UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING, SCIENCE & MATHEMATICS

School of Engineering Sciences

The Development of a Hybrid Simulation Modelling Approach Based on Agents and Discrete-Event Modelling

by

Tai-Tuck Yu

Thesis for the Degree of Doctor of Philosophy

July 2008

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS

SCHOOL OF ENGINEERING SCIENCES

Doctor of Philosophy

THE DEVELOPMENT OF A HYBRID SIMULATION MODELLING APPROACH BASED ON AGENTS AND DISCRETE-EVENT MODELLING

by Tai-Tuck Yu

This thesis initially presents the work carried out on the research hypothesis – *agent-based simulation is better than traditional discrete-event modelling*. To test this assertion, a comparison of these two modelling approaches is made by way of a case study. The scenario, a global repair operation of a fleet of civil jet engines, is a real lifecycle costing example which involves logistics and is typical of problems commonly modelled using either of these paradigms.

To carry out the comparison, the method involved building a discrete-event model which matched the functions of an existing agent-based model as closely as possible. Rigorous control was applied during its implementation phase by way of formal code walkthroughs and model dynamic testing. Among the internal metrics, lines of code provided an estimate for model size while the McCabe Cyclomatic Number measured structural complexity. The external software quality of maintainability was derived from these metrics and estimated by modelling experts through Delphi sessions. The dynamic performance of each model was determined by the execution times of successfully completed simulation runs over a range of engine fleet sizes.

This research went on to develop a hybrid approach (which is currently the subject of a Rolls-Royce patent application) which draws on the strengths of both agent and discrete-event paradigms. In order to combine agent roles and discrete-event processes, a new model was implemented using a three-layered architecture. A full fleet simulation was developed using this hybrid approach. Although the code size is slightly larger and run times slightly longer than the conventional model, the thesis argues that, crucially, it is more maintainable as it reduces the conceptual gap between problem and model.

Contents

List of Figures

List of Tables

Declaration of authorship

I, *Tai-Tuck Yu*, declare that the thesis entitled *The Development of a Hybrid Simulation Modelling Approach Based on Agents and Discrete-Event Modelling* and the work presented in it are my own and have been generated by me as the result of my own original research. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University;
- Where any part of this report has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- Where I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this report is entirely my own work;
- I have acknowledged all main sources of help;
- Where this report is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- Parts of this work have been published and some articles are under review as:

1. ***Patent***

- Tai-Tuck Yu, J.P. Scanlan and G.B. Wills, 'Agent-like Discrete Event Modelling Software Architecture', *In association with Rolls-Royce plc*, 2008.

2. *Journals*

- Tai-Tuck Yu, J.P. Scanlan and G.B. Wills. ‘Flexible Model Building Using a Hybrid Approach’, *Computers in Industry* (Prepared, awaiting filing of patent)

- Tai-Tuck Yu, J.P. Scanlan and G.B. Wills. ‘Agent-Based and Traditional Discrete-Event Modeling in Value-Driven Design of Gas Turbines’, *Journal of Computing & Information Science in Engineering* (Prepared, awaiting filing of patent)

3. *Conference*

- Tai-Tuck Yu, J.P. Scanlan and G.B. Wills, ‘Traditional Discrete-Event Modelling and Agent-Based Modelling: A Quantitative Comparison’ In *Proceedings of 7th AIAA Aviation Technology, Integration and Operations Conference (ATIO)*, Belfast, September 2007.

4. *Internal reports*

- Tai-Tuck Yu, J.P. Scanlan and G.B. Wills, ‘A Comparison of Agent-based Modelling and Traditional Discrete-event Modelling’, *Technical Report No. ECSTR-LSL07-006,* 16th November 2007, University of Southampton. (This report is cited in A. Stranjak *et al.* ‘A Multi-Agent Simulation System for Prediction and Scheduling of Aero Engine Overhaul’, In *Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)- Industry and Applications Track*, May, 12-16., 2008, Estoril, Portugal)

- Tai-Tuck Yu, J.P. Scanlan and G.B. Wills. ‘Traditional Discrete-event Modelling With Agent Characteristics’, *Technical Report No. ECSTR-LSL08-001*, 15th March 2008, University of Southampton.

5. *Presentation and masterclass*

- Tai-Tuck Yu, ‘Modelling & Simulation’, *A masterclass held for the Life Cycle Costing Department, Rolls-Royce, Bristol*, 16th October 2007.

- Tai-Tuck Yu, ‘Agent-based and Discrete-event Simulation, *A presentation and discussion held for the Life Cycle Costing Capability Improvement Council, Rolls-Royce, Derby*, 16th January 2008.

Signed: ……………………………

Date: ………………………………

Acknowledgements

Although this is the only place in the thesis where the heart can be allowed to precede the head, my acknowledgement of the help I received from various people in different ways is no less valid. My expressions of gratitude are heartfelt and sincere.

First of all, I am greatly indebted to Prof. J.P. Scanlan and Dr. G.B. Wills for their honest, timely, and enthusiastic encouragement, as well as their firm, fair, and friendly guidance during the preparation of this thesis. Their wide knowledge of simulation, software metrics, and the research process has been of immense help in directing me towards worthwhile areas of research. Their personal friendship has provided a window for viewing interesting facets of academic life.

Also, I am grateful to Rolls-Royce Group plc, Derby for allowing me to use an agent-based engine fleet simulation model developed within their Strategic Research Centre as the basis for this case-study. In particular, I wish to record my thanks to Mr. Peter Swann and Dr. Armin Stranjak for their help with the agent-based model. A special note of thanks is also due to Mr. David S. Knott for easing forward the supply of this essential piece of enterprise software and data as well as for initiating and driving the process to patent the agent-like discrete-event modelling architecture.

Last, but not least, without the constant, unstinting, and loving support of my wife, this thesis would not have been possible. She has been a true better-half, often enduring quiet quarters while I was away from the haven of home in pursuit of this postgraduate degree.

The subject of this research largely overlapped one of the areas of investigation in the IPAS (Integrated Products And Services) Project. As they also had closely similar aims, it was therefore absorbed into the project. The research is supported by the Rolls-Royce University Technology Centre for Computational Engineering at the University of Southampton.

Nomenclature

AAII	Australian AI Institute
ABM	Agent-based model
ACM	Association for Computing Machinery
ADEM	Agent-like discrete-event model
AI	Artificial intelligence
API	Application programming interface
ASC	Aftermarket service centre
ASCII	American Standard Code for Information Interchange, a standard for encoding characters.
C	A general-purpose procedural programming language
C++	A general-purpose OO programming language
CAS	Complex Adaptive System
CBT	Complete binary tree
C-K	Chidamber-Kemerer metrics for OO programs
COBOL	Common Business-Oriented Language, a general-purpose programming language.
COCOMO	Constructive Cost Model
COTS	Commercial off-the-shelf, usually applied to software rather than hardware.
CPU	Central Processing Unit – the processor or logic unit for executing computer programmes.
CRV	Component repair vendor
DDES	Distributed DES
DEM	Discrete-event model
FIPA	Foundation for Intelligent Physical Agents, an IEEE Computer Society standards organization.
FORTRAN	Formula Translating system, a general-purpose, procedural, imperative programming language.

Gaia	An agent-oriented software engineering methodology
GPSS	General Purpose Simulation System, a specialised simulation language for discrete-event modelling.
H-K	Henry-Kafura information flow complexity for software systems
HLA	High Level Architecture, a general-purpose architecture for distributed modelling defined under IEEE Standard 1516.
IEEE	Institute of Electrical and Electronics Engineers, Inc.
IPAS	Integrated Products And Services, a research programme funded by the UK Technology Strategy Board's Collaborative Research and Development Programme and Rolls-Royce plc.
ISO	International Standards Organization
JADE	The Java Agent Development Framework
Java	A general-purpose OO programming language, sometimes taken to mean a computing platform.
JVM	Java Virtual Machine
LOC	Lines of code
MCN	McCabe Cyclomatic Number
MoD	Ministry of Defence
NASA	National Aeronautics and Space Administration
OHB	Overhaul base
OO	Object-oriented
PC	Personal computer
PES	Pending event set
PQ	Priority queue
PSC	Parts service centre
RR	Rolls-Royce plc
RTI	Runtime Infrastructure, a component of HLA.
SD	System Dynamics, a modelling paradigm.
SIMSCRIPT	An English-like general-purpose simulation language
SIMULA	A general-purpose simulation language, also regarded as the first OO programming language.
SPEEDES	Synchronous Parallel Environment for Emulation and Discrete-Event Simulation, a simulation framework.
SRC	The Strategic Research Centre, Rolls-Royce plc, Derby

TRL	Technology Readiness Level
UML	Unified Modelling Language, a language notation for documenting OO models of systems
UNIX	A computer operating system developed in 1969 at Bell Labs.
WMC	Weighted methods per class
WSC	Winter Simulation Conference
XML	Extensible Markup Language, a general purpose specification for creating custom markup languages.

Chapter 1
Introduction

In an engineering business operating in a free and open market, it is a truism that to stay still is to lag behind one's competitors. It follows that, if a business were to survive, thrive and remain competitive, it will always need to innovate and move forwards by reducing costs, improving its products, and simplifying its processes. Of course, such tasks underpin the customer-facing, non-technical activities like marketing, sales, and after-sales service as much as the purely technical ones. In manufacturing, the inescapable and highly desirable goal of making things cheaper, better, and faster can often be costly. This is especially true where a product is complex and its initial acquisition cost may be but a minor component of its whole lifecycle cost. Therefore, it is imperative that, after a product's initial entry into service, all subsequent changes must be correctly identified and implemented at the first attempt if at all possible.

A common traditional method of minimising errors and containing start-up cost is to make a physical prototype using a pilot process which contains all elements of the change required. This approach may be iterative and it enables decisions to be either confirmed or modified before commitment to a programme of full-scale production. Although it can prevent expensive and catastrophic errors from being made, but because physical entities are involved, the product time to market can seldom be reduced significantly.

However, the advent of cheap, high-powered computers and user-friendly application software have made the mathematical modelling of engineering products and processes readily available to enterprises of all sizes. In contrast to physical prototyping, structural changes to the mathematical models, like the addition of previously validated objects and subsystems, can be made even '*on the fly*' as the models are virtual entities. These software modelling tools enable different solutions to be explored relatively cheaply and quickly, thus allowing a good enough, but not necessarily the globally optimal, solution to be selected while at the same time shortening the product time to market. It is nevertheless necessary to remember that a tool which can bring about such benefits quickly also has the real potential to deliver catastrophes just as quickly if it is handled carelessly (Neumann, 1993).

1.1 Simulation and engineering processes

Although simulation is perceived as similar to, but is in essence different from, the real-world, it is nevertheless an appropriate technique for imitating and understanding complex systems (Simon, 1998). Simon further proposes that simulation is not only an aid for studying poorly understood systems but can itself be a source of new knowledge. This is because, in practice, knowledge is constructed from the roof down, not from the foundation up, and that makes it possible to discover incrementally finer details at lower and more fundamental levels. Robinson (2003) puts it succinctly when he describes simulation as the *'experimentation with a simplified imitation (on a computer) of an operations system as it progresses through time, for the purpose of better understanding and/or improving that system'*. Expressed in this way, simulation as a tool for abstraction is no older than the electronic computer but, in all its forms, imitation of the real world is very much older.

Although it is possible for some processes commonly encountered in an engineering manufacturing environment to be modelled mathematically to yield analytical and deterministic solutions, it is often the case that non-trivial, practical systems can easily become too complex for such a solution to be attempted (Banks *et al.*, 1999; Law and Kelton, 1999). In practice, these engineering processes are stochastic and dynamic, for example, because of uncertainties introduced by human participation and the discontinuities brought about by machine unreliability. To gain some understanding of these systems, the usual approach is to construct computer simulation models at the appropriate level of detail so that various *'what if'* scenarios can be studied by varying the models' operating parameters and analysing their outputs. Once an understanding has been achieved, the models can be used to predict system behaviour.

1.2 Software metrics

The demand for greater realism in modelling inevitably leads to large program sizes as the scope is widened and model granularity becomes finer. A detailed, high-fidelity simulation model can become unmanageably large and complex if its development is not strictly controlled. Software metrics like size, structural complexity, maintainability, reliability as well as many others have been subjected to extensive study over the past 35 years. Although a goal of such metrics is to characterise a software program or project, its chief use in a commercial environment is to provide an objective and predictive measure so that managers can control costs and resources in a software development project. The often used assertion by DeMarco (1982) puts it baldly as – *You cannot control what you cannot measure*. While DeMarco refers specifically to absolute, quantitative measures, it can apply just as well to ordinal, qualitative measures.

In some respects, depending on an inappropriate metric is worse than having no metric at all. For example, as programming languages have evolved from assembly language to object-oriented language, a frequently used metric like lines-of-code (LOC) becomes invalid when used to compare programmer productivity across different languages (Jones, 1994). When used on its own, the misleading and false message it gives is that productivity is higher when a programme is implemented in assembly code rather than object-oriented code where, from experience, the opposite is true. Therefore, care must be exercised when selecting metrics for comparing programmes implemented in different languages to ensure that the metrics are suited to the task.

1.3 Aims of this research

The research to be undertaken is a case study which compares an agent-based simulation model with a functionally identical traditional discrete-event model. The benefits and drawbacks of both modelling paradigms, as exemplified by these two models, are to be measured thus giving an objective and quantifiable comparison.

Although the comparison will be largely quantitative, the qualitative aspects of these modelling paradigms will not be excluded. When the models have been evaluated, an alternative modelling architecture will be considered by combining their beneficial characteristics.

1.4 Statement of research

The statement of hypothesis, including the null hypothesis, is –

- H_1: Agent-based modelling is better than traditional discrete-event modelling.
- H_0: Agent-based modelling is *not* better than traditional discrete-event modelling.

The context of the hypothesis is described as follows –

- The hypothesis is restricted to the class of problem being modelled, which is in the logistics and supply chain area, with particular reference to the global repair operation of a fleet of large, modern jet engines for civil aircraft.
- The hypothesis is also restricted to the phase between the requirements analysis and the system maintenance activities of the traditional '*Waterfall*' systems development lifecycle. As set out in Figure 6-6, the lifecycle activities included in this phase are the following – '*Requirements analysis*', '*Design*', '*Developent*', '*Integration and test*', '*System implementation*', '*System operation*', and '*Maintenance*'.

1.5 Methodology for comparing the modelling paradigms

The steps to be taken so that a comparison can be made between the two modelling paradigms are listed below –

- Construct and validate a discrete-event model which is functionally identical to the existing agent-based model implemented by Rolls-Royce (RR) Strategic Research Centre (SRC).

- Select an appropriate set of software metrics so that the internal and external software properties of the model can be objectively measured or assessed.
- Analyse the code for both models and extract the selected metrics for each software class or module. Values for sub-models or smaller partitions of the model can be calculated using these primitive measurements.
- Measure the time taken by each model to complete a simulation run under a range of model inputs.
- When some understanding of the paradigms' benefits and drawbacks has been reached, an alternative modelling approach may be formed by drawing from their better characteristics.
- Construct a model using the new approach and compare it against the previous two models.

1.6 Structure of thesis

The remainder of the thesis consists of seven further chapters the contents of which are as follows –

- Chapter 2 contains overviews of computer modelling and software metrics.
- Chapter 3 considers the development of traditional discrete-event simulation including distributed modelling and the introduction of artificial intelligence into this modelling paradigm.
- Chapter 4 sets out the progress of agent-based simulation up to the present time.
- Chapter 5 examines some software metrics which are likely to be useful for this research.
- Chapter 6 describes the experiment to be carried out on the two simulation models, including the case study scenario, the gathering of model metrics, and a discussion of the results.

- Based on the results of the preceding chapter, Chapter 7 presents a modelling architecture which combines the best features of the two modelling paradigms.
- Finally, Chapter 8 sets out the conclusions, the contributions of the thesis, and the proposals for future work.

Chapter 2
Overview of Modelling and Software Metrics

2.1 Introduction

In this chapter, an overview of modelling paradigms and software metrics is presented. First, significant developments in modelling which are relevant to the engineering context are considered. After that, software metrics which may be used to characterise simulation models are presented.

The extensive use of computer simulation is seen not only in engineering applications but also in medicine, healthcare, economics, business, management science, public administration, the traditional sciences, and computer games. In the social sciences, models have been constructed to study group dynamics (Hoffmann *et al.*, 2007) as well as the psychosocial behaviour of human individuals and groups, for instance Thoyer *et al.* (2001) and Izquierdo *et al.* (2008). Modelling is a relatively cheap and convenient investigative tool for imitating the real world. Its expanding use in such diverse fields marks it out as an important tool for research and decision support. Furthermore, it is often the only tool available for exploring and understanding how complex engineering systems work.

A historical overview of simulation over the past 50 years is given in Nance and Sargent (2002). The overview focused on the evolution of discrete-event modelling as the dominant technique for system analysis especially in operations research and the management sciences. In observing the development of simulation generally, they noted the symbiotic relationship between simulation and computer science. In both disciplines, contemporaneous needs drove developments which brought about mutual benefits.

They pointed out in particular that *'the external influences on simulation are dominated by those associated with digital computer technology'*. Primarily, these are the advances in computer hardware, programming languages, graphics, networks, and the World Wide Web all of which can be employed to make models more realistic, responsive, and accessible.

In contrast to the external influences, the internal factors includes the significant development of –

- ***Modelling methodology.*** This encompasses event-list management, automated and semi-automated modelling techniques, time-flow mechanisms, and validation and verification.
- ***Analysis methodology***. Conway (1963) identified simulation as more than just model building. His paper initiated a large area of research which has since expanded to include simulation experiment design and comparison of alternatives, variance-reduction techniques, output analysis, metamodels, and optimization, and the application of artificial intelligence in simulation.

Both modelling and analysis methodology are relevant to this review. Those aspects which are immediately applicable to the context of this thesis will be considered in greater detail in Chapter 3, on Traditional Discrete-Event Modelling, and in Chapter 4, on Agent-Based Modelling.

2.2 Categorisation of simulation

Nance and Sargent (2002) categorise simulation in two ways and they are based on –

- ***The objective of the simulation study*** which may be system analysis, education and training, acquisition and system acceptance, research, and entertainment. It may be noted here that where the objective is the analysis of complex engineering systems, the intent is invariably to mimic behaviour so as to understand and then improve the performance of those engineering systems.

- ***The representation of time and state in the simulation model*** which is to say whether continuous modelling, discrete-event modelling, or a hybrid of continuous and discrete-event modelling is involved. It should be stressed that in these modelling paradigms, both time and state are explicitly represented. Strictly, modelling using the Monte Carlo method may not be included in this categorization scheme for the reason that while state sequencing is present, time is not explicitly represented.

It is the second category which is better known and is the one used in most journal papers, conference proceedings (for example, Maria (1997) and Carson (2004)), and standard textbooks on simulation (for example, Banks *et al* (1999), Law and Kelton (1999), and Robinson (2003)). Although the first category is useful, this thesis will consider the second not only because it is overwhelmingly used in the engineering sciences but also to be in keeping with by far the larger body of published literature. A further reason for doing so is that this method of classification has a relatively long history (Shannon, 1977) and is clearly set out in Figure 2-1 as a part of Shannon's taxonomy of simulation languages.

Today, Shannon's classification of 'continuous change' and 'discrete change' are commonly referred to as 'time-driven' and 'event-driven' modelling respectively. This naming convention of the two branches of modelling is in current use and can be found quite frequently in the more recent conference and journal papers, for instance, those from the Winter Simulation Conference and the ACM Transactions on Modeling and Computer Simulation.

Four major modelling paradigms emanate from these two branches and they are traditional discrete-event, agent-based, system dynamics, and dynamical systems (see Figure 2-2). Of these four approaches, the dynamic systems approach is not encountered as frequently as the other three in the modelling of complex engineering systems such as those for logistics and supply chains. This may be because the dynamic systems approach involves well-understood physical systems with behaviour which evolves over time according to a system of differential equations. It does not lend itself well to the discontinuous, event-driven nature of logistics and supply chain problems where large numbers of items with diverse, individual properties may have to be modelled.

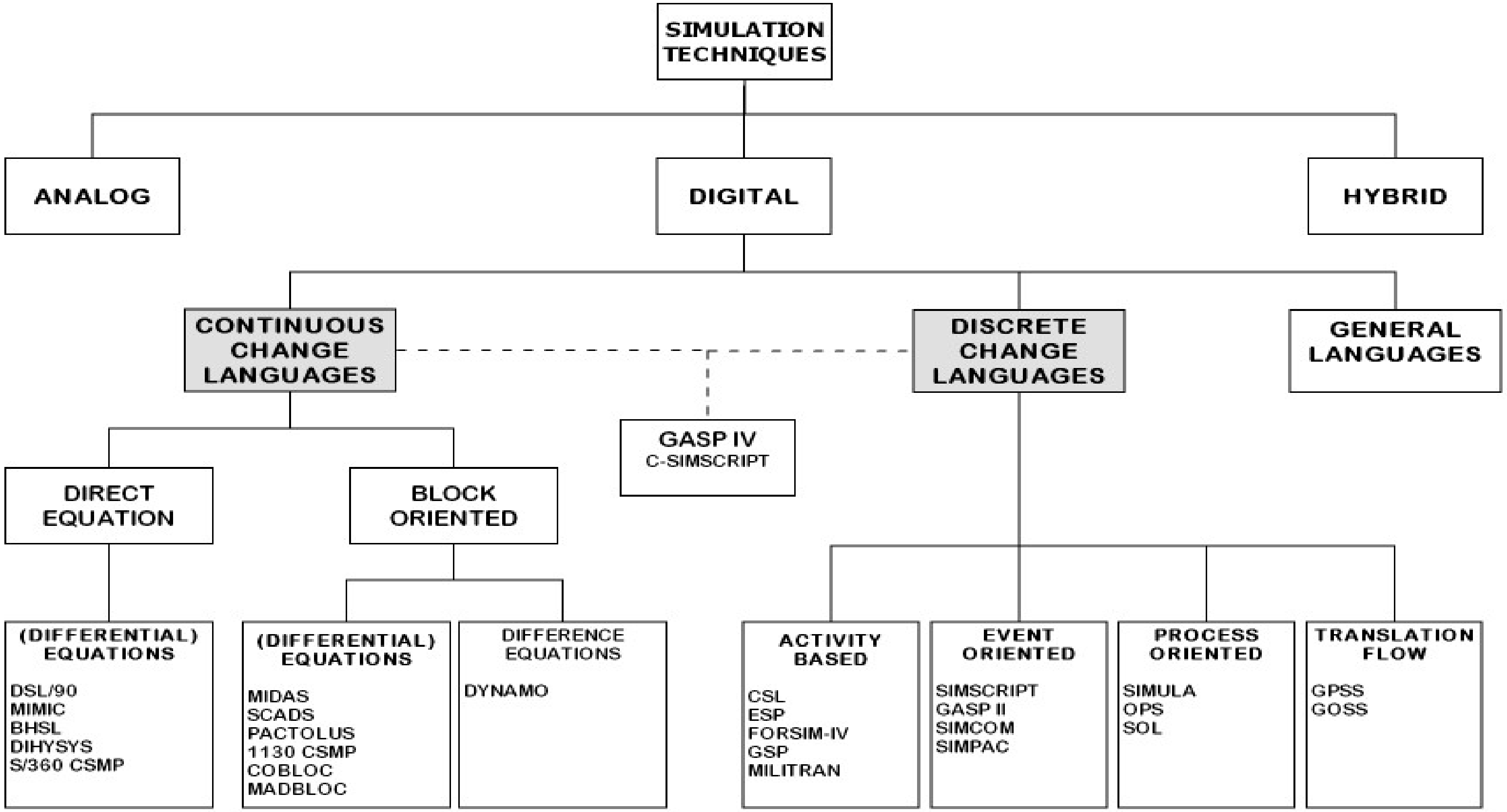


Figure 2-1: Simulation techniques and languages (Shannon, 1977)

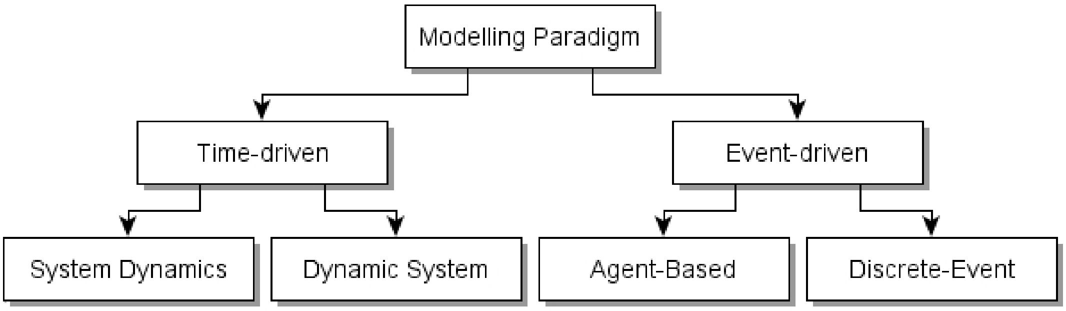


Figure 2-2: Current classification of modelling paradigms

The categorization of modelling approaches set out by Borshchev and Filippov (2004) reproduces the hierarchy shown in Figure 2-2. Their scheme is illustrated in Figure 2-3 which also indicates the approximate range of scale and levels of detail and abstraction at which any of the four paradigms may be employed. It gives a clear signal that none of the paradigms on its own can be used to model all conceivable scales and levels of a system. Therefore, in a model which has a large range of detail or abstraction levels, there is a good likelihood that one or more modelling paradigms will have to be used appropriately (see Section 2.3 Misuse of modelling paradigms).

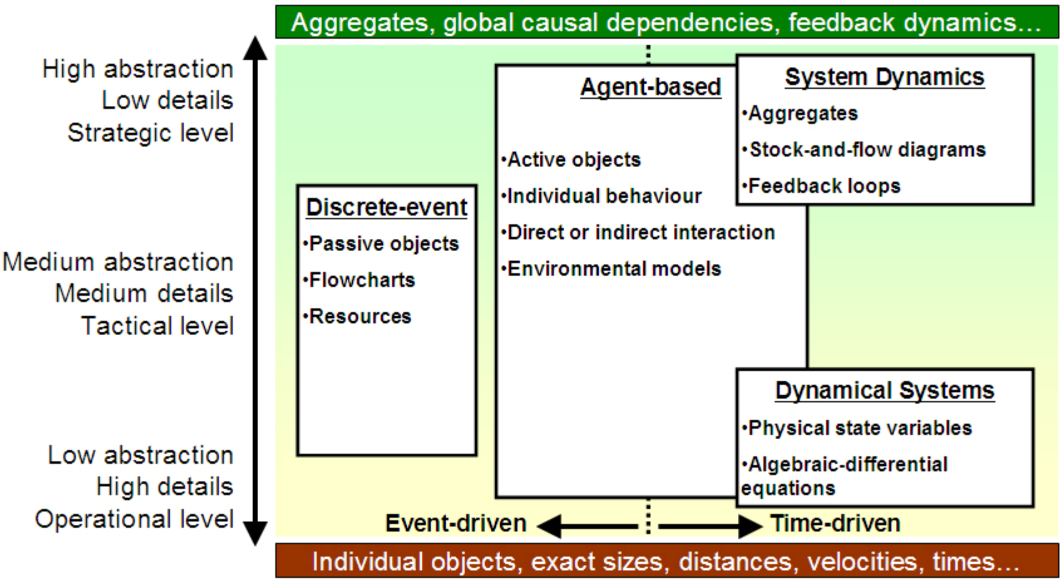


Figure 2-3: Modelling paradigms and regions of applicability (based on Borshchev and Filippov, 2004)

2.2.1 Time-driven modelling

Time-driven modelling, where the model time component is advanced at fixed and regular intervals, is represented in the main by two approaches – dynamic systems and System Dynamics (SD).

2.2.1.1 Dynamic systems

As indicated earlier in Figure 2-3, the dynamic systems paradigm is used to model physical systems where low abstraction and high detail levels are required. In particular, this deterministic, mathematical approach involves the integration of algebraic differential equations over time. The trajectories of the state variables, once given an initial value, can be predicted with certainty. For example, the dynamic response of an electronic control system or a mechanical mass-spring-damper system to various types of inputs can be studied using a commercially available modelling tool like MATLAB Simulink (Mathworks, 2007), SDX (SDX, 2005), and Extend™6 (Extend, 2005).

Figure 2-4 shows how a dynamic system can be constructed by connecting block functions together to obtain the output of a second order differential equation by evaluating it as two first order differential equations.

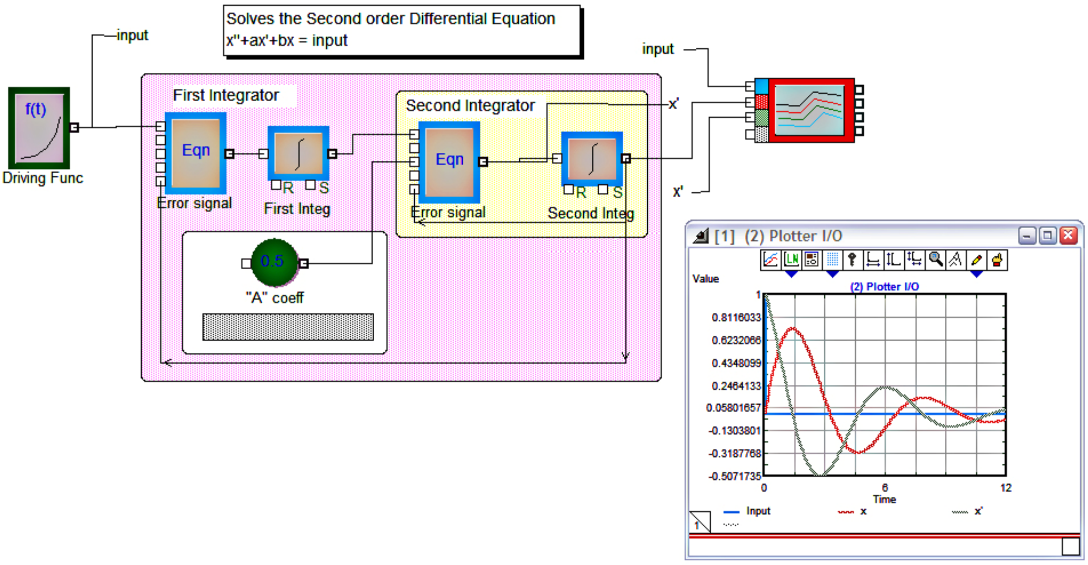


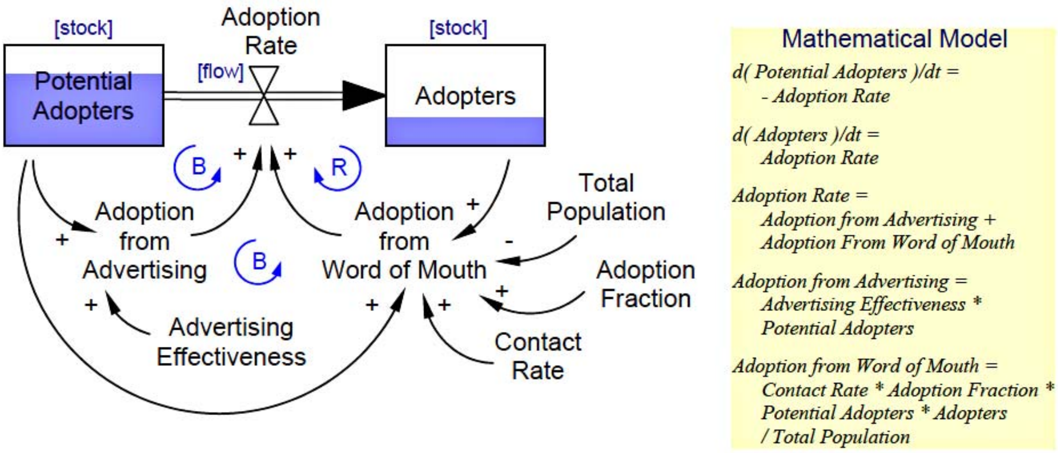
Figure 2-4: A simple example of a dynamic system model (Extend, 2005)

2.2.1.2 System Dynamics

This modelling paradigm has been elucidated by its pioneer, Jay W. Forrester, as –

> *The study of information-feedback characteristics of industrial activity to show how organizational structure, amplification (in policies), and time delays (in decisions and actions) interact to influence the success of the enterprise.* (Forrester, 1961)

System Dynamics (SD) is characterised by its modelling of a system as stocks (or stores), flows, time delays and feedback loops. Using these constructs, a complex and homogeneous system can be modelled as a causal structure with flows and feedback loops linking the stocks. The distinguishing feature of SD is its use of feedback loops. It recognises that the outputs of a node can act on and change the environment in which the system operates and thus modify subsequent inputs to that node or some other node. Such feedback can be either positive (indicated by the 'R' reinforcing feedback loop in Figure 2-5) or negative (indicated by the 'B' or balancing feedback loop). It can also sometimes result in non-linear behaviour which is often found in complex systems.



d(Potential Adopters)/dt =
- Adoption Rate

d(Adopters)/dt =
Adoption Rate

Adoption Rate =
Adoption from Advertising +
Adoption From Word of Mouth

Adoption from Advertising =
*Advertising Effectiveness **
Potential Adopters

Adoption from Word of Mouth =
*Contact Rate * Adoption Fraction **
*Potential Adopters * Adopters*
/ Total Population

Figure 2-5: Elements of System Dynamics modelling (Borshchev and Filippov, 2004)

SD is a top-down modelling approach and as such it uses aggregated values to represent stocks. It is difficult, but not impossible, for it to model heterogeneous populations where the effect of clustering and individual behaviour may be important. It can achieve this by segregating a large population into smaller and related groups, but still with homogeneous properties which are more tightly defined. Although it has been used with reasonable success in the understanding of supply chains and logistics networks, it is nevertheless limited by its requirement that the input variables have inherently uniform properties. Therefore, this paradigm is commonly used to model problems where abstraction is high and details are low. SD is typically used to model problems such as global population dynamics, the macro-economics of a country, ecological systems, and national health systems.

2.2.2 Event-driven modelling

Event-driven modelling is distinct from time-driven modelling in that its fundamental unit for measuring model progression during a simulation run is the event rather than time. A model advances from event to event in variable time steps rather in constant time steps. Event-driven modelling is represented in the main by two approaches – discrete-event modelling (DEM) and agent-based modelling (ABM).

2.2.2.1 Discrete-event

Traditional DEM is centred on processes, which may be described in other words as logical sequences of activities. It also requires a modeller to consider the resources and constraints which should be applied to the processes.

What occurs within an activity can frequently be abstracted as a time delay and it is a basic assumption of this modelling paradigm that nothing of consequence occurs between successive activities. They do not differ much from engineering processes where nothing which can affect the final result exists in the time intervals between consecutive activities. Using an event-driven paradigm for such processes enables computationally efficient models to be implemented especially those where the inter-activity time is large when compared with the time taken by the activity itself.

Two concepts are fundamental to discrete-event simulation and they are –

- The ***simulation object*** which contains a set of variables describing its state and attribute. Examples of simulation objects include various types of queues, timers, random number generators, and the items which flow through the processes.

- The ***event***, assumed to occur instantaneously in simulated time (and not real time), acts on the simulation object to change its state and possibly to schedule future events for any object within the model. An event can mark the start or end of an activity.

A model is constructed by linking simulation objects together to form activities and processes. The process shown in Figure 2-6 consists of time delays, queues, and a gate through which items (aircraft in this example) are passed at simulation times scheduled by a data structure called the event list or event calendar. The event list is essential to the operation of the model and much research effort has been expended in making it as efficient as possible. In a large model where a large number of items have to be processed individually, the computation time required for managing the event list can be significant. Therefore this data structure will be considered in greater detail in Chapter 3.

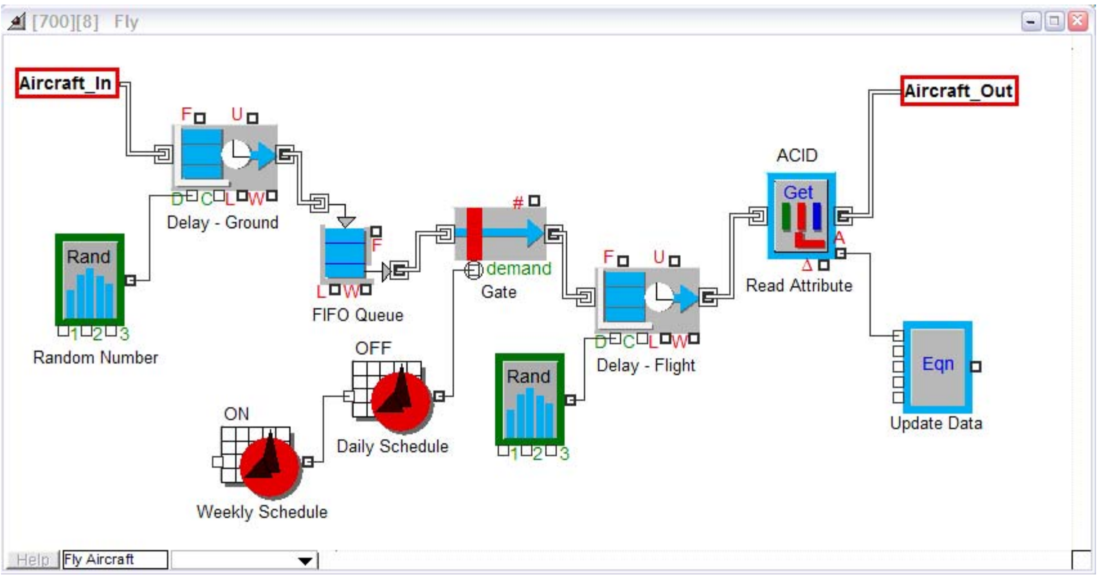


Figure 2-6: Components of a simple traditional discrete-event sub-model

2.2.2.2 Agent-based

ABM draws from a number of different disciplines like systems science, complexity science, management science, and computer science for its theoretical foundations, conceptual world view, and philosophy (Macal and North, 2006).

In stark contrast to the other three modelling paradigms outlined earlier, agent-based modelling (ABM) is a bottom-up approach while the others are top-down. Where the top-down approach requires a complete overview of a system as well as central control of the model, it may be inferred that in a bottom-up approach neither of those requirements is absolutely necessary. It is therefore possible to build a multi-agent system without complete system knowledge since control is distributed among the agents which have limited spheres of influence.

A notable characteristic of an agent is its capacity for autonomous action. The notion of '*autonomous action*' is usually contrasted with the absolute obedience of an object when it is invoked. To implement this ability to decide for itself, an agent in a multi-agent system is endowed with rules governing its behaviour and the ability to accept inputs from its environment, to learn from previous experience and adapt future actions, and to communicate with other agents. The behaviour of a multi-agent system can be more than just the sum of the behaviour of its component agents because the interaction among them can sometimes result in system behaviour which is not explicitly programmed in any of them. This emergent behaviour is largely unpredictable and is of great value to the study of social science problems and systems where human decision making is the dominant feature. However, it may not always be of benefit to engineering problems where predictability and repeatability are important.

The flexibility of an agent technology in operation can be demonstrated by integrating a real production agent system with an agent-based model, using the latter as a testbed. This can be achieved relatively easily when compared with the other paradigms because agents are designed to be loosely coupled and highly cohesive entities which communicate asynchronously by message passing. For the same reasons, an agent-based model can be developed relatively easily from a multi-threaded single process into a distributed multi-process model.

Figure 2-3 shows that ABM is the most versatile of the four major modelling paradigms since it can be used on systems spanning the widest range of abstraction and details. While it appears that ABM alone can be used to model problems of all scales and details, nonetheless, it needs to be borne in mind that the use of the other modelling paradigms may result in simpler and more maintainable models. It is important to match paradigm, problem, modeller, and model user as closely as possible since the benefits of a model are realised post-implementation and that phase of the model lifecycle can be much longer than the implementation phase. Selecting an inappropriate modelling paradigm can be costly.

2.3 Misuse of modelling paradigms

The main difference between a time-driven and an event-driven model is that the state variables in the former change continuously over time (for example, the depth of water in a container) while those in the latter change only at events or at discrete points in time (for example, the length of a queue as determined at the random time people join or leave it).

It should be noted here that numerical quantities in a model can actually vary continuously only in an analogue simulator whereas in a digital computer they are merely perceived as continuous. When a time-driven model is advanced in a digital computer at time steps small enough to enable it to resolve adequately changes to the state variables of interest, the changes may be considered to be continuous to all intents and purposes.

Based on this principle, it is reasonable to view time-driven modelling in a digital computer as a special case of event-driven modelling in that an event can be triggered at each of the fixed time intervals thus causing the model's mathematical equations to be re-evaluated. Although such event-driven continuous modelling is possible, the overheads required to maintain the built-in mechanisms of discrete-event modelling makes this combination computationally inefficient.

Conversely, an event-driven model can be stepped through time at fixed intervals. However, time-driven discrete-event modelling suffers from three significant drawbacks –

- An event cannot be guaranteed to occur always at a time interval boundary thus resulting in a loss of precision.
- If two or more events occur separately within the same time interval, it will not be possible to determine their chronological order. Such a lack of resolution can give rise to unexpected results in instances where the order of events is important.

- The model continues to consume computing time even when nothing is happening between events and in doing so increases the elapsed time of a simulation run.

In view of these inefficiencies and potential sources of errors, practical continuous models tend to be time-driven and discrete-event models tend to be event-driven.

2.4 Model metrics

Since the models studied in this thesis are software entities entirely, it is logical to describe their properties by employing metrics which are used in the field of software engineering. This section will provide an overview of software metrics and then consider some aspects of model characterisation and complexity.

2.4.1 Software metrics

There is a large number of software metrics, both broad and narrow, which may be applied singly or in combination. The better known ones, like Halstead's software science metrics (Halstead, 1977) and McCabe's cyclomatic metric (McCabe, 1976), have been in existence for more than 30 years and have been used with mixed success in large, well controlled software development projects. They provide a quantitative measure to the entities and attributes encountered in the course of the life of a piece of software, from analysis to maintenance. Fenton and Pfleeger (1997) categorize them broadly into three classes and they are associated with –

- ***Processes***. These are software-related activities like analysis, design, specification writing, coding, and testing.
- ***Products***. They are outputs from processes and include entities such as written specifications, program code, and test data.
- ***Resources***. These are the inputs required by the processes. Examples of such entities are personnel, computing hardware and software.

Among the attributes from these three classes, the *'products'* attributes are immediately relevant and useful to the subject of this thesis while metrics from the other two classes are less so. These software product attributes may be further distinguished as –

- ***Internal attributes***. They may be determined purely by examining the product itself. For program code, attributes such as program size and code complexity can be measured without resorting to the execution of the program.
- ***External attributes***. They indicate how the product relates to its environment. In the case of program code, metrics for such software qualities as understandability, modifiability, and testability are pertinent.

It would be immensely useful in practice to be able to predict the external attribute of software quality by measuring the internal attributes like size and complexity. Some studies (Li and Henry, 1993; Rombach, 1987; Rombach, 1990; Wake and Henry, 1988) have confirmed statistically valid links between the quality of maintainability and metrics like program size and complexity. Banker *et al.* (1993) provided robust reinforcement to those comparatively lightweight studies by confirming their results with code complexity and maintenance cost measurements carried out on a real, large, commercial banking software system.

The reason that much effort has been expended on measuring software maintainability is that maintenance costs typically varies between 50% to 65% of overall lifetime costs (Somerville, 2001) and its enhancement and adaptation sub-activities can make up more than 80% of the maintenance effort (Krogstie *et al.*, 2006). Focusing on software maintenance is therefore justified as it is a highly significant activity within the software lifecycle. Moreover, its three sub-activities – understanding, modifying, and testing – are equally applicable in the analysis and design front-end stages of the software lifecycle.

The attributes of size, complexity, and maintainability will be considered in greater detail in Chapter 5.

2.4.2 Model characterisation

Although the performance of models, in terms of elapsed execution time, has been routinely measured it has been applied only to specific implemented models and not to the conceptual models. This metric does not address the issue of model characterisation as it is not directly related to a model property or attribute.

A suggested reason why model characterisation has not made any apparent progress has been put forward by Brooks and Tobias (1996) where the difficulty with defining a model's level of implementational detail and software complexity was highlighted. It is possible to arrive at similar results using a finely detailed model as well as a coarser one. Therefore, in order to compare two models objectively, there is a need to establish first their levels of detail and complexity. Despite the acknowledgement that there is widespread use of modelling in a large number of technological disciplines, there has been almost no work carried out to date in characterising simulation models.

Intuition suggests that the more detailed a model is, the greater will its fidelity be and therefore the more accurate the results it will yield. In general, this linear relationship will hold true as long as factors which influence a model's outputs, e.g. the accumulation of approximated stochastic inputs over a model's simulated lifetime, are not overwhelming. This effect is illustrated in Figure 2-7 and is from a study by Costanza and Sklar (1985), cited in Fulton *et al* (2003), where *'articulation'* (or model complexity and scope) bears a non-linear relationship to *'effectiveness'* (or articulation and accuracy combined). Figure 2-7 shows that effectiveness quickly reaches an optimum with increasing model articulation. It should be noted from that even though data is sparse at the high end of the *'articulation*' axis, thus calling into question the validity of the *'effectiveness frontier'*, the shape of the curve is to be expected because the effect of cumulative computation errors will tend to make a large, complex model less accurate. Other similar sensitivity studies by Stockle (1992) found only a negligible decrease in accuracy with model simplification while Halfon (1983a) and Halfon (1983b) reported a diminishing return in accuracy when model detail was increased. The large variation in results from these studies may indicate the existence of one or more unknown but significant factors which have not been taken into account.

2.4.3 Model complexity

Gell-Mann (1995) states that, both in software and in general, all measures of complexity are context-dependent or even subjective to some extent. This is because the representation of an entity depends on, for example, the level of detail, and the assumption of previous knowledge and understanding. In the same vein, but in the context of software development, Brooks (1987) argues that complexity is an essential (or inherent) and not an accidental (or incidental) property. The consequence is that, in software projects – and modelling is almost wholly software – technical problems and those associated with communication and management, will inevitably arise because it will be difficult to obtain a clear overview. Hence, *'loose ends'* such as errors and uncertainties in system requirements can propagate quickly and widely throughout the body of software and their root causes will be difficult to find and control.

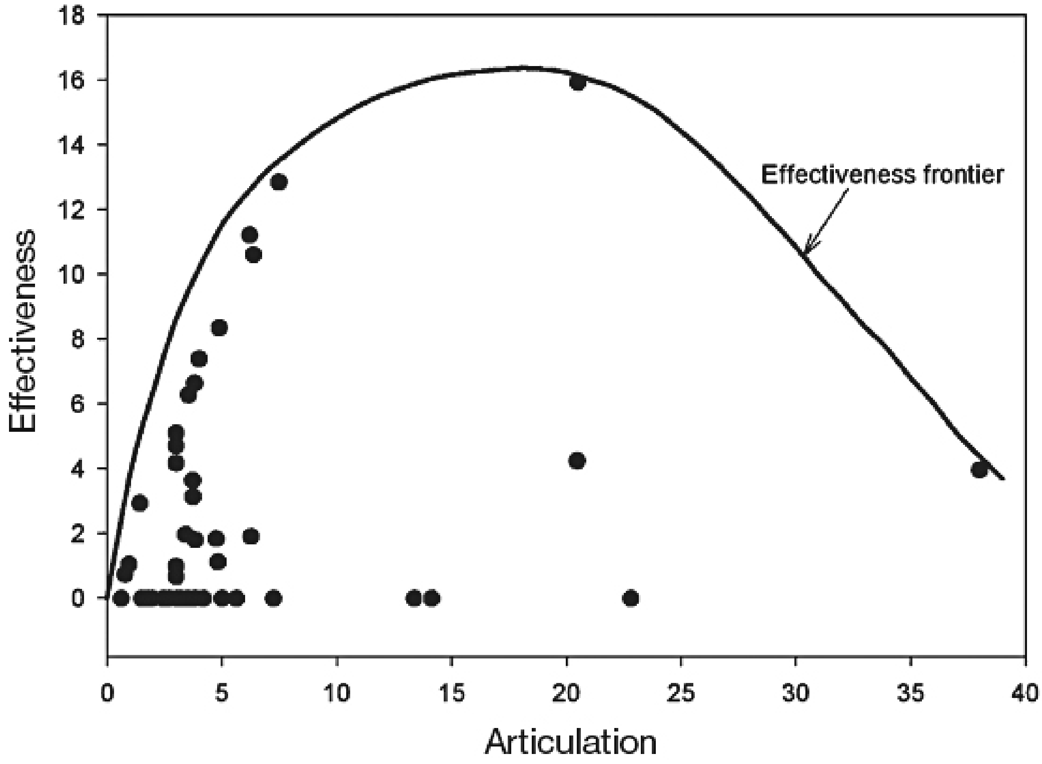


Figure 2-7: Relationship between model complexity and accuracy (Fulton *et al.*, 2003)

Brooks also observed that, in addition to this essential complexity, most of the complexity encountered in a piece of software is arbitrary in nature since the software does not exist alone but must conform to human and system constraints and these constraints can, and do, evolve over the useful life of the software. This effect was also noted by Belady and Lehman (1985) when they postulated the three laws of program evolution, which are continuing change, increasing unstructuredness, and statistically smooth growth. As a model undergoes change during normal post-implementation maintenance, it becomes increasingly disordered, and hence more complex, unless specific effort is expended to prevent it. It follows that to be able to measure a model's complexity objectively is a desirable goal.

Brooks and Tobias (1996) noted that, although a software structural complexity metric like McCabe Cyclomatic Number (MCN) has been in existence for a number of years, in practice, the level of detail or level of complexity in a simulation model is assessed qualitatively. In an attempt to estimate the complexity of a model objectively, they suggested that in a discrete-event model, the events and the relationships between events are the equivalents of McCabe's nodes and edges respectively. However, as it is likely that two or more of these event flowgraphs are possible for the same conceptual model, it is necessary to take the lowest MCN to give an objective and quantitative indication of model complexity.

A novel method of measuring software complexity, the α-metric, was proposed by Kokol *et al* (1999). It employed a technique called long-range correlation and is founded on the information entropy inherent in the content of human writing as established earlier by Shannon (1951). The assumption that there is a correlation between a programme's complexity and the entropy of its information content was confirmed by experiment. However, the study did not reach, nor has it since then been continued to, the stage where the nature of the relationship could be established. Compared with the automated means for obtaining MCN for a large program module, this method appears to be more efficient and considerably less dependent on the programming language used. Also, it can be applied equally well to a textual source like a specification in any of the formal specification languages or a compiled program source in any machine language.

More recently, it has been recognised that simulation models have become very complex. This trend has, in part, been driven by the ready availability of high-performance computer hardware as well as a lack of understanding of the real systems (Chwif *et al.*, 2000). However, Chwif *et al* (2000) do not offer any objective criteria to determine when a model becomes too complex to run, enhance, or maintain feasibly. With the demand for greater realism in simulation has come the corresponding need for a quantitative metric to determine when this threshold has been crossed.

Because the evidence of earlier work points to complexity as the underpinning measure for software or model characterisation, it will be considered further in Chapter 5 where software metrics will be set out in greater detail. It may be noted here that an outcome of having a complexity metric is that it provides a degree of objectivity when different models are to be compared.

2.5 Previous work on comparing modelling paradigms

There does not appear to be any journal publication, conference proceeding, or workshop paper presenting either a qualitative or quantitative comparison of agent-based modelling against discrete-event modelling. A search for agent-based modelling revealed a workshop paper (Parunak *et al.*, 1998) comparing it with equation-based modelling, i.e. dynamic systems and Systems Dynamics. Another search for publications for discrete-event modelling uncovered two studies (Brailsford and Hilton, 2000; Morecroft and Robinson, 2006) which were qualitative comparisons against the System Dynamics modelling paradigm. Morecroft and Robinson (2006) stated that although there is a high level of interest in knowing what modelling method best fits a particular type of problem, there is almost a total absence of such comparative studies.

2.6 Summary

A taxonomy of the major modelling paradigms in current use was presented. It is based on the representation of time and state in the simulation model, classifying the paradigms as either time-driven or event-driven.

The four modelling paradigms – dynamic systems, System Dynamics, discrete-event modelling, and agent-based modelling – were outlined showing their main distinguishing features and the level of abstraction and details where they may be employed.

Among numerous requirements when modelling a problem, it is important to use the modelling paradigm which provides the closest conceptual match. Using an inappropriate modelling paradigm can result in inaccurate results as well as computational inefficiencies.

The internal attributes of program size and structural complexity provide convenient quantitative indicators of the external attribute of maintainability. These metrics enable models to be compared objectively.

Comparisons have been carried in the past between models implemented using different paradigms but none has been made between traditional discrete-event and agent-based modelling.

Chapter 3
Traditional Discrete-event Modelling

3.1 Introduction

In this research, which is in the form of a case study (stated previously in Sections 1.3 to 1.5), each item involved needs to be tracked along the supply chain and logistics network. The attributes of these individual objects, such as unique identifier, life, size, and weight, are therefore important. In these problem domains, the model state variables change only when events occur and nothing of consequence occur between adjoining events. In time-driven modelling paradigms like System Dynamics (SD) and dynamic systems, group and bulk properties are paramount and the state variables change continuously. Therefore, in this research, it is more appropriate to use an event-driven modelling paradigm than a time-driven one.

Procedural languages like FORTRAN and C, object-oriented languages like Java and C++, and specialised simulation languages like SIMSCRIPT, SIMULA, and GPSS have been used for many years in discrete-event simulation. A survey of simulation languages is given in Shannon (1977) and more recently in Low *et al* (1999) where descriptions of a number of runtime libraries are also presented.

Manual crafting of program code for a simulation model is very time consuming and prone to errors. A development, both to speed up and improve the quality of model building on the PC desktop, has been the commercial introduction of visual programming environments, e.g. AnyLogic™ (XJTechnologies, 2006), Arena™ (Arena, 2007), and Extend™ (ImagineThat, 2006; Krahl, 2001). The last, in particular, provides a comparatively low-cost yet powerful and versatile environment with an open architecture which can be enhanced by the knowledgeable user.

The body of published literature distinguishes between two types of discrete-event simulation – sequential and distributed (or parallel). Generally, experiments in the former are carried out in a single process executing in one processor while those for the latter are partitioned into multiple processes and executed in parallel over two or more physically separated processors. Distributed modelling continues to be a specialized process which requires a high level of programming skills. Although there is no commercial tool which can deal with all required aspects of distributed modelling, the production of such a model has been aided in recent years by the provision of the High Level Architecture standard framework (described later in Section 3.3.2.3).

Since the event list is essential to the operation of a discrete-event model, this chapter gives a description of it together with details of the techniques which have been employed to make it run efficiently under different conditions. That is followed by a consideration of distributed modelling as well as the inherent problem of time synchronization which has been eased by the development of a standard framework. The melding of time-driven systems with event-driven systems and the use of artificial intelligence techniques in discrete-event modelling conclude the chapter.

3.2 The event list in traditional discrete-event modelling

In this section, the event list is considered in some detail for the reason that it is the data structure which underpins discrete-event modelling.

It is fundamental in both sequential and distributed discrete-event simulation that a set of future or pending events is processed in a valid and consistent order. This is handled by a data structure called an event list which stores an organised set of events and enables new events to be inserted at the correct locations. In a simple model executed in a single processor, the event list may just be a linear linked list holding a small time-stamped set of pending events in increasing chronological order. A considerable amount of research has been carried out in the last four decades in pursuit of a general solution for the efficient operation of the pending event set (Marín, 1997), or priority queue (PQ) (Knuth, 1973) where the priority is the event time-stamp in this instance.

As it is possible for two or more events to be scheduled to occur simultaneously in the same computer process using a common event list, the execution of these events needs to be serialised. One method is to select any of these events at random with equal probability to reduce the effect of serialisation over a simulation run. Where event order is not important, they may be executed in any order. The latter method is faster as it incurs a lower overhead than the former when deciding where to insert the entry in the event list.

In a large and complex model which has to process many individual items, the cascading of generated events during a simulation run can quickly result in a very large set of pending events. As the order of arrival of events at the event list may not be in chronological order of simulation time, the events have to be inserted into the data structure after the correct time locations of the event list have been found. For a large list, the computation time needed to manage this data structure alone can take up a significant proportion of total computation time. As the time taken to insert an event in a simple linear linked list is proportional to the length of the list, i.e. $O(n)$ where n is the number of pending events, if a linear search is used to determine the point of insertion, alternative techniques have been devised to make it speedier and more efficient.

These PQ techniques are categorised as either tree-based or list-based. In Marín's (1997) empirical comparison of some of these algorithms, which is an extension of an earlier study (Jones, 1986), he lists 13 tree-based and 9 list-based PQs. Notably, Jones' empirical comparison did not include calendar queues since it predates their appearance in discrete-event modelling. Also, Marín demonstrated that for a large pending event set, i.e. for $n \geq 10^4$, calendar queues outperformed tree-based PQs when measured as average, or amortized, time taken to insert an event for a hold operation. This outperformance is to be expected since calendar queues are specialised data structures which are tailored for managing the pending event sets of discrete-event models.

A PQ may be conceptually simple but the algorithm needed to implement it may be large and may involve storage and computation overheads. For instance, the Calendar queue (Brown, 1988) has relatively short and efficient multiple lists but they require extra space to be reserved for list expansion and these lists have to be resized should their upper size limits be exceeded. Although resizing occurs infrequently, it is computationally very expensive and the algorithm carries code which is not used frequently.

The numerical values in Table 3-1 represent the relative average insertion time,

$$T_{insert} = \frac{T_{PQinsert}}{T_{CBTinsert}} \tag{3-1}$$

where $T_{PQinsert}$ is the average insertion time for any of the PQs and $T_{CBTinsert}$ is the average insertion time for the PQ implemented as a complete binary tree (CBT).

Marín (1997) used the idealised case of a CBT as the basis for comparing performance. A complete binary tree is a binary tree (one in which each node has two children) where all levels are completely filled except for the final level where the nodes are all to the left, filling up consecutive positions. Each technique was executed using the classic hold model with different event access patterns simulated by different probability distributions. It may be noted in Table 3-1 that the different tree and list techniques are sensitive to the type of event distribution to different degrees. In particular, among the four types of queue, the Calendar queue performed relatively badly for the triangular distribution when the pending event set is large, i.e. $n \geq 10^4$.

Table 3-1: Relative amortized run times,T_{insert}, for different probability distributions (Marín, 1997)

Event distribution	Exponential		Uniform		Triangular		Bimodal		Biased	
Queue type	n=10	n=10^4	n=10	n=10^4	n=10	n=10^4	n=10	n=10^4	n=10	n=10^4
Complete binary tree	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Skew heap	1.56	2.69	1.66	2.78	1.72	2.84	1.40	2.50	1.71	2.88
Splay tree	1.71	2.70	1.73	2.60	1.69	2.53	1.46	2.28	1.55	16.89
Calendar queue	2.24	0.78	2.19	0.75	2.20	1.86	2.53	0.79	3.37	1.12

Describing all the PQ techniques in Marín (1997) would involve too lengthy a digression but three of the more prominent ones among them, i.e. the skew heap, the splay tree, and the calendar queue, are considered further in Section 3.3.1 together with two other PQs, the lazy queue and the event horizon.

3.3 Types of discrete-event modelling

Discrete-event modelling may be categorised as sequential, where a model is implemented as a single process and executed entirely within a single processor, or distributed, where a model is divided into sub-models and executed as individual processes in physically separated processors.

3.3.1 Sequential discrete-event simulation

Other than the development of the sequential discrete-event model-building environment to speed up and make model creation easier, the progress of sequential discrete-event modelling has largely concentrated on the development of the PQ to make it computationally as efficient as possible. In sequential discrete-event simulation, all activities in a model are serialised by way of a single PQ thus making it the most critical part of the model.

Although five PQs are presented in the following sections under the heading of sequential discrete-event simulation, the skew heap (Section 3.3.1.1) and the lazy queue (Section 3.3.1.4) can be adapted for use in the parallel simulation environment (Rönngren and Ayani, 1997). It is done by granting only one of the parallel processes exclusive access to the PQ at any time by locking and unlocking the appropriate parts of the data structure.

3.3.1.1 The skew heap

The skew heap is described by its originators, Sleator and Tarjan (1986), as a self-adjusting form of the heap data structure which Knuth (1973) calls a priority queue. A heap is an ordered binary search tree where the node with the minimum value is at the root. A heuristically self-adjusting tree is not subject to structural constraints such as tree height balance conditions for a balanced tree. This has the advantage of avoiding the time and space overheads required to maintain the structural constraints. A balanced tree is one where all the leaf nodes are at the same level. Moreover, the depth of a balanced tree is shallower when compared with an unbalanced tree, thus resulting in shorter search times to locate nodes for deletion or places where new nodes are to be inserted.

Inserting an entry into the heap involves the central operation of combining or melding the single-value heap into the original multi-value heap such that the new entry ends up at the appropriate heap location. In the worst case, performance of the skew heap is bounded by $O(\log_2 n)$ time (Sleator and Tarjan, 1986). Because of its economy and speed, this technique is commonly used not only in sequential discrete-event modelling but also in combination with the Time Warp mechanism (Fujimoto, 1990) used in parallel and distributed modelling because it tends to minimise the amount of rollbacks (described later in Section 3.3.2).

3.3.1.2 The splay tree

The splay tree is a self-adjusting form of the binary search tree. As frequently used nodes are migrated towards the root of the tree by a sequence of simple tree rotations – an action described as splaying – it is also self-optimizing. When used for the management of the event list in discrete-event modelling, this splaying operation, which consists only of the simple tree rotation heuristic, tends to restructure the tree after a node insertion or deletion so as to minimise tree depth. It can reduce the worst-case amortized search time to $O(\log_2 n)$ where n is the number of events (Sleator and Tarjan, 1985). The efficiency of this PQ is the same as that of a balanced tree (Knuth, 1973) while avoiding the space overheads and algorithmic complexity needed to maintain the constraints of a balanced tree.

Marin (1997) noted that most of the tree-based methods have worst case search times of $O(\log_2 n)$.

3.3.1.3 The calendar queue

The calendar queue (Brown, 1988) is a multi-list data structure modelled on a calendar and used much in the way a human would use a calendar, i.e. writing to or erasing from the appropriate day with each day containing a short, sorted linked list. When a new event is generated, the day or sub-list to which it belongs is calculated and the event is inserted into that linked list. The insertion point average search time of this multi-list method is of the order of $O(1)$. The benefit of a constant level of performance is considerable and is especially noticeable when the pending event set is large (see Table 3-1). However, its advantage does not extend to all sizes of the pending event set since PQs implemented as single trees or linked lists perform better for small event sets where $n \leq 10$.

The year is a rolling circular data structure which is initially sized such that about 75% of the events fall within it. The length of a day is set to ensure that the size of its linked list does not degrade the average performance for the year. To ensure good performance, Rönngren and Ayani (1997) mentioned that the average length of the linked lists should be about two elements long. As the PQ grows and shrinks during a simulation run, the day and year lengths have to be adjusted in size to maintain its efficiency. When the lower and upper size thresholds are reached the day lengths are halved or doubled respectively. Similarly, the year length is adjusted to ensure that the day lengths are evenly distributed to prevent empty days from occurring. This resizing operation is costly since it is of the order of $O(n)$ where n is the number of elements in the PQ.

Multi-list PQs tend to have dedicated storage to handle the overflow of event data but the calendar queue manages this problem elegantly by allowing events which extend more than a year into the future to wrap around thus spreading the overflow amongst the day sub-lists.

3.3.1.4 The lazy queue

The lazy queue (Rönngren *et al.*, 1991) is a multi-list PQ with events which are divided into three categories (see Figure 3-1) –

- ***Near future (NF)***. This is a buffer of events which will be executed almost immediately and consists of a link list for enqueue operations and a sorted array to handle dequeue operations.
- ***Far future (FF)***. This contains a number of chronologically ordered but unsorted sub-lists. When the NF sorted array has been completely dequeued, the earliest FF sub-list is sorted using the Quicksort algorithm and transferred to the NF data structure. It is the delaying of

work until the last possible moment which gives this technique its name.

- ***Very far future (VFF)***. This is implemented as a link list and with the passage of time, a part of it is transferred into the FF data structure

Rönngren and Ayani (1997) substituted the link lists in the NF and VFF data structures with skew heaps thus enabling them to operate more efficiently in the worst-case.

The assumption which underpins this PQ is that the largest proportion of new events in a time interval is inserted into the FF data structure. Since this enqueue operation involves just an appending of the event to the appropriate sub-list, the amortized performance is *O*(*1*). A dequeue operation occurs only in the NF data structure and involves taking out the earliest element from a sorted list. This also results in an amortized performance of *O*(*1*). The operation of this PQ will be degraded should the distribution of events be such that the largest proportion falls in the NF and VFF data structures.

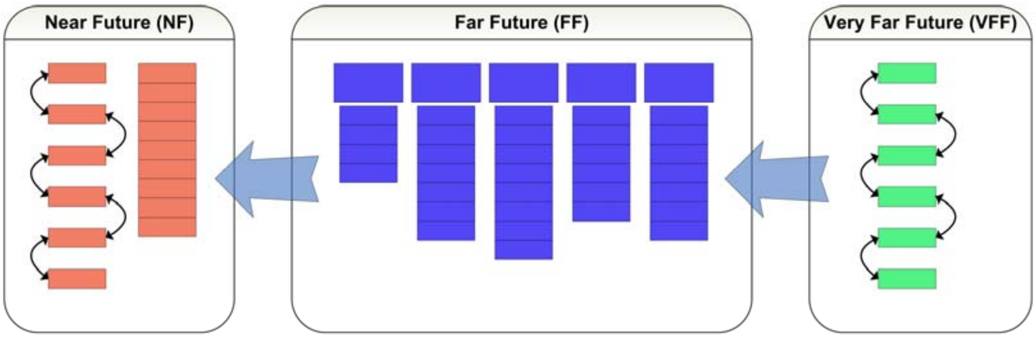


Figure 3-1: Components of the Lazy Queue (Rönngren *et al.*, 1991)

3.3.1.5 The SPEEDES queue and the event horizon

The SPEEDES queue is a two-list technique (Steinman, 1994) which introduces the concept of the '*event-horizon*'. Although it was originally developed for parallel discrete-event simulation it may also be used for sequential simulation.

As illustrated in Figure 3-2, the pending events are stored in the primary event list which is maintained in increasing time-stamped order while the generated events are stored in an unsorted secondary list. The earliest generated event is noted and just before the simulation reaches this event, the secondary list is sorted and merged with the primary list. If the primary list is implemented as a linked list, the amortized cost of inserting the generated events is $O(n)$ where n is the size of the pending event set. However, the SPEEDES queue approach to PQ management can be made more efficient for large pending event sets by substituting the linked list with a variant of the balanced binary tree (the SPEEDES tree), or the heap (the SPEEDES Qheap) (Steinman, 1996).

The event horizon, a term borrowed from physics and astronomy describing the boundary of a black-hole beyond which it is not possible for anything to escape, is marked by the last pending event before the earliest generated event. The boundary of the event horizon is shown as vertical dotted lines in Figure 3-2. Its extent fluctuates from cycle to cycle and as the event sequence within its boundary will not undergo further change, it allows distributed, parallel processes to be synchronised up to that point in time. Hence, the event horizon is the simulation model's lookahead (Fujimoto, 1990) as the events before it are guaranteed to be in chronological order and that no pending event will generate an event earlier than it. For these two reasons, the event horizon is a technique for preventing deadlocks – the situations in which all participating parallel processes cannot safely advance in simulation time because doing so might violate the Principle of Causality thus halting a simulation run. This principle is a tenet of distributed simulation which states that events are caused by earlier ones and so a generated event must always occur after the generating event.

3.3.1.6 Performance of priority queues

The performance of the PQs described in the earlier sections are summarised in Table 3-2. In reflecting real and practical modelling, it is evident from the expected amortized costs for the enqueue and dequeue operations that the multi-list calendar queue and lazy queue are superior to the tree-based skew heap and splay tree. This view is reinforced by the fact that the commercially available modelling tool, Extend™6 (ImagineThat, 2006), employs a variant of the calendar queue in the implementation of discrete-event models.

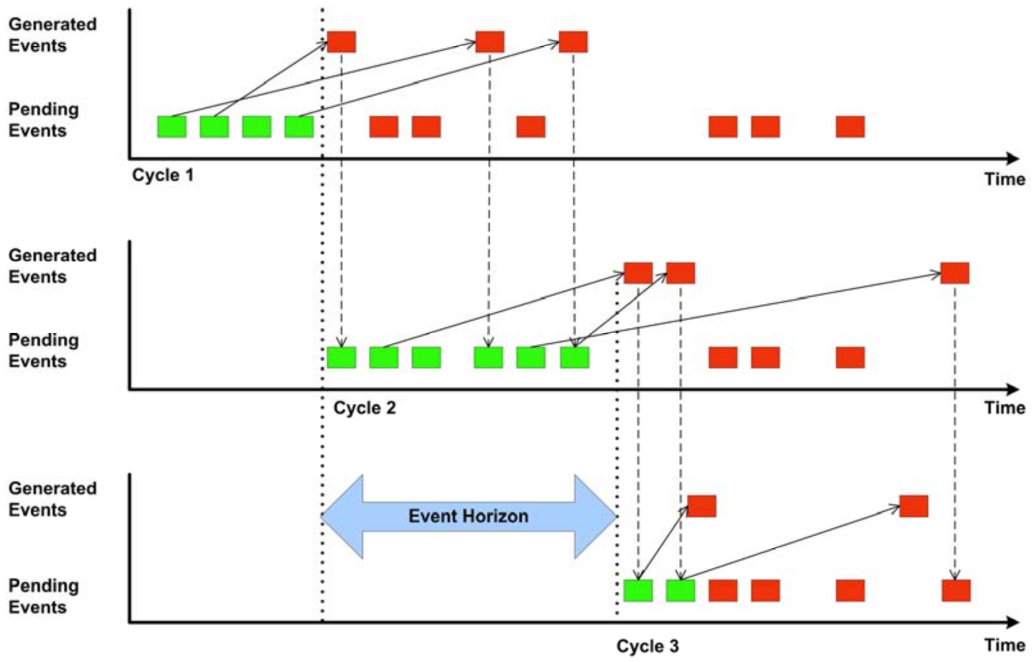


Figure 3-2: The operational principle of the event-horizon using the SPEEDES queue (Steinman, 1994)

Table 3-2: Performance of priority queues (Rönngren and Ayani, 1997)

Operation	Enqueue (amortized)		Enqueue (single operation) Maximum	Dequeue (amortized)		Dequeue (single operation) Maximum
Queue type	Expected	Worst case		Expected	Worst case	
Skew heap	$O(\log_2 n)$	$O(\log_2 n)$	$O(n)$	$O(\log_2 n)$	$O(\log_2 n)$	$O(n)$
Splay tree	$O(\log_2 n)$	$O(\log_2 n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$
Calendar queue	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$
Lazy queue	$O(1)$	$O(n)$	$O(n\log_2 n)$	$O(1)$	$O(n)$	$O(n\log_2 n)$

3.3.2 Distributed discrete-event simulation (DDES)

The nature of the event list in sequential discrete-event simulation demands that events are processed in chronological sequence, one at a time. In practice, despite the use of techniques (mentioned in Section 3.3.1) to speed up processing, sequential discrete-event simulation on one processor has proved to be inadequate for large, detailed models to complete their simulation runs within a reasonable time.

Moreover, in the approach to increasing the performance of such models by partitioning, distributing, and executing them in parallel over a number of separate processors networked together, the event list structure cannot be simply adapted likewise for the distributed processes to work together as a coherent entity. While distributed parallel processing will reduce elapsed run time by some degree for models which are not wholly sequential, it introduces a fundamental problem to this paradigm – the management of simulation time over all the participating processes. It is a problem because in the absence of time management and in the presence of diverse hardware and software all performing at different speeds, there is every possibility that at some point during a simulation a future event will appear to influence a past event. To have the cause occur after the effect makes the simulation meaningless and therefore techniques have to be devised to preserve the Principle of Causality.

K.M. Chandy and R.E. Bryant were attributed with the initial idea of distributed simulation in 1977 (Misra, 1986). It is to be understood here that distributed processes refer to those processes which are executed concurrently, or in parallel, but are spatially separated.

One way of managing DDES is to have the processes all proceed in lockstep to the beat of a global simulation clock. However, this synchronous technique does not exploit fully the benefits of concurrent execution in parallel processing in terms of speed of execution but it provides the means for the interoperation of models or sub-models implemented using diverse modelling packages. Generally, it results in relatively poor performance and sometimes in the loss of fidelity (Fujimoto, 1990). In instances where low model response times are important, this technique is rarely, if ever, used and time synchronisation techniques are most commonly based on asynchronous processes. The methods are traditionally divided into two broad classes which are known as conservative and optimistic time synchronisation (Fujimoto, 1990; Reynolds, 1988).

3.3.2.1 Conservative time synchronisation

The defining characteristic of conservative time synchronisation is that it strictly avoids the execution of any causality error and therefore advances the simulation run only when every event which can affect the event in question has been completely processed. This approach was pioneered by Chandy and Misra (1979). The basic mechanism of this synchronization scheme is the event message, containing the sender's logical time, which is broadcast from one of the parallel processes to the other participating processes. There is a distinct possibility that the messages will not arrive in the same order in all the processes because of the variable delays caused by network topology and the unpredictability of network traffic. Allowing the simulation processes to progress according to the order of the raw message queues will inevitably result in deadlocks.

Reynolds (1988) is of the opinion that there is a wide spectrum of conservative techniques. He lists seven example variants on the Chandy and Misra scheme. These variants have resulted from attempts at avoiding the deadlock problem. A method of achieving this is by the sending of non-event time-stamped messages to determine the smallest timestep to advance the simulation. However, it is possible for these non-event messages to proliferate and overwhelm the model event message population at any time thus resulting in performance degradation.

Another drawback of these conservative protocols is that a programmer must be involved with the details of the protocols in order to achieve good performance (Fujimoto, 1990). This requirement to tune the model code to the synchronization scheme is also likely to result in fragile code and loss of maintainability.

3.3.2.2 Optimistic time synchronisation

What distinguishes optimistic time synchronisation from the conservative approach is its ability to detect and recover from causality errors. In contrast to conservative time synchronisation, it does not determine when it is safe to proceed, but proceeds until a causality error is detected. Instead of advancing by the smallest safe timestep, it advances as far as possible.

Most of the optimistic time synchronisation schemes are variants of the Time Warp mechanism. This is based on the seminal work of Jefferson (1985) on the Virtual Time paradigm which has been developed further by Fujimoto (1989).

In the Time Warp approach, the model states for each distributed process as well as the messages sent and received are saved at intervals. Recovery from a causality error is achieved by rolling back the current model state to a time before the cause of the error. An error is detected if a timestamp earlier than the current simulation time of the process is received. Although computation effort is wasted both in moving the model forward and backward in simulation time, Fujimoto (1990) reports that the detrimental effects due to rollback and thrashing (where incorrect computation is made and rolled back repeatedly in quick succession) seldom occur in practice. Also, Fujimoto observed that where rollback has to be made, the rollback distance tends to be limited because processing always starts with computation of the smaller timestamps and computation far into the future occurs later in the cycle.

While it cannot be denied that this approach exploits parallelism, the necessity of saving the model states in each of the participating distributed processes has the potential to degrade performance severely. Unlike conservative protocols and because of its ability to roll back, there is less of the necessity to tune the model code to the requirements of time synchronization thus resulting in greater maintainability.

The event horizon presented earlier in Section 3.3.1.5 may be considered to be an optimistic synchronisation technique. Unlike the Time Warp approach, it does not have the storage and computation overheads required by the rollback mechanism since it always ensures strict adherence to the Principle of Causality for all events within the event horizon.

3.3.2.3 High-Level Architecture

It has been the norm that DDES time synchronization, by whatever technique, has been implemented in an *ad hoc* fashion. The absence of a standard can, and invariably does, lead to difficulties when attempting to run together multiple models built by different programmers. This is true even if the models employ the same time synchronization technique.

The consequential lack of interoperability and reusability were recognised as a major waste of effort for simulation and modelling activities within the US Department of Defense (Kuhl *et al.*, 1999). The inability to derive more benefit from existing models provided considerable impetus to standardise an architecture for distributed modelling. Work initiated in the early 1990s resulted in the definition of the High-Level Architecture (HLA), a standard for distributed modelling which has been adopted by the IEEE as Standard 1516 (IEEE, 2000). Figure 3-3 gives a concept-level view of HLA showing the central importance of the Runtime Infrastructure (RTI) as it provides all the services needed for the participants or federates to operate together as a coherent unit.

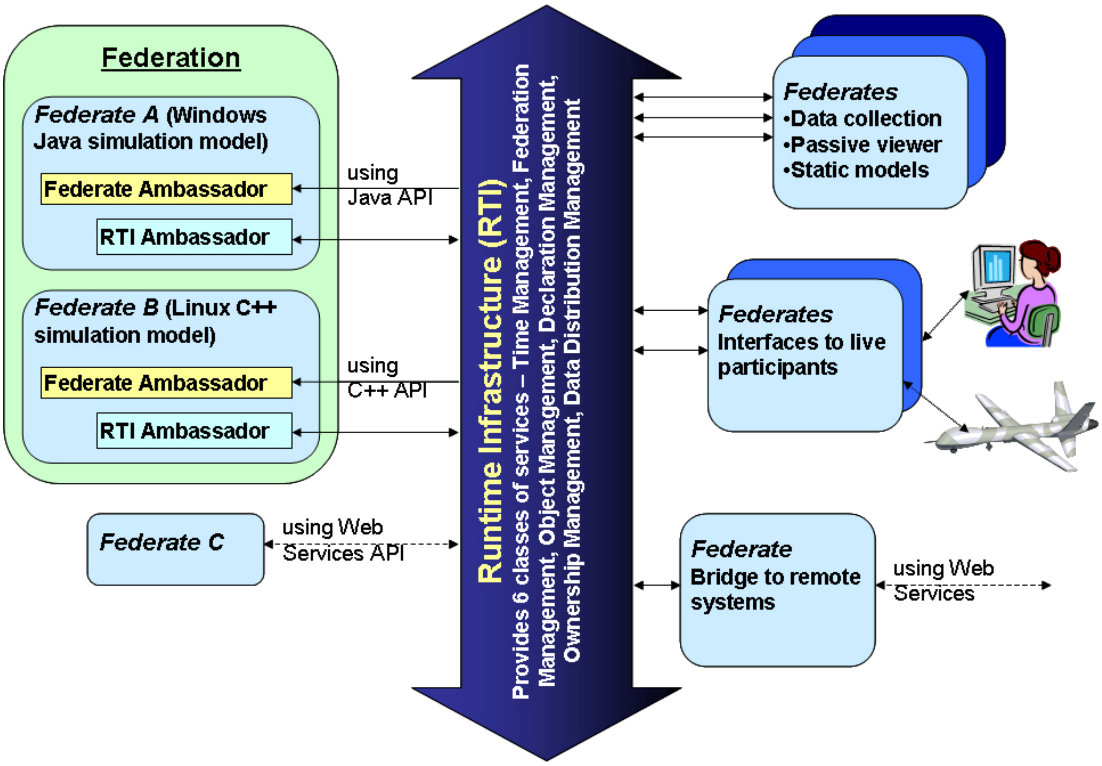

Federation
Federate A (Windows Java simulation model)
Federate Ambassador
RTI Ambassador
using Java API
Federate B (Linux C++ simulation model)
Federate Ambassador
RTI Ambassador
using C++ API
Federate C
using Web Services API
Runtime Infrastructure (RTI)
Provides 6 classes of services – Time Management, Federation Management, Object Management, Declaration Management, Ownership Management, Data Distribution Management
Federates
•Data collection
•Passive viewer
•Static models
Federates
Interfaces to live participants
Federate
Bridge to remote systems
using Web Services

Figure 3-3: Conceptual diagram of the High-Level Architecture

3.3.2.4 Speeding up a model

It has been implied thus far that a model executes as an individual process within a computer processor. A model can be divided into sub-models with each running as a separate process in a single processor under the management of a multi-tasking operating system. A drawback of such an arrangement is that the elapsed time of a simulation run is not reduced although parallel processing appears to be taking place. This is because, in reality, only one process can be executed at any time. Each process is allocated a small time-slice and the processes are switched in quick succession for execution. Switching between processes incurs additional computing effort, or overheads, to store and restore the states for the model and the computer system thus diverting resources from the task of model computation. Therefore, true parallel processing can occur only when the sub-models are executed as processes in physically separate processors or cores.

An alternative is to let the sub-models run as separate threads in a process (Butler and Sandén, 2001). In general, threads of execution contain smaller sequences of computer instructions, share resources like memory and files, and have lower overheads than processes. As such, they may be considered as lightweight processes, and since they are able to switch execution quickly between threads, they can make more efficient use of a processor's clock cycles. Butler and Sandén's experimental measurements for a distributed discrete-event model showed that by increasing the number of processors, performance improved proportionally up to the point when the event list starts to hold back system performance. They demonstrated that, in parallel simulation, threads are an excellent substitute for computer processes.

There is a theoretical limit to speeding up a model containing some code which can be executed in parallel. Amdahl (1967) expressed it as –

$$\textit{Overall Speedup} \quad = \quad \frac{1}{(1-P)+\dfrac{P}{N}} \tag{3-2}$$

The relationship expressed in Equation (3-2) is generally known as Amdahl's Law where *P* is the fraction of the code which can be executed in parallel, (1 – *P*) is the fraction to be executed sequentially, and *N* is number of processors or distributed processes. It may be inferred from the equation that the greater benefit is derived when *P* is made as near unity as possible rather than when *N* is made very large. *P* is unity for the special case where the whole body of code for a model is executed in parallel in different processors. In practice, this can be done for sensitivity or optimization studies where the same code is replicated in different processors and run independently using different parcels of data.

3.3.3 Hybrid continuous/discrete systems

As the detail of a system gets progressively finer and approaches the realm of the natural process, time-driven or continuous modelling becomes more appropriate than event-driven modelling. In a system where the processes are at widely different levels of abstraction, with continuous-variable dynamics at the lowest and logical decision-making at the highest, a hybrid model will be more appropriate than either a wholly continuous or wholly discrete model. The continuous behaviour is specified by a set of differential and algebraic equations while the discontinuous changes are represented by discrete event switching logic.

It should be noted that discrete modelling here is not to be confused with the discrete-event modelling considered in Section 3.3 where an event-driven simulation advances at variable time intervals which are demarcated by events. In contrast, a time-driven hybrid continuous/discrete simulation model advances at appropriately small, fixed timesteps until an event is detected, i.e. when the threshold value of a model state variable is exceeded. The event time is located and the timestep is adjusted so that the simulation can be advanced up to that point precisely. After this point of discontinuity or mode change, a different set of equations may be used and the simulation continues at a fixed timestep as before.

According to Barton and Lee (2002), combined or hybrid discrete/continuous systems are those systems where discrete state and continuous state dynamics interact to such an extent that they have to be analysed simultaneously. Typical examples of discontinuous or stepwise behaviour may be found in electronic controls where the continuous dynamics of a system can change abruptly because of the switching of a diode, or in the operation of a burst-disc to relief dangerously high pressure in a chemical plant.

Also, Barton and Lee (2002) commented that it was widely accepted that almost all engineering models of dynamic systems contain discontinuities such as hard limits, hysteresis, and deadbands, and therefore cannot be handled correctly or accurately with a completely continuous model alone. Some of the reasons for incorrect results and loss of precision were presented earlier in Section 2.3. Therefore, to model real world problems accurately, simulation tools must be able to perform continuous/discrete simulation. In particular, such a tool must feature the ability to handle event times with precision, process runtime differential and algebraic equations, and deal with discontinuous state changes, event iteration, and chattering (Mosterman, 1999).

Interest in simulation using hybrid systems has become more intensive in recent years as the need to model discrete controllers operating on continuous processes has increased. This activity has also been driven by the integration of artificial intelligence in simulation.

3.4 Artificial intelligence in discrete-event modelling

Artificial intelligence (AI) became a distinct stream of research in Computer Science at the Dartmouth Artificial Intelligence Conference in 1956 when John McCarthy coined the term *'artificial intelligence'*. AI now encompasses a host of loosely related disciplines like neural networks, pattern recognition, image processing, natural language and speech processing, robotics, symbolic computation, automated reasoning, expert systems, and autonomous agents.

When embedded in discrete-event models, AI can simulate the functions performed by a cognitive human and so enlarge the scope of simulation, increase its accuracy, and make it appear more realistic. However, Miller *et al* (1992) cautions that the use of AI in simulation should be selective as AI is not a panacea and can grow to dominate a simulation inordinately. Despite this, it should nevertheless be recognized that AI techniques like data dependencies, backward chaining, and abduction can greatly simplify the creation of some simulation models.

Ören (1977) proposed that AI could be used to assist simulation rather than to form an integral part of a simulation model. This proposal was subsequently developed to show how AI could help in model construction by specifying the model components, the interfaces between them, and also the model parameter values (Ören, 1979; Ören, 1986). Since then, a number of general reviews concerning AI in simulation have been published the more detailed and useful amongst which are Shannon (1987) and Tsatsoulis (1990).

Similarities between AI and simulation in terms of knowledge representation, learning, and natural language understanding have been pointed out by Vaucher (1985). Indeed, Rothenberg (1991) noted the symbiotic relationship between AI and simulation when he observed that while models made use of AI techniques, AI has in turn made use of models as sources of internal expertise. Miller *et al* (1992) is of the view that in the same problem domain, AI is more suited to addressing the problem at a coarser structural level (i.e. where the individual processes making up a system are not well understood) while simulation is better suited at the finer end of the spectrum. It appears that the scope of a problem can be extended and addressed by harnessing together the strengths of both disciplines.

It may be recognised at this point that AI techniques are used in simulation in two quite different ways (Tsatsoulis, 1990) –

- ***AI-based*** modelling where AI techniques are embedded and used to model the problem.
- ***AI-assisted*** modelling where AI systems are used to aid model construction, output analysis and presentation, and optimization.

While AI contains numerous disciplines, it is knowledge-based, rule-based, or expert, systems which is the most common technique found in AI-assisted and AI-based modelling until recent years when agent-based modelling has become dominant. The research direction taken over the past decade is evident in the conference papers presented at the Winter Simulation Conference between 1997 and 2007 (WSC, 2007).

An example of an expert system for assisting the human modeller was discussed in de Swaan Arons (1999) where a number of parameterised models were initially stored in a database so that under the guidance of the expert system, the modeller could select the most appropriate model. A fundamental drawback of that system was that due to the limited number of models in the database, it had a very slim chance of matching the modeller’s requirements exactly despite interpolation between, and extrapolation from, existing models. Subsequently, a different approach was adopted by de Swaan Arons and van Asperen (2000) and that was to apply an expert system in the Arena™ simulation tool so that it could support the modeller during model definition. Their goal was to generate the model automatically. In this they had qualified success as they could only implement models using those Arena™ modules which had a high degree of built-in functionality.

A less ambitious but successful application of an expert system in AI-supported simulation was presented in Law and McComas (2003). Here, the fitting and selection of the most suitable model input probability distribution was made by the ExpertFit package, a rule-based system which draws on a large base of knowledge and data accumulated over 25 years. Its maturity and usefulness is evidenced by its inclusion in commercial-off-the-shelf simulation packages like Flexsim™ and SIMPROCESS™.

A recent example of an expert system in AI-based simulation is described in Vakas *et al* (2001). Here the decision-making behaviour of a battlefield commander is simulated by three sets of rules based on fuzzy logic. Combined with the Myers-Briggs personality profiling method, it has been used successfully as a testbed for evaluating US warfare doctrine since human personality can significantly influence the outcome of a battle.

3.5 Summary

The event list or priority queue (PQ) is the data structure which is central to the correct and efficient operation of both sequential and parallel (and distributed) discrete-event modelling. Considerable effort has been expended on list-based and tree-based techniques.

None of the existing PQ techniques offer the most efficient solution to pending event sets of all practical sizes. The algorithmically simple linked list is preferred where the pending event set (PES) is smaller than about 10 items. However, the constant level of performance offered by multi-list PQs like the calendar queue and the lazy queue provide a good general solution for PESs larger than about 10. For this reason, a widely available commercial general purpose modelling package like Extend™6 which is used for building the sequential discrete-event models in this research uses a variant of the calendar queue. Where the sub-lists are likely to be large, tree-based PQs have been incorporated into the multi-list methods to improve performance.

The growing demand for greater realism in modelling has been met by larger models and different modelling approaches. The development of parallel and distributed modelling has resulted in the formalisation of a standard for interoperability – the High Level Architecture. The introduction of hybrid continuous/discrete modelling has extended the useful range of either continuous or discrete modelling on its own.

The overall speedup of a model is mainly dependent on the proportion of model code which can be made to execute in parallel.

Artificial intelligence (AI) techniques have been used as an integral part of the model itself (AI-based modelling) as well as to aid the modelling process (AI-assisted modelling). AI-based modelling has enabled realistic human decision-making to be incorporated into systems of external processes.

Chapter 4
Agent-based Modelling

4.1 Introduction

Although more than 50 years have passed since term Artificial Intelligence (AI) was coined by John McCarthy at the 1956 Dartmouth Summer Research Conference on Artificial Intelligence (Russell and Norvig, 2003), the promise, originally held out by the AI community, of simulating human intelligence (alternatively referred to as general intelligence or strong AI) in a machine remains largely unfulfilled. It is still a long-term goal. Early AI research concentrated on matching human intellectual capacity for general task planning using symbolic reasoning but that was recognized as being too ambitious for that time. There were severe limitations in computing hardware so that it was not possible to produce useful results in a timely manner. Subsequent effort has focussed on sub-fields such as knowledge representation, natural language processing, and machine perception and learning and useful computing techniques have emerged from them. Expert systems, a classic example of which is MYCIN (Buchanan and Shortliffe, 1984), enjoyed some visible commercial success in the 1980s. A standard textbook like Russell and Norvig (2003) gives a detailed and comprehensive treatment of these sub-disciplines.

While AI research narrows down on isolated *fragments* of the artificial intelligence problem, agent technology concentrates on intelligent *systems*. Research in agent technology is an approach which is not in the traditional AI mould since it aims to be lightweight on AI techniques but makes technique integration an essential element in the production of intelligent behaviour. According to Jennings *et al* (1998), the concept of the agent emerged from three closely related areas –

- Artificial intelligence.
- Object-oriented programming and concurrent object-based systems.
- Human-computer interface design.

These elements may be discerned to be present to different degrees in agent systems. Since there is such a spectrum of agents, the opportunity will inevitably arise when it will be difficult to make a distinction between a very rudimentary agent system and a conventional software system.

This chapter first considers what constitutes an agent because it is important to form, at least, an opinion of one of the fundamentals of this subject. Two classes of agents from opposing ends of the agent spectrum are then presented and they are followed by a consideration of the use of agent technology in simulation modelling. As agents are capable of communicating with each other to achieve their goals without human intervention, complex adaptive systems and the emergent behaviour of multi-agent systems are also considered. Seeing the versatility of this technology, it is natural to make use of it whenever possible and so this chapter is concluded with a section on the use and misuse of agents.

4.2 Definition of an agent

As this is a relatively new and fast-developing field, it is important to have a view of its base constituent – the agent. In the past decade and a half, agreement in the research community with regard to the notion of the agent has become less vague and the boundary less elastic.

To emphasise the early lack of a consensual definition, Franklin and Graesser (1996) suggested a taxonomy of 11 alternatives from various sources in an attempt to arrive at a definition which is not so lax or so restrictive as to render it unworkable. They accepted the impossibility of achieving a sharp-edged definition since mathematical concepts provide sharp definitions but real world categorization is almost invariably fuzzy. However, to endow their definition with some rigour, they proposed a '*mathematical style definition*' of an agent as –

> *An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in*

> *pursuit of its own agenda and so as to effect what it senses in the future.*(Franklin and Graesser, 1996)

This definition contains two elements which are widely acknowledged within the agent research community as essential for agenthood, i.e. its situatedness in an environment, and its capacity for autonomous, goal-directed action. While the authors felt that they succeeded in distinguishing between an agent and a program, they concluded that this basic definition encompassed too much and so they proposed dividing agents into smaller sub-categories using properties such as the ability to be mobile and to communicate and learn. This reflects the reality of a wide variety of practical agent systems where agents are not all monochrome but are spread over a broad spectrum.

Comparing an agent with a conventional computer program implies that agent code itself possesses properties which make it inherently different from conventional code. It may be argued that at the implementational level, a body of program code will appear similar to another and the requirement to distinguish agent from non-agent code is probably untenable. To resolve this may require either a widening of the scope beyond computer code to include software analysis and design philosophy or it may have to be elevated to a viewpoint at the conceptual level.

At the conceptual level, a definition which has been increasingly adopted by researchers is Jennings' (2000) précis of Wooldridge's (1997) description of agent-based systems –

> *An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.*

It contains three concepts, all of which need to be considered together to satisfy the notion of agenthood, i.e. situatedness, flexibility, and autonomy. An agent's ability to interact with its environment sets it apart from an AI system which has no need of an environment. Its capacity for autonomous action enables it to have control over its own actions and to function without direct human intervention. It achieves flexibility by being responsive to changes in its environment, pro-active in its goal-directed actions, and social in interacting with other agents to reach the objective it has been set.

4.3 Categorisation of agents

As in the categorisation of simulation models (see Section 2.2), agents may be classified in different ways. One method commonly encountered in research literature is to describe an agent according to its function, for instance a shipping agent or a sales agent. This categorisation is deficient since there can be a very large number of such agents and so a usable taxonomy is likely to be cumbersome and may be ambiguous in places. In the context of this thesis, it is more practical to divide agents broadly according to their architecture with deliberative agents and reactive agents (Wooldridge and Jennings, 1995), respectively at the stronger and weaker ends of the spectrum of the notion of agency.

4.3.1 Deliberative agents

A deliberative agent presents to the outside world the impression of rational thinking in its ability to plan how a problem may be solved. It does this by searching through a store of behaviours, proving the validity of its plan, managing its internal representation of its world, and predicting the effects of its actions. Hence, it is able to generate and select an alternative course of action without human intervention. In this strong notion of agency which is closely analogous to human decision-making, logical reasoning is carried out by pattern matching and symbolic manipulation and in the presence of a symbolic model of the agent's world (Wooldridge and Jennings, 1995). It overlaps very largely with symbolic AI and therefore suffers from the same, notable, and persisting problems in –

- The process of translating of the real world into an accurate, adequate symbolic description – the transduction problem.
- The representing of real world entities and processes in symbolic form and getting the agents to reason with the information – the representation/reasoning problem.

Both these difficulties are exacerbated by the seeming inability of existing symbolic AI techniques to produce the required outputs within a useful timeframe. Moreover, the practical viability of the symbolic AI approach itself was put into doubt when Chapman (1987) presented theoretical results which indicated its unusability in any time-constrained system. For this reason, research here has diminished as the effort increased in the search for more workable and efficient alternatives.

However, deliberative agents are beneficial where the rate of change in the environment of a problem is relatively low and complete knowledge of the problem may not be available. In practice, human systems are dynamic and normally not deterministic, and where the penalty of incorrect action is high, getting the correct response which may not necessarily be optimal can override other considerations.

4.3.2 Reactive agents

Where deliberative agents give the illusion of rational thinking, reactive agents seem to behave reflexively by retrieving an explicitly pre-programmed behaviour very quickly. The basic architecture of a reactive agent consists of four parts, which are the agent environment, the input (or percept), the agent itself, and the output (or action). The agent senses an input from its task environment, maps the input to the output, and effects a change to its environment. Typically, the transforming of percept to action is achieved by a lookup which may be in the form of a table, a coherent set of condition-action (or '*if-then*') rules, or sometimes a neural network. An example of a simple hardware implementation of a reactive agent is a thermostat but the more sophisticated ones are control systems for mobile robots where real-time response is a prerequisite. Historically, the development of techniques for real-time AI was motivated by the relative failure of deliberative methods to deliver results which were timely and has been most visible in the field of robotics.

A significant advance in this class of agents is the introduction of the subsumption architecture pioneered by Rodney Brooks (1986). Instead of decomposing a robot control problem in the traditional manner – as in the sequence of perception, modelling, planning, task execution, and motor control – it is sliced into levels of competence which operate asynchronously. The lowest level of competence, consisting of a class of valid behaviours, provides the fastest response to the most immediate task. The next higher level of competence includes, or subsumes, the competence of the previous level as a subset of its own behaviour. It is possible for complex behaviour to result by using such an incremental control structure. This may best be illustrated by the example shown in Table 4-1 where the highest level of competence is shown as possessing the ability to be flexible in pursuit of its goal. While this may be true in theory, it is difficult to achieve in practice since the reactive architecture has the drawback of having to specify a problem completely at design time. Its minimal learning capability can sometimes leave it with the inability to find a way out of a problem for which it has not been explicitly programmed.

Table 4-1: Levels of competence for a mobile robot using the reactive subsumption architecture (Brooks, 1986)

Level	Competence
0	Avoid contact with objects (whether the objects move or are stationary).
1	Wander aimlessly around without hitting things.
2	‘Explore’ the world by seeing places in the distance that look reachable and heading for them.
3	Build a map of the environment and plan routes from one place to another.
4	Notice changes in the ‘static’ environment.
5	Reason about the world in terms of identifiable objects and perform tasks related to certain objects.
6	Formulate and execute plans that involve changing the state of the world in some desirable way.
7	Reason about the behaviour of objects in the world and modify plans accordingly.

Brooks’ purely reactive subsumption architecture was extended by Matarić (1992) into a behaviour-based one. Where the former has simple lookups for mapping between its input and output, an agent implemented using a behaviour-based architecture has a limited internal representation of the world in which it resides and it also contains more sophisticated algorithms which enable activities such as learning, navigation, and path finding to be performed. In comparison with purely reactive agents, behaviour-based agents exhibit greater flexibility because of its ability to learn by storing state information dynamically (Matarić, 1997). In view of this, emergent, or unprogrammed, group behaviour in groups of homogeneous and heterogeneous behaviour-based agents can be of a higher level. The behaviour-based agent is based on reactive architecture but in terms of agency, it lies in between a purely reactive and a purely deliberative agent.

4.3.3 Hybrid deliberative-reactive agents

On the one hand, a deliberative agent is relatively sophisticated in the AI sense but suffers from the inability to respond within a useful timeframe to changes in its environment. It has a decision cycle time lasting between minutes and hours. On the other hand, a reactive agent can respond in real-time (it has a decision cycle time of less than a millisecond) to changes in its environment but lacks the sophisticated intelligence of the deliberative agent. A hybrid deliberative-reactive agent exploits the benefits offered separately by a deliberative agent and a reactive agent and offers none of their drawbacks.

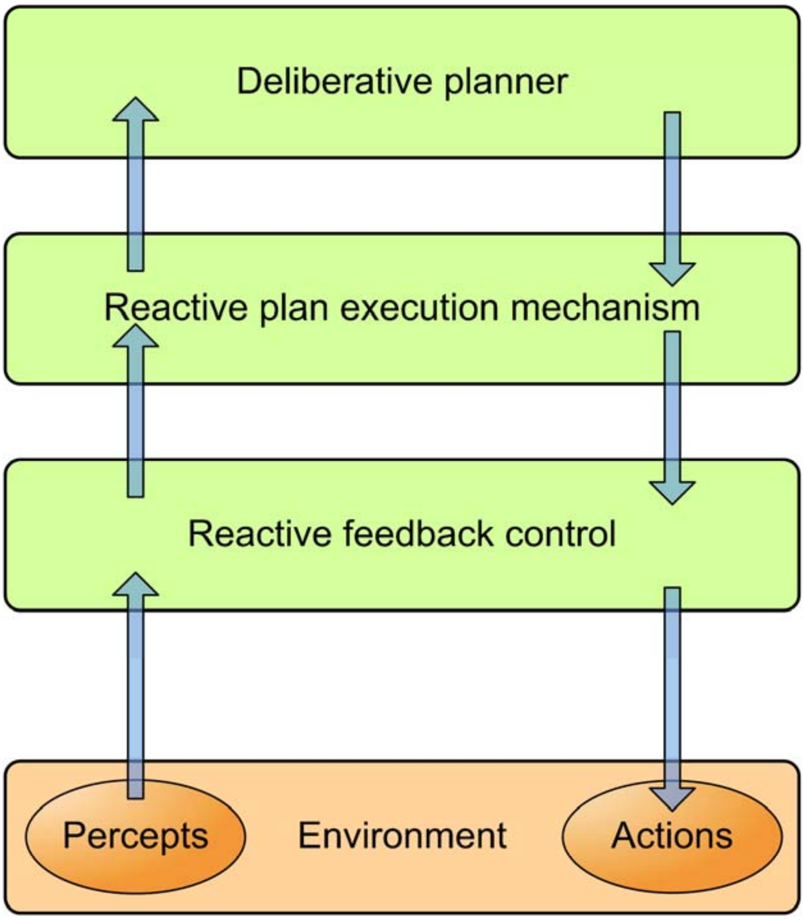


Figure 4-1: Hybrid deliberative-reactive architecture (Gat, 1992)

As in the subsumption architecture, the hybrid architecture is partitioned into layers but differently. A layout which has proved to be successful is shown in Figure 4-1 (Gat, 1998). In the field of robotics, this multi-layer approach has been implemented as the SSS architecture (Connell, 1992) and the ATLANTIS architecture (Gat, 1992).

Data inputs from the environment are acted upon very quickly by software in the '*Reactive feedback control*' layer which contains primitives for interacting with the environment. As the processed data is passed up to the '*Reactive plan execution mechanism*' layer, it is processed further and a part of its output forms the higher-level input into the layer below it. The processed data may be in the form of input parameters to the primitive in the '*Reactive feedback control*' layer selected by the plan execution mechanism. Finally, the '*Deliberative planner*' accepts inputs from the layer below and, in return, provides a strategic execution plan for the plan execution mechanism.

Simulation models can also utilize the principles of hybrid architecture in which the reactive layer provides short-range tactics for immediate use while the deliberative layer provides the long-range strategies. An implemented example which exploits this architecture is a large-scale, complex urban traffic control system where the decisions made by the low-level agents are mediated by the high-level agents (Choy *et al.*, 2003). The simulation results showed that such a system can reduce vehicle stoppage times considerably.

4.4 Agent-based models

Figure 4-2 shows the result of a recent evaluation of agent-related technology by its practitioners (Luck *et al.*, 2005) and is displayed in the now-familiar form of the Gartner Hype Cycle. This is a convenient graphical representation which sets out the five phases, marked out along the horizontal axis, through which new technology is considered to pass through to reach maturity. It provides a snapshot of the relative maturity, and risk to investment, of a range of technologies as each of them progresses through the phases at different speeds. It is clear from the figure that of the agent-related technologies, agent-based simulation has overcome the barrier of initial over-expectation (or hype) and has achieved the stability where its benefits and practical applications are beginning to be better understood. Agent technology used for simulation modelling has a comparatively wide degree of acceptance and may reasonably be viewed as poised for greater commercial exploitation.

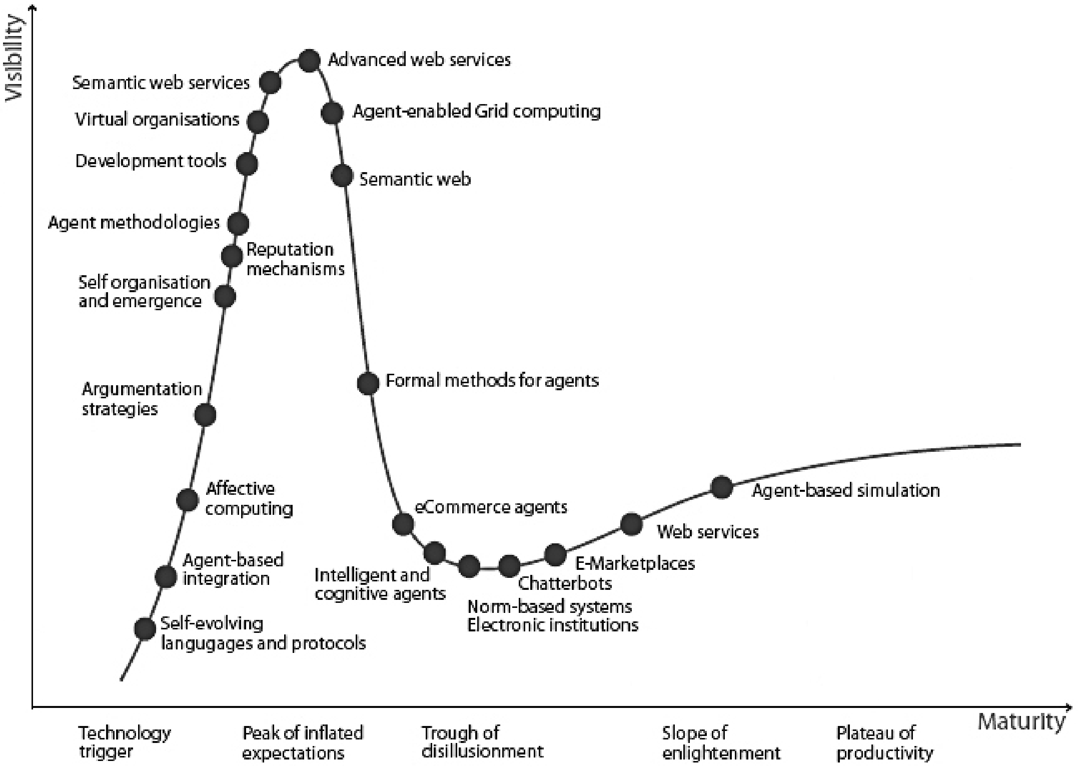


Figure 4-2: Assessment of agent technology readiness (Luck *et al.*, 2005)

Agent technology has begun to emerge from the research laboratories and has been successfully implemented in real-world systems, for instance, in the control systems for manufacturing engine cylinder heads at DaimlerChrysler (Bussmann *et al.*, 2004), and in simulation models for military training and decision support (Cioppa *et al.*, 2004; Martin, 1999). Even so, implementing a multi-agent system is a complex task and it is still a new technology which, in the main, lacks the software tools and trained human resources needed to engage the marketplace (Luck *et al.*, 2005). This assertion is evident in the seven brief case studies of agent systems for production use in manufacturing, logistics, training, and energy production and distribution presented in Belecheanu *et al.* (2006). Taylor *et al.* (2005) also highlighted the wide gulf which exists between "*the promising world of agents and the uncompromising world of the enterprise*". From these publications of agent proponents, it appears that the technology remains confined to a niche within software engineering in the commercial world.

The recommended method of evaluating a technology's maturity is to assess it against the Technology Readiness Level (TRL) scale originally developed by

NASA in the 1980s. Graettinger *et al.* (2002) and MoD (2007) both contain an equivalent TRL scale adapted for software (see Table A-1 in Appendix A). When assessed against this scale, the example systems cited earlier would be rated nine out of the nine-point scale, indicating that the technology has been proven through successful real world deployment. The software systems described in Belecheanu *et al.* (2006) and Bussmann *et al.* (2004) satisfy TRL9, i.e. "*Actual system proven through successful mission operations*." (Graettinger *et al.*, 2002). Nevertheless, commercially-ready agent-based systems are by no means in widespread use.

Today there is a developing consensus for the definition of an agent (Jennings, 2000) and all are agreed that an agent should work autonomously within its environment to achieve the goal it has been set (Franklin and Graesser, 1996). These attributes makes the agent paradigm potentially very well suited to discrete-event simulation of complex problems especially where high-level logical reasoning is predominant and where the information to specify the problem completely may not all be available.

It is reasonable in a model where agent technology is employed, that two or more agents with different knowledge and emphasis be made to collaborate as a coherent functional group. Figure 4-3 shows how such a multi-agent system may be organized at its simplest level. The agents are highly coherent modules and a number of them with related functions may be networked together in a loose cluster with each agent limited in its view and influence within its problem domain. There is no identifiable central control of the group as this function is distributed amongst the agents, embedded within each is its individual limited set of control rules. Their network topology is usually pre-determined and they communicate their requests and intentions with each other by message passing. It may be noted at this point that because agents communicate in this manner, they are more naturally suited to distributed simulation than traditional distributed processes which tend to be constrained by strong, inflexible coupling considered necessary for fast and predictable execution.

Parunak *et al* (1998) concur that in an agent-based simulation, agents '*correspond one-to-one with the individuals being modelled, and their behaviors are analogs of the real behaviors*'. This aspect of agent technology has been developed further by Kendall (Kendall, 2000; Kendall, 2001) where human roles form the basis of multi-agent system analysis, design, and implementation. It is also a concept integral to the Gaia methodology (Wooldridge *et al.*, 1999) which starts from the premise of an organisation. Such an entity is made up of roles, their relationships to one another, and patterns of their interactions. In particular, agent roles are considered to be similar to offices or positions with permissions, responsibilities, and rights attached. Similarly, the Australian AI Institute's methodology (Kinny *et al.*, 1996) based on the belief-desire-intention technology has the role as its basic unit of abstraction. Many examples of such role-centred agent systems can be found in the papers presented at the Winter Simulation Conference (WSC, 2007) in recent years.

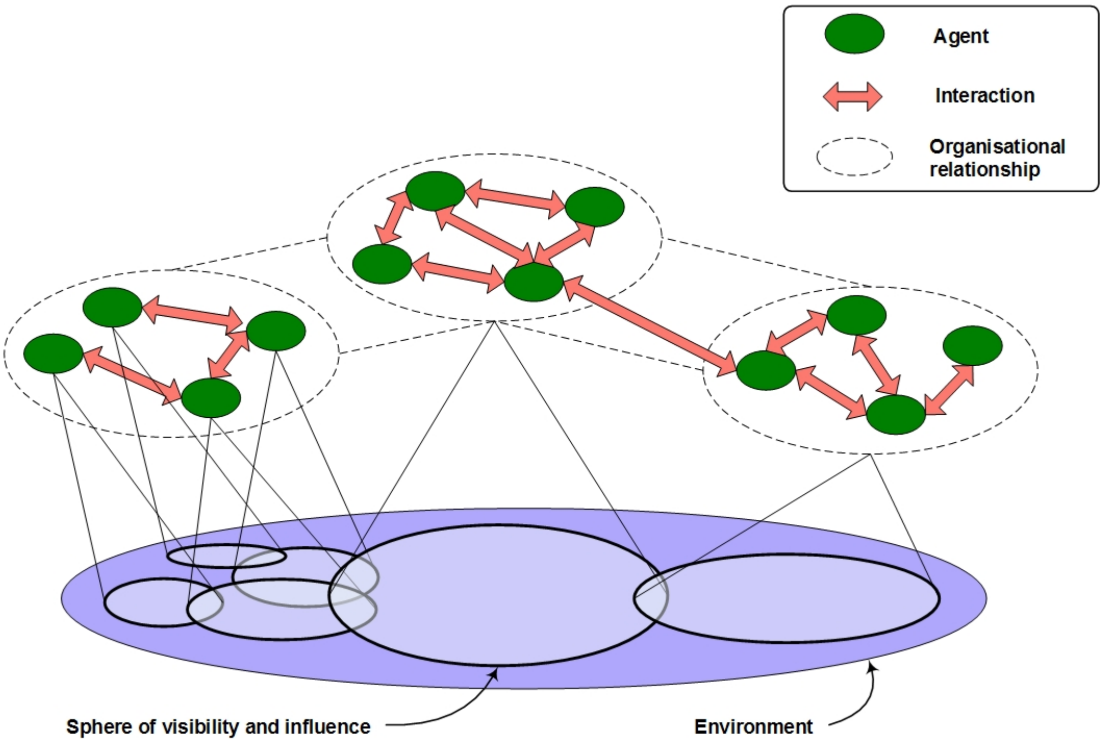

Agent
Interaction
Organisational relationship
Sphere of visibility and influence
Environment

Figure 4-3: Canonical view of an agent system (Jennings, 2000)

An advantage agent-based simulation holds over traditional discrete-event simulation is its ability to emulate human individual and group behaviour. Although discrete-event simulations have included expert systems to model human behaviour (Vakas *et al.*, 2001) they tend to be monolithic and can be inflexible in operation, difficult to maintain, and are only as good as the rules contained in their knowledge-bases. The social sciences have adopted agent-based simulation for investigating social phenomena chiefly because of its ease of use (Davidsson, 2002) and there are numerous implementations of agent-based simulations which include the model of human behaviour in the decision-making loop. For the same reasons, this simulation paradigm has made it easier to study the group behaviour of a large number of entities which interact with each other as well as their dynamically changing environment, e.g. micro-robots (Dudenhoeffer and Jones, 2000).

4.4.1 Analysis and design

As mentioned earlier in Jennings *et al.* (1998), a field which contributed substantially to the development of the agent concept was OO programming and concurrent object-based systems. Its influence is evident in the methods which have been employed in the analysis and design of agent systems to date. They either extend or adapt existing OO development methodology to take agents into consideration. Among these agent-oriented methodologies, the more conspicuous ones are Gaia (Wooldridge *et al.*, 1999), the AAII methodology (Kinny *et al.*, 1996), and Agent UML (Odell *et al.*, 2000). The evolution of agent-oriented software engineering is still at an early stage and it is uncertain how these methodologies will develop.

The Gaia methodology enables a designer to start with an abstract concept and work progressively towards a concrete realisation. For example, starting with the abstract concept of the organization, it is broken down into roles which are further defined by the four attributes of responsibilities, permissions, activities, and protocols. A role then has to be instantiated with an identifiable individual who may have more than one role and may not remain in the same role throughout the life of the organization.

UML is the *de facto* standard for OO modelling and Agent UML extends UML to enable the modelling of agent systems. The extensions are provided to help with the expression of both the interaction of concurrent threads in multi-agent systems, and the notion of the role where an agent may play more than one role. It is notable that Agent UML has the active support of the Object Management Group as well as the Foundation for Intelligent Physical Agents (FIPA, 2002). The standardization work of these two bodies may further spur the widespread adoption of Agent UML for analysis and design of agent systems.

4.4.2 Modelling tools

Because agent-based simulation is relatively new, commercial off-the-shelf model development packages are still very rare except for the multi-paradigm toolkit AnyLogic (XJTechnologies, 2006). This is a significant drawback which is likely to be overcome in time with the proliferation of agent-based modelling tools which have expanded capability. Generally and currently, building a model still involves a great deal of manual coding. Aerogility (LostWax, 2005), marketed as a specialised agent-based tool for decision support in the aerospace aftermarket, does not fall in the category of an agent-based modelling package since a model still requires considerable bespoke coding in Java by their highly-skilled developers. In the meantime, however, a software framework like Java Agent Development Framework (JADE, 2006) and JACK (JACK, 2007), an agent-based integrated simulation environment like SeSAm (Klügl *et al.*, 2003), and standardised libraries like REPAST and SWARM (Tobias and Hofmann, 2004), can help to ease the modelling process.

4.5 Complex adaptive systems

A Complex Adaptive System (CAS), which has its roots in the biological sciences, is essentially a learning system with agents as its basic element. By implication, agents communicate and therefore an agent-based system makes sense only if it is a multi-agent system. As agents interact with each other and with their environment, they learn and become more complex over time, and by adapting to the pressures imposed upon them, their robustness and reliability are enhanced (Dooley *et al.*, 1995).

CAS is a relatively new field and, inevitably, its definition suffers from the uncertainty common to such new research areas. Dooley (1996) attempted to offer a concise definition but it turned out to be a lengthy synthesis of the seminal contributions by the principal researchers in this field. Nevertheless, it is a useful working description as it states clearly all the important features of a CAS in the way they are currently understood (see Appendix B).

Because a CAS adapts itself to environmental influences, it is an appropriate model for simulating dynamic systems like economies and supply chains which evolve according to fluctuations in business and commerce relationships. The proposal that a supply network emerges rather than result from purposeful design by a single entity has been advanced by Choi *et al* (2001) who also postulated that the performance of a supply network can be optimized by combining human managerial control and the network's emergent behaviour. To investigate the apparently spontaneous birth and death of a part or the whole of a supply network and possibly to learn how to optimize its performance, Pathak *et al* (2003) have built a multi-paradigm simulator using agent technology for that purpose.

4.5.1 Emergent behaviour

Emergence may be described as the spontaneous appearance of novel and coherent properties during the process of self-organization in complex systems (Goldstein, 1999). The emergent property cannot be easily deduced or predicted analytically from the properties of the individual components of the system. For example, the shape of a flock of birds cannot be deduced from the behaviour of one bird. It arises out of interactions between the components and it is because such interactions increase combinatorially with the number of components in the system which makes emergent behaviour in large systems unpredictable.

The element of unpredictability in large multi-agent systems may be of immense benefit in the study of group dynamics, e.g. crowd behaviour in a confined space, or investor behaviour in a stock market, but becomes a severe drawback in engineering where results must be predictable and repeatable.

4.6 Use and misuse of agents

Wooldridge and Jennings (1999) warn that, despite its appealing versatility and the intense enthusiasm of the research community to establish its use, agent-based software should not be used indiscriminately. Recognizing this, Macal and North (2006) provide some hints regarding the type of problems where agents may be used to good effect.

In coming to a decision whether to use or not to use agent technology, both technical and non-technical issues must be considered (Jennings and Wooldridge, 1995; Taylor *et al.*, 2005) if its usefulness is to be realised in practice. When wider system issues are taken into account, it is possible that other modelling paradigms like traditional discrete-event modelling or SD may turn out to give a better solution in a shorter time. It would be too optimistic to view agent technology as the silver bullet or philosopher's stone for all the difficulties likely to be encountered in simulation modelling today. It needs to be borne in mind that "*intelligent agents are ninety-nine percent computer science and one percent AI*" (Etzioni, 1996) which is taken to mean that conventional software technologies and techniques should not be neglected but should instead be exploited as much as possible even within an agent-based implementation.

There are situations when full-blown agents should not be used and it is important to note them. Jennings (2000) points to the existence of systems where predictability is a desirable property and which must be guaranteed. Because of its inherent capability of autonomous action, there is considerable scope for unexpected behaviour to emerge in a multiagent system. A consequence of this is that response times cannot be guaranteed. It may therefore be necessary to curtail an agent's power of self-determination thus making it more like a conventional passive object.

Another instance where it is inappropriate to employ agent technology is in systems where only a single thread of control exists (Wooldridge and Jennings, 1999) as agent systems are multithreaded so that they can be executed in parallel. This is in opposition to the philosophy of agent technology even though serializing the execution of a program can ensure repeatable results and sometime improve performance.

4.7 Summary

The concept of the agent has sprung from the closely related areas of AI, object-oriented programming and concurrent object based systems, and human-computer interface design. While AI has focused on the depth of research in the sub-fields of intelligence, agent research tends to involve the wider issues of intelligent systems.

Although none exists yet, there is a developing consensus for the definition of an agent as originally proposed by Wooldridge (1997) and restated by Jennings (2000).

There is a broad spectrum of agents which may be categorised by their architecture. Reactive agents must be completely specified at design time and they respond in real-time to changes in their environment. In contrast, deliberative agents, which are closely related to traditional AI systems, can give correct results in the absence of complete information but they take too long for their responses to be useful.

The reactive subsumption architecture can result in the type of complex behaviour normally expected of deliberative agents. It achieves this even though the computation it does to transform an input into an action is a just simple mapping which may be in the form of a lookup table or a set of condition-action rules.

The hybrid reactive-deliberative architecture makes use of the strengths of the constituent architectures while avoiding their weaknesses. The principle of layering according to agent function can be applied to simulation models as well.

Agent-based simulation has reached the point in its development where it is ready for commercial use. However, its progress is hindered by the severe lack of commercial-grade modelling tools.

Agents are centred on roles and tend to operate at a high level of logical reasoning. It is evident from the methodologies for analysis and design that the role is the basic unit of abstraction in agent systems.

A Complex Adaptive System is a multi-agent system which evolves over time in response to the changes in its problem environment. This concept seems to match the requirements of the supply chain well.

Emergent behaviour in multi-agent systems is usually unpredictable and is unlikely to be of benefit to engineering problems.

Although agent technology is versatile, it is not the panacea for all present difficulties in simulation modelling. One must be wary of the common pitfalls when employing this technology.

Chapter 5
Software Metrics

5.1 Introduction

Measurement underpins the engineering process and ever since there has been the need to control software projects, metrics have been developed to help meet that need. Having existed as a subject in its own right for more than 50 years, software metrics has grown to such an extent that it is necessary to sub-categorise it. A useful scheme is to sub-divide metrics into those measuring properties which are either internal or external to the software (Fenton and Pfleeger, 1997). Objective and quantitative metrics like the '*McCabe cyclomatic number*', for determining the structural complexity of program code, and *'lines of code'* (LOC), for gauging program size, are internal properties while the more subjective software quality metrics like *'maintainability'* and *'usability'* are external. While internal properties provide indicators which are more useful to the software developer, external properties tend to be of greater importance to the software user.

Programming language plays an important role when measuring software internal properties and this is especially significant when moving across language generations – from assembly to procedural to object-oriented and to agent-oriented languages. It also needs to be considered when comparing two or more programs that the metrics selected to characterise them are themselves not strongly related to the programming paradigms. To ensure a like-for-like comparison this principle must be followed as closely as practicable.

For a metric to be useful, it must be employed in the limited context in which it was originally validated. Just how useful a metric is can be determined by the degree to which it helps a user make a decision (DeMarco, 1982).

This chapter considers chiefly those aspects of software metrics which are likely to be useful to this study. As it is software code rather than its development process which will be considered, it is product metrics and not so much process metrics which will be the area of focus. The remainder of this chapter is set out as follows – metrics for program size, models for software quality, the maintainability quality factor, the measurement of software complexity, module coupling and cohesion, the effect of programming languages on metrics, and finally some code metrics for object-oriented languages are presented.

5.2 Program size

Although program size is a simple concept, this benchmark is fraught with difficulties. Traditionally, it has been quantified by the number of lines of code and it has been the metric widely adopted throughout the software industry. This section will consider how this metric has been used and how it may be adjusted to account for the different programming languages.

5.2.1 Lines of code

Among the very early software metrics to be put to commercial use for estimating development effort, and even program complexity, is '*lines of code*' (LOC). It is primarily a measure for program size and its long and widespread use has validated it to the extent that Basili and Hutchens (1983) proposed that it should be used as the *'null hypothesis'*, or benchmark, against which other metrics were to be compared. It is chiefly for this reason that LOC is considered here.

Despite its pedigree, LOC is not as well-defined as it should be. This may be attributed mainly to the existence of a variety of textual programming languages as well as visual programming languages which generate code automatically behind the user programming interfaces. Also, in the absence of any formal or informal standards, the definition of what constitutes a LOC is unclear. As a typical program can contain multi-statement lines, as well as data declarations, multi-line comments and blank lines which do not execute program function, simply counting the number of lines laid out on a page can diminish this metric's usefulness.

There is a need to remove such ambiguities and to that end, Park (1992) has developed a framework for program size measurement which includes some comprehensive checklists to help decide what should be included in a LOC count under different conditions. Broadly, these conditions recognise that code may be manually written, automatically generated or translated, or reused from a library since the LOC metric can be used as a measurement of programmer productivity. The benefits of providing a flexible measurement framework for LOC count are clear – it enables the consistent communication of requirements across a software project and it ensures the repeatability of LOC measurements.

LOC has been used traditionally for gauging programmer productivity, as LOC per unit time, with procedural languages like FORTRAN and C. It becomes invalid when used on its own to compare the productivity of different programmers using different languages (Jones, 1994). According to this measure, when implementing the same functions both in a low-level language like Assembler and in a high-level language like Java, a programmer can be shown to be less productive in the high-level language when all other factors remain the same. This is misleading, counter-intuitive, and exemplifies the effect of carelessly using a metric out of its validated context. Similarly, the productivity of a programmer using a visual programming environment can be grossly overstated if LOC of the automatically generated code is used as the only measure.

In a modular language, it may be more pertinent to count procedures and sub-routines instead of LOC. For object-oriented programs, a more accurate size metric has been shown to be the count of objects and methods (Lorenz, 1993), or classes and functions (Williams, 1994).

5.3 Software quality

In general terms, quality is taken to mean nothing more or less than conformance to requirements or, defined more specifically by the International Standards Organization (ISO), it is an attribute which a product or service must have to meet the needs and expectations of a customer (ISO, 1991). Quality viewed as fitness for a user’s needs is also favoured by Kitchenham (1989). Practically, quality is a composite of numerous overlapping attributes (Fenton and Pfleeger,

1997) and its notion is best illustrated by a model. Some general software quality models are presented in the following section.

Depending on his role, whether as manager, implementer, or end-user, a customer places different degrees of importance to the different quality attributes. While these attributes may be quantified, their value or weight will vary according to how they are perceived. For example, an end-user will place high importance on usability, reliability, and execution efficiency while a software developer will view maintainability, reusability, and interoperability to be more desirable. More than that, in a set of desirable attributes, some of them can be potentially contradictory (Conte *et al.*, 1986). For example, generalising a program so that it can run on different devices will tend to make it run slower on any of the devices. Conversely, increased efficiency often comes at the price of decreased portability, understandability, and maintainability. Therefore, while it is easy to contrive a single metric for overall software quality, it must be recognised that for the reasons just described such a rating cannot be universally useful (Boehm *et al.*, 1980; Boehm *et al.*, 1976).

5.3.1 Software quality models

Among the early software quality models, the two which have most influenced the development of software quality metrics are those of Boehm *et al* (1980) and McCall *et al* (1977). They are similar in some respects. Their hierarchical nature, clearly displaying the decompositional approach both have taken, are shown in Figure 5-1 and Figure 5-2 respectively. In both models, the same pattern of decomposing into three levels of abstraction can be seen. The hierarchical levels common to both models are –

- The manner of software usage and it may be categorized by the following questions –
    - How well can it be used as it is?
    - How easy is it to maintain?
    - How well can it be used if its environment is changed?
- Quality factors.
- Software primitives or attributes

.

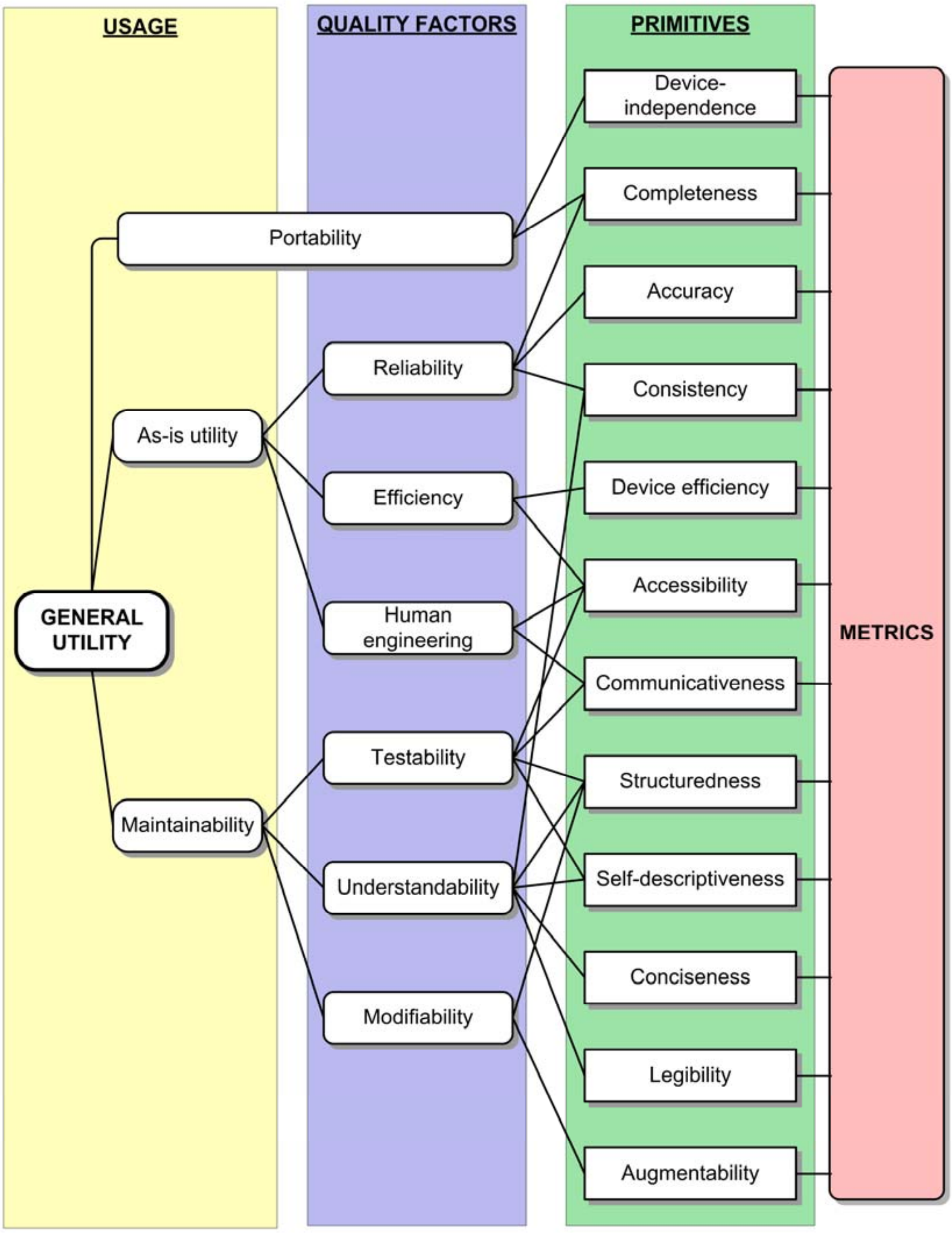


Figure 5-1: Software quality characteristics tree (based on Boehm *et al.*, 1980)

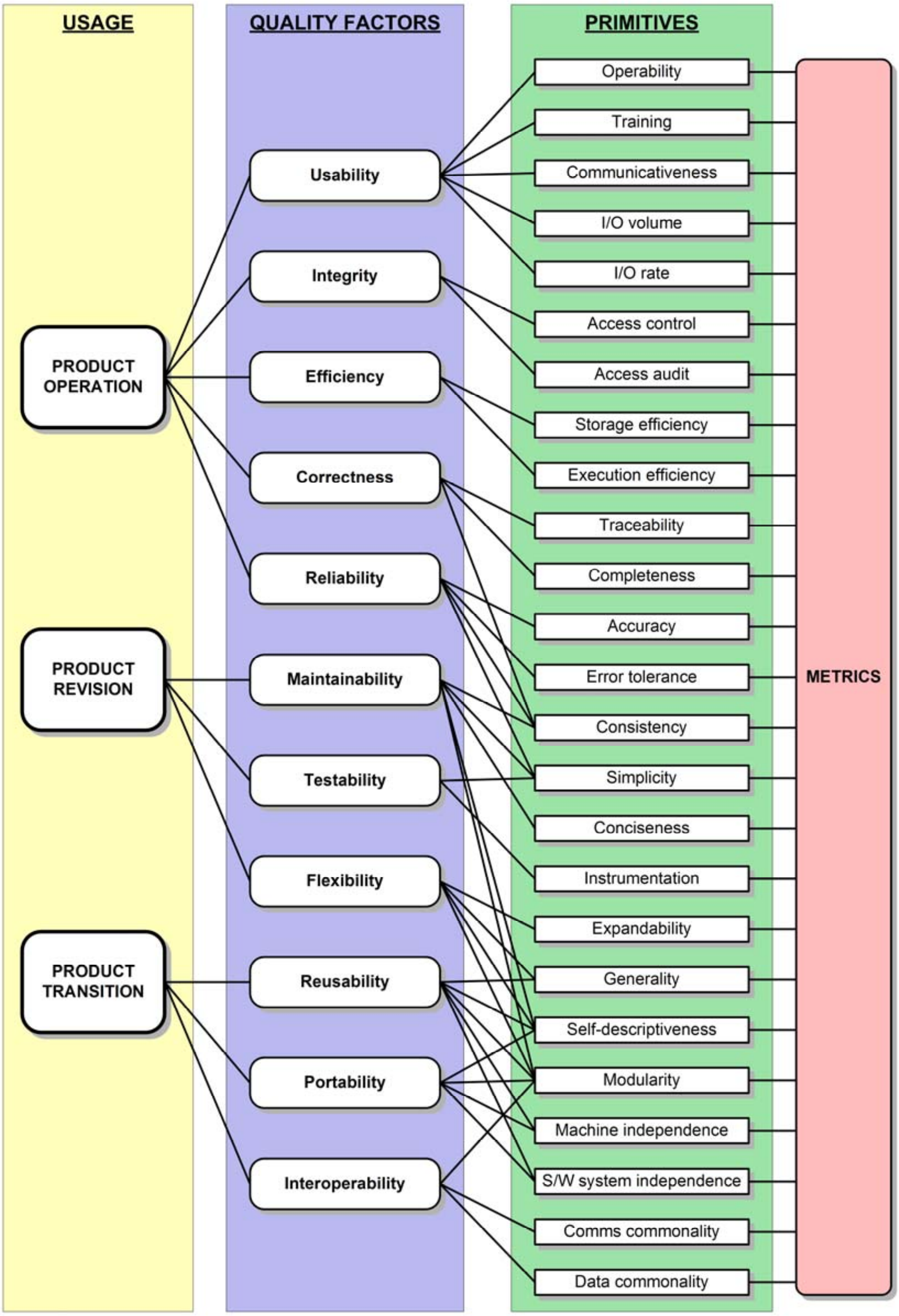


Figure 5-2 : Software quality model (based on McCall *et al.*, 1977)

As the quality factors, e.g. usability, testability, and portability, are still too abstract to be quantified meaningfully, they are decomposed further into attributes or primitives which are highly differentiated with respect to each other. The necessity of doing that is shown in the figures that the quality factors are not mutually exclusive but can consist of two or more key primitives which are sometimes shared with other quality factors.

A software quality milestone was reached in 1991 with the publication of ISO 9126 which used McCall's model as its basis. Some of the primitives are assigned to different quality factors in the two models. ISO 9126 is the first international standard providing a global common framework for evaluating software quality and has remained unamended even though it has subsequently been assessed as ambiguous and incomplete (Al-Kilidar *et al.*, 2005; Jung *et al.*, 2004).

Although these three models differ in detail and do not always map directly onto each other, their quality factors may be broadly viewed as functionality, reliability, efficiency, usability, maintainability, and portability. They are claimed to be comprehensive (ISO, 1991) and it may be inferred that an aspect of software quality can be described adequately by one or more of them. Particular attention must be exercised when comparing these models as the terms used are not always equivalent in definition. For example, '*understandability*' is defined from the software developer's aspect in Boehm *et al*'s model while the same term is considered from the software user's viewpoint in ISO 9126.

A software quality programme based on any of the three fixed models just presented need not include all their quality factors. The nature of the software to be developed, maintained, or procured will determine which of the factors are relevant, and the perceived degree of importance will determine how much each factor should be weighted (Fenton and Pfleeger, 1997). Although assigning numerical quantities to quality factors appears objective, it is subjective in practice. However, its value is realised in providing a common and consistent quality framework to all members of a software project team.

An alternative to the fixed model or *'big-bang'* approach, where all the required attributes are defined and fixed at the start, is to define the attributes and build the quality model incrementally as a software project progresses. As the requirements for a project are changed, so new attributes are defined with suitable and objectively measurable quantities to expand the scope of the model. In contrast to the inflexibility of the fixed model approach, the time horizon of this incremental method is short and there are opportunities for change during the course of a project. Therefore, it is more efficient and more likely to result in a closer fit to the project quality requirements. This adaptive approach pioneered by Gilb (1988) and extended by Kitchenham and Walker (1989) provides a good fit to the evolutionary nature of software development.

5.3.2 Maintainability

It is well established that software maintenance costs typically varies between 50% and 65% of overall lifetime costs (Guimaraes, 1983; Nosek and Palvia, 1990; Somerville, 2001). Of the activities included in the maintenance phase (i.e. post-implementation and delivery), enhancement and adaptation activities can make up more than 80% of the maintenance effort (Krogstie *et al.*, 2006). To be able to complete these activities, it is necessary to understand the software first, and then to modify and test it (Boehm *et al.*, 1980). Focusing on the maintainability aspect of software quality is justified since maintenance activities comprise a large proportion of software lifecycle time and expenditure.

It is logical to assume that the lower the complexity of the software, the less mental effort is required to understand it, and hence the more maintainable it is. Over a number of years, a large amount of software metrics has been collected on various software applications with the intention of describing maintainability using a single quantity. One such representation which is commonly used in software project quality assurance programmes is the Maintainability Index given by Oman and Hagemeister (1992) as –

$$\begin{aligned} \mathit{Maintainability\ Index} \ = \ & 171 - 3.42\ln(aveE) - 0.42avev(F) \\ & - 16.2\ln(aveLOC) + 0.99aveCM \end{aligned} \tag{5-1}$$

where *aveE* is the average Halstead Effort per module

avev(F) is the average extended cyclomatic complexity per module

aveLOC is the average lines of code per module

aveCM is the average number of lines of comments per module

The metrics for Halstead Effort and the extended cyclomatic complexity are described in Sections 5.4.1 and 5.4.2 respectively.

It has been contended that despite the large body of data on which the Maintainability Index is based on, it should nevertheless not be used on its own but must be used in conjunction with an estimation based on the '*man-in-the-loop*' (Welker, 2001). In this regard, the Delphi Method (described in Section 5.3.3), a method which can bring about the near-consensus agreement of a group of experts when applied well, has been used successfully to estimate quantities for problems where full knowledge is lacking. The well-established COCOMO (Constructive Cost Model) methodology (Boehm, 1981) for estimating software project cost and schedule is built on the usage of historical project data together with expert input obtained from Delphi estimation sessions. A variant of Boehm's Delphi method is frequently employed in software project estimation (Stellman and Greene, 2005) where the views of experts are used as surrogates for high quality project data.

5.3.3 The Delphi Method

The Delphi Method has its roots in The RAND Corporation in the early 1950s when it started to be developed as an interactive method for forecasting future trends in the '*inexact sciences'* like social science and political science (Helmer-Hirschberg and Rescher, 1958). Since then, it has been adapted for estimating task requirements in a variety of industries, usually in their sales and marketing activities. It has also proven to be especially effective when employed for scheduling activities in a software project plan (Stellman and Greene, 2005), for example, by estimating the time and human resources needed for developing a new software system or enhancing an existing one. The philosophy of the Delphi Method, as well as an evaluation of the technique and description of example applications is more than adequately covered in numerous textbooks. One of the popular and heavily referenced text, by Linstone and Turoff (2002), is also freely available online.

A tenet of the Delphi Method is that the consensual opinion reached by a relatively small panel of relevant experts participating anonymously in a controlled debate is more likely to give an accurate answer than one elicited from an individual. Further, because of the small number of participants involved in the execution of this method, it is accepted that it cannot produce statistically significant results, nor is that its intention (Brown, 1968; Gordon, 1994). Together with the participants' input of experience and intuition, a significant portion of its value lies in the inherent capturing, generation and synthesis of ideas in the iterative process of reaching consensus. Thus, anonymity to minimise the influence of psychological factors such as specious persuasion and the bandwagon effect, and feedback of information to encourage close, collective reasoning of an argument, are the two irreducible elements of the Delphi Method (Gordon, 1994).

While details of specific applications of the method may vary from one instance to another, the underlying principle of the Delphi process remains unchanged and may be described as follows (Brown, 1968; Gordon, 1994) –

- After a problem has been identified, between three and seven experts (Stellman and Greene, 2005) from disciplines related to the problem are invited to participate. They are assured that the statements they make will not be attributed by name.
- The problem is described to the participants at the first, or '*kick-off*', meeting. They are also given the initial questionnaire with the expressed expectation that they will work independently and provide their responses, as well as the degree of confidence in their scores, within a set timeframe.
- The scores to the questions are collated and analysed. If the results for a question differ greatly, the participants at the extreme ends of the range, i.e. outside the interquartile range (Brown, 1968), are invited to give a reasoned reassessment of their scores in view of the opinions returned by the other participants. This phase of the process is usually repeated and the questions may be refined according to the feedback obtained from the questionnaire.
- When the range of scores for a question is judged to have converged sufficiently, the median value is taken as the consensus score.

The application of this method for estimating software maintainability is described in greater detail in Appendix E.

5.4 Software complexity

As opposed to indirect metrics such as the software quality factors considered earlier, software complexity deals with the direct measurement of software attributes. It is important to be able obtain a measure of complexity as it is this characteristic of a body of software which can subsequently give rise to problems when it has to be understood, modified, tested, executed, and used.

There are numerous complexity metrics in existence and they may fall in any of the four categories (Fenton and Pfleeger, 1997) listed below –

- ***Problem or computational complexity*** – the complexity of the underlying problem.
- ***Algorithmic complexity*** – the complexity of the implemented program code to solve the problem.
- ***Structural complexity*** – the complexity of structure of the program code used to implement the algorithm.
- ***Cognitive complexity*** – the effort required to understand the program code.

Frequently, when complexity is mentioned in the software context, it is structural complexity which is meant. The same meaning is adopted in this thesis. Structural complexity may be decomposed further into three parts (Fenton and Pfleeger, 1997) –

- ***Control-flow structure*** – the sequence in which program code is executed.
- ***Data-flow structure*** – the trail of a data item as it is handled by a program.
- ***Data structure*** – the organisation of the data itself.

5.4.1 Halstead software science

The earliest metrics based on a coherent model of complexity was Halstead's software science (Halstead, 1977) which merged theories from computer science with those from cognitive psychology. Halstead proposed a set of metrics consisting of measures such as program length, vocabulary, volume, level, difficulty, and programming time. Of special interest is the metric for programming effort, *E*, which is still often used as the surrogate for program complexity in computation intensive applications. It is shown in Equation (5-2) and is described as the mental effort needed to understand and code a program and this quantity is estimated as –

$$E = \frac{\mu_1 N_2 N \log_2(\mu)}{2\mu_2} \qquad (5\text{-}2)$$

where μ_1 is the number of unique operators

μ_2 is the number of unique operands

$\mu = \mu_1 + \mu_2$

N_1 is the total occurrences of operators

N_2 is the total occurrences of operands

$N = N_1 + N_2$

Although the set of Halstead metrics was implemented and used in numerous automated metric gathering tools, it has largely faded from view because its underlying theories have been shown to be questionable (Coulter, 1983). In addition, Bowen's study (Bowen, 1978) of detected errors in program code concluded that Halstead's program length, N, had a weaker correlation between actual and predicted results than McCabe's cyclomatic number. An inherent weakness of this metric model was its strong dependence on the procedural programming languages which therefore diminished whatever remained of its validity when applied to other programming paradigms. The widespread adoption of object-oriented programming languages from the early 1990s has made Halstead software science metrics largely irrelevant.

5.4.2 McCabe's cyclomatic number

A metric for structural complexity is less dependent on programming language than the Halstead software science metrics. One such metric is McCabe's cyclomatic number (McCabe, 1976) which is still widely used as it has been shown to give good correlation with software quality factors like maintainability and testability (Grady, 1994).

The control flow of a program module can be represented as a flowgraph containing nodes and edges where the nodes are logic decision points and the edges are blocks of code between the decision points. The cyclomatic number, *v*, for a module is calculated as –

$$v(F) = e - n + 2p \tag{5-3}$$

where the flowgraph F has e edges, n nodes, and p connected components.

For a module with no procedure or sub-routine calls, p has a value of 1. In general, for a collection of flowgraphs, C, with k connected components, the cyclomatic number for them is calculated as –

$$v(C) = \sum_{1}^{k} v(C_i) \tag{5-4}$$

Equation (5-4) states that the cyclomatic number for a collection of connected flowgraphs is just a simple summation of the cyclomatic number of each of the connected components.

The cyclomatic number, v, is also a measure of the number of linearly independent paths through the flowgraph. For this reason, v may be used as a metric for the basis or minimum number of test cases to be executed for a module to ensure complete coverage. McCabe (1976) suggested from empirical evidence that a value of v greater than 10 indicated likely problems with module testability, and hence maintainability. Similarly, Grady (1994) concluded from a large-scale study of FORTRAN code that an upper limit of 15 should be enforced.

Calculating v for a large module using Equation (5-3) or Equation (5-4) can be very onerous for the reason that a flowgraph has to be constructed first. To enable the automated calculation of v from program code syntactic constructs, McCabe (1976) demonstrated that for a module with one connected component, v could be simplified to –

$$v(F) = d + 1 \tag{5-5}$$

where d is the total number of predicates, or decision statements, at the decision points. As only a total count is needed of logic predicates at points where program branching occurs, Equation (5-5) enables the automated determination of v from program code to be made by another program, thus speeding up the calculation dramatically.

A drawback of *v* is that it can only be evaluated late in the traditional software lifecycle, that is, at or after the coding stage and not before that. However, McCabe and Butler (1989) proposed a design complexity metric based on the same principle as the cyclomatic number but applied at the earlier design stage to a software structure chart in which the logical connections of the software modules are laid out. Although mathematically rigorous, it has not been adopted by software practitioners possibly because of the cumbersome task of having to reduce the structure chart iteratively before the cyclomatic number can be calculated.

It may be noted that McCabe's cyclomatic number is meaningful only when calculated for programs where the program control structure is fixed at compile time and thereafter. Data-driven or agent-based programs are capable of reconfiguring the control structure during program execution. Thus their structural complexity, and their cyclomatic number by implication, may not remain constant in the course of a single run. For this class of program, McCabe's cyclomatic number will not be valid as a measure of software quality with particular reference to program reliability.

5.4.3 Henry-Kafura's information flow complexity

Where McCabe's cyclomatic number is concerned with complexity within a single module or several connected modules, the Henry-Kafura (H-K) information flow complexity metric (Henry and Kafura, 1981) deals with inter-module information flow. As this metric works at the module call-graph or system structure chart level, it enables information flow complexity to be estimated early in the traditional software lifecycle, that is, at the design stage.

The metric for each system module is calculated using Equation (5-6) and those modules which have excessively high H-K values are marked out as possessing high risk of encountering maintenance problems during their lifetimes. To support this assertion, Henry and Kafura (1981) noted in their study on the UNIX operating system that modules with a high H-K value correlated with those having a record of high maintenance changes.

For a module, *M*, the H-K information flow complexity is calculated as –

$$\text{information flow complexity (M)} = length\,(M) \times ((fan - in(M)) \times (fan - out\,(M)))^2 \tag{5-6}$$

where *length* is the module length measured in LOC.

fan-in is the number of local flows terminating in *M* plus the number of global data structures from which information is retrieved by *M*.

fan-out is the number of local flows emanating from *M* plus the number of global data structures updated by *M*.

The HK quantity also indicates a module's degree of connectedness with other modules. It follows that a high H-K value signals a high likelihood that a change made in the module will ripple out to the adjacent modules and beyond. However, Equation (5-6) shows that a H-K value of zero can result when either the *fan-in* or *fan-out* value is zero. This can be misleading, for instance, in the case where *fan-in* is zero and *fan-out* is large and positive thus clearly indicating that the module affects other modules even when the H-K value of zero points to the module's isolation.

The model for this metric, initially presented in Henry and Kafura's seminal paper of 1981 has been studied in depth and has been refined and simplified (Shepperd and Ince, 1989) resulting in the following expression –

$$\text{Shepperd complexity (M)} = ((fan - in(M)) \times (fan - out(M)))^2 \tag{5-7}$$

They proposed a number of refinements about the inclusion and exclusion of modules, and local and global data structures. In particular, module length was left out because its inclusion in Equation (5-6) made the right-hand side into a hybrid expression – *fan-in* and *fan-out* could be determined at design time while *length* could only be determined at coding time at the earliest. It was only valid to use Equation (5-6) after code has been produced, thus obviating the original intention of estimating system complexity earlier in the software lifecycle.

In validating the results obtained using Equation (5-7) against real data, it was found to correlate well with the H-K metric. More than that, it exceeded the accuracy of the H-K metric for estimating software development time significantly (Shepperd and Ince, 1989).

5.4.4 Module coupling and cohesion

The aim of the structured design approach, which consists of techniques and considerations for program design, is to make the software implementation process less complex, more maintainable, and hence less costly (Stevens *et al.*, 1974). The need to divide a software system into modules also brings with it the natural need to determine how best to partition it while minimising inter-module communication. Stevens *et al*'s seminal paper introduced the two important concepts of external coupling and internal cohesion. In addition to the stated aims of structured design mentioned by Stevens *et al.* (1974), a common objective of minimising coupling and maximising cohesion is to encourage and ease software reuse. The great impact which these two software properties can have on software quality were noted by Bowen *et al.* (1983) when they observed that 5 of the 11 quality factors in McCall's software quality model (see Figure 5-2) were directly dependent on coupling and cohesion. These five quality factors, i.e. maintainability, flexibility, portability, reusability, and interoperability, are important when software is being revised or moved from one computer system to another.

External coupling is the degree of the interdependence between software modules. The fewer and simpler are the connections between the modules, the easier it is to understand one module without reference to the others. To date, there is no recognized standard measure for coupling. Fenton and Pfleeger (1997) list six categories of coupling on an ordinal scale from most to least desirable. Relation R_1 and R_2 are classified as loosely coupled while R_4 and R_5 are tightly coupled. The list of coupling categories is shown in Table 5-1.

Internal cohesion is the extent to which the different components of a module are needed to perform the same task. Both Stevens *et al.* (1974) and Yourdon and Constantine (1979) present almost identical lists of degrees of module cohesion on an ordinal scale from most to least desirable. They are shown in Table 5-2.

A module may exhibit one or more types of coupling and cohesion. Where that is true, the module is categorized by the least desirable type of coupling or cohesion.

Table 5-1: Types of coupling (Fenton and Pfleeger, 1997)

Relation	Coupling type	Description
R_0	No coupling	There is no communication between the modules, i.e. they are totally independent of each other.
R_1	Data	The modules communicate by parameters using either a single data element, or a homogeneous set of data items which have no control elements.
R_2	Stamp	The modules accept the same record type as a parameter.
R_3	Control	One module passes a parameter (or flag) to another with the intention of controlling its behaviour.
R_4	Common	The modules refer to the same global data. Although convenient, this type of couple is undesirable as a change to the format of the global data will require all common-coupled modules to be changed as well.
R_5	Content	One module branches into the inside of another module and alters the content.

Table 5-2: Types of cohesion (Yourdon and Constantine, 1979)

Cohesion type	Description
Functional	The module performs a single well-defined function.
Sequential	The module performs more than one function and they occur in the order prescribed by the specification
Communicational	The module performs more than one function but all on the same body of data which is not organized as a single type or structure.
Procedural	The module performs more than one function and they are related only to a general procedure affected by the software.
Temporal	The module performs more than one function and they are related only by the fact that they must occur within the same timespan.
Logical	The module performs more than one function and they are related only logically.
Coincidental	The module performs more than one function and they are unrelated.

5.5 Programming languages and paradigms

The history of computer programming languages does not appear to have a clear taxonomy which may be used to outline their development. A reason why may be that a new programming language is often the confluence of two or more earlier languages as well as of the ideas being circulated at the time. Very broadly, a programming language may fall wholly or largely into one of the programming paradigms, i.e. procedural (or imperative) programming, object-oriented (OO) programming, logical programming, or functional programming. It may further be classified according to its intended area of use which, for instance, may be as a system programming language, concurrent/distributed programming language, scripting language, or as a general purpose language. As examples, under this method of classification, Java may be described as an object-oriented concurrent programming language and C a procedural system programming language.

The choice of language used for a program can affect its quality. An empirical study comparing seven programming languages (Prechelt, 2000), showed significant differences in program size and structure, execution efficiency, and reliability between procedural, object-oriented, and scripting languages. Although the reported results were not counter-intuitive, the contribution to the results due to programmer capability was not thought to be significant even though productivity, as measured in LOC per hour, varied greatly within languages. Prechelt's study involved relatively small samples for each language and should therefore be taken as indicative rather than definitive. To reduce the variation in programmer capability, the outliers can be filtered out using the large body of language productivity data, in LOC per function point, collated by Jones (1996).

The Ada language was an outcome of the cost of software maintenance crisis experienced by the US Department of Defense in the 1970s. It was recognized that a well-designed language can increase productivity, maintainability, and reliability. Ada is a strongly typed language and provides support for a wide range of compile-time checks so that errors can be detected early during coding (Barnes, 1996). Together with the language features, the processes of standardizing Ada and rigorously validating its compiler and tools have combined to enable software of high quality to be implemented.

Until the early 1990s, research on software metrics was founded on the commonly used procedural languages like FORTRAN, COBOL, and C. The advent of OO programming and its wide acceptance by industry in general, compelled researchers to question the continuing validity of those measures. It was recognised that there were fundamental differences between procedural and OO languages.

There are concepts in OO languages for which procedural languages have no equivalence – for example object, class, attribute, inheritance, and message passing. Despite this, traditional metrics like LOC and McCabe's cyclomatic number can still be applied meaningfully at the lowest level of an OO program.

5.6 Object-oriented software metrics

The relevance of considering metrics for OO software is that almost all agent-based simulation models are implemented in an OO programming language like Java or C++ rather than a procedural language like FORTRAN and C. To address the features which distinguishes OO programming from the other programming paradigms, a set of six candidate measures were proposed by Chidamber and Kemerer (1991) and these measures have formed the basis of much research in this area. The suggested Chidamber-Kemerer (C-K) metrics were –

- **Weighted methods per class** (*WMC*) – sum of the complexity of each method within a class. *WMC* is usually calculated by an adding up the McCabe cyclomatic number for each method in the class.
- **Depth of inheritance tree** (*DIT*) – one may infer that design complexity increases with tree depth and that the software is less maintainable the deeper the tree.
- **Number of children** (*NOC*) – a high value implies a low level of reuse because of the high degree of dependency. It therefore leads to decreased testability and maintainability.
- **Coupling between objects** (*CBO*) – the extent of non-inheritance coupling between objects of different classes. A high value indicates low modularity and hence low maintainability.

- **Response for a class** (*RFC*) – the number of methods invoked in response to a message. A high value implies difficulty in testing and therefore decreased maintainability.
- **Lack of cohesion of methods** (*LCOM*) – the number of disjoint sets of methods in a class. Ideally, there should be only one set of methods in a class. A high value indicates bad class subdivision and a low degree of encapsulation.

Li and Henry (1993) reported that the correlation between maintenance effort and the C-K metrics was better than that for the null hypothesis, i.e. the LOC metric. More significantly, they also reported that the quality of the C-K metrics was good enough to enable a prediction of maintenance effort to be made.

Although the whole set of C-K metrics are normally used to characterise a body of OO software, some of these metrics may nevertheless be applicable to procedural language software which has highly structured and modular design. For example, a module written in C containing a number of procedures and functions may be structured and used in a similar manner to an OO class with numerous methods. Similarly, a message to a C module can initiate the execution of a number of procedures and functions, much like the response of an OO class after receiving a message. Therefore, it is not unreasonable to use the two C-K metrics, *WMC* and *RFC*, for procedural language software.

5.7 Summary

Software metrics may be categorised as those measuring the internal or external properties of a program. For such metrics to be useful and meaningful, they must be employed in the context in which they were originally validated.

Program size is usually measured as lines of code and is considered as the benchmark metric for program size. Care must be exercised to determine what should be counted as a line of code. Although LOC has been popular as a software metric for many years, it can be unhelpful and misleading when used to compare programs implemented in different programming paradigms. It is sometimes better to use modules, e.g. procedures and classes, as the basic unit of program size since it is less language dependent.

Software quality is a composite of several overlapping attributes which are perceived and prioritised differently by people in different roles. Therefore it is not likely that a universally recognised, meaningful and useful quality metric can be represented using a single number. The Boehm, McCall, and ISO 9126 software quality models are hierarchical and they use broad factors to characterise quality. These factors are decomposed into primitives