetSim project, described in this thesis, investigates how different components developed in different programming languages within different packages can be linked in a single application. This is done by developing prototype software for simulation on top of widely-used packages. It is intended to show how this application might cover the entire requirements of a simulation application package including graphical user interfaces, simulation executive and output analysis.

Thus, the DotNetSim prototype integrates a graphical modelling environment developed within Microsoft Visio™ with a simulation engine developed within the Microsoft .NET Framework and an Excel-based simulation output analysis environment. Components integrate through the application of the object-oriented programming paradigm, which enables data to cross packages by instantiation of objects and invocation of the appropriate methods, that is, intermediate files are not required.

Our study employs an event-based simulation worldview and resorts to the Event

Graph methodology [108] to capture the application logic and the dynamics of DE models. Other graphical modelling methodologies and worldviews could, however, be substituted by swapping components.

1.3. THESIS ORGANISATION

This thesis consists of ten chapters laid out as shown in Fig. 1.2.

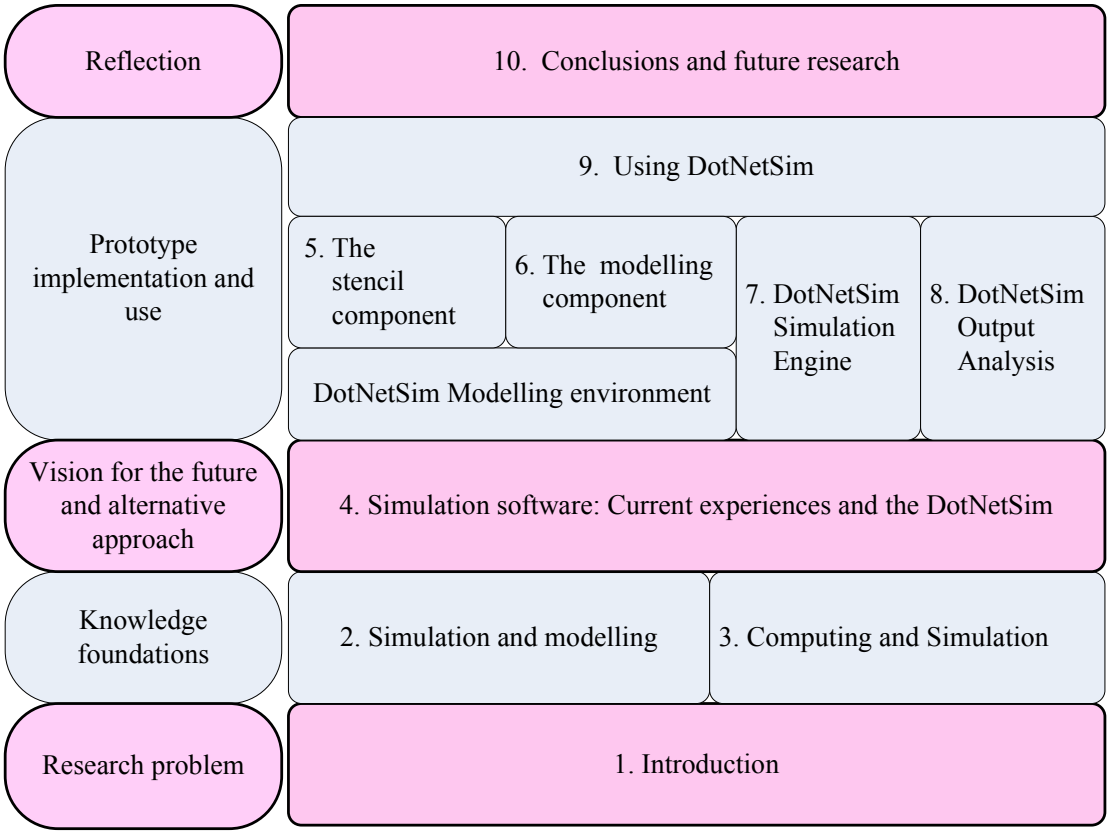

Fig. 1.2: Structure of the DotNetSim: Next Generation Simulation Software thesis

The thesis outline and a succinct overview of the remainder of its chapters are as follows:

Chapter 2 and chapter 3 define some relevant concepts from modelling, simulation and computing fields to clarify the relationship between these three disciplines, with the aim of understanding the current trends in DES software and to speculate on its future developments. Chapter 2 highlights the linkage of simulation and modelling,

emphasising that simulation interfaces with a wide range of disciplines, such as information systems, statistics, modelling and computing to structure problems and devise solutions. It concentrates on discrete event simulation and its use to analyse complex systems by repeatedly emulating their operation over a period of time. The iterative simulation process is decomposed into development stages, which are then explained. The logic structure of a discrete event (DE) model and its temporal dynamics are captured in two functional components: the application logic and the simulation engine. The application logic abstracts the relevant entities and their interrelationships while engaged in the activities or processes aimed at attaining the objectives of the system. The simulation engine executes the application logic through time. This calls for the definition of the time flow mechanisms, the 'classical' simulation worldviews and the simulation engine which they mirror.

Chapter 3 explains concepts of computer science which constitute the knowledge base of current trends and possible future developments of simulation software. It begins by reviewing the most commonly used programming paradigms and their underlying principles, rules and concepts. It continues by describing the composition of classes, objects and components. Interfaces, cohesion and coupling, binding times and location of components are also explained and the reuse of components, their integration and interoperability are discussed. It concludes by briefly reviewing the co-evolution of simulation and computer science to make sense of current trends in the simulation software and to beat a path towards future developments.

Chapter 4 devises an alternative vision for the development of simulation software based on the current expectations of users, the general trends in software development and some examples of developers who have already changed their development strategy. Current expectations go beyond proper functioning, portability, ease of use

and extensibility and refer mainly to ease and quickness of customisation. Users aim to tailor and assemble only the functionality that each simulation solution requires. This calls for alternative approaches to the development of simulation packages, rather than the existing huge, monolithic applications which are increasingly difficult to customise. This chapter discusses some examples of simulation software developers who have already changed their strategy so as to integrate simulation and generic software tools in order to facilitate customisation. It develops an alternative vision for simulation software that relies on a 'generalise-specialise' development strategy.

The DotNetSim project is then introduced to investigate how the latest Microsoft integration technologies can contribute to an alternative approach to the development of simulation software which would fit in this vision. Prototype software for simulation is proposed and its functional structure outlined. The technological background is briefly described.

Chapters 5 to 9 describe and comment on the implementation of the DotNetSim prototype and demonstrate its use. DotNetSim consists of a graphical modelling environment, a base simulation engine and an output analysis environment. Chapters 5 and 6 describe and comment on the DotNetSim graphical modelling environment and its emulation of the Event Graph methodology [108] for simulation modelling. The DotNetSim sits on widely-used Microsoft packages and provides a diagrammatical modelling environment for modelling DES systems to be simulated using the event-based simulation worldview.

Chapter 5 introduces the DotNetSim modelling environment, setting out its objectives and conceptual framework. Then, it begins its description and comments on the implementation of the DotNetSim prototype, focusing on the stencil component. The stencil component of the DotNetSim modelling environment uses Microsoft

Visio™ to implement the Event Graph extended modelling notation by 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 which perform the main operations on the stencil file. This menu is our first step towards the customisation of Visio™ solutions by exchanging data between Visio™ and Excel™. Finally, some comments are made on the implementation and reusability of the stencil component are made.

Chapter 6 focuses on the modelling component of the DotNetSim modelling environment. It describes and comments on the implementation of the functionality required to capture the logic of DE models diagrammatically as Event Graphs and to place the modelling data in relational databases. Upstream from the DotNetSim modelling component, the user might provide descriptive lists of the DE model, written within widely-used Microsoft applications, which are readable and convertible into Event Graphs. Downstream from the modelling component is the DotNetSim simulation engine (described in chapter 7) and other Microsoft applications by which a simulation is reported. Data is bi-directionally exchanged between the different packages by instantiating the object model of one application from within the other. Finally, some comments on the implementation of this component are made and this leads to further discussion on the value of the integration of different Microsoft applications for modelling DES systems.

Chapter 7 describes and comments on the implementation of the DotNetSim simulation engine, which prototypes an event-based simulation executive that runs, through simulated time, the models devised within the DotNetSim modelling environment. Written within the .NET Framework, it reads the modelling data collected as described in chapter 6, runs the event-based simulation and stores the

results of each replication into an Excel workbook. The integration of the DotNetSim simulation engine with other components based on other Microsoft applications is done by instantiating the corresponding classes. Upstream from the DotNetSim simulation engine is the Visio graphical modelling environment and downstream is the Excel-based output analysis environment for analysing and reporting the simulation results. This chapter also comments on the experience of developing this application by customising and integrating these components, leading to further discussion on the value of such .NET integration for discrete event simulation.

Chapter 8 focuses on the implementation of the DotNetSim output analysis environment, an Excel-based component for analysing and reporting the results of a DE simulation. It describes the functionality built on top of Excel™ to illustrate the use of generic software to derive data analysis and reporting tools which integrate with the other modelling and simulation components of the DotNetSim prototype. Like the other two coarse-grained components, the development of the DotNetSim output analysis environment stresses integration with other applications. Upstream from the output analysis environment is the simulation engine that instantiates this Excel template to place the simulation results. Downstream from it are the Microsoft applications used to report the analysis of the simulation results. Finally, some comments on the implementation and extension of this component are made prior to further discussion on the value of the integration of different Microsoft applications for modelling DES systems.

Chapter 9 demonstrates the use of the DotNetSim prototype by modelling and simulating a single server system which is a fine-grained component that occurs in a large number of DES systems. It begins by creating the Event Graph stencil with the modelling notation. Then, it models the single server system in an Event Graph within

the DotNetSim modelling component by following the menu driven modelling commands. Simulations of the single server model are then run by the DotNetSim simulation engine, which places the results in an Excel worksheet. These are displayed and analysed within the DotNetSim output analysis environment.

Chapter 10 summarises the thesis, reviewing the major arguments, the underlying research and the major contributions to an alternative developing strategy of DES software. It presents some lessons for the development of commercial DES software which were inferred from the strengths and weaknesses that the implementation of the DotNetSim prototype has revealed. Finally, some issues for future research are proposed. This includes the dependencies, assumptions and standards for the development of libraries of simulation components that can be deployed as web services.

Elements of this thesis have been published in

Carvalho A. (2006) DotNetSim: Net Generation simulation software. Poster Session: Advanced Institute of Management, Lancaster University.

Carvalho A., Pidd. M. (2005) What Value is Microsoft's .Net to Discrete Event Simulation? In: Kuhl M. E., Steiger N. M., Armstrong F. B., Joines J. A. (eds). Proceedings of the 2005 Winter Simulation Conference, PhD Colloquium, 4-7 December 2005: Orlando, FL, USA.

Carvalho A., Pidd, M. (2004) Stringing simulations together with DOT.NET, some experiences. In: Brailsford S., Oakshott L., Robinson S., Taylor S. (eds). Proceedings of the Simulation Study Group Two-Day Workshop, OR Society, 23-24 March 2004: University of Birmingham, UK.

Pidd M., Carvalho A. (2006) Simulation Software: Not the same yesterday, today or forever. Journal of Simulation (1). Palgrave Macmillan Ltd: Basingstoke, UK.

The companion CD-ROM to this thesis contains the thesis itself, the DotNetSim prototype source code and demonstration examples.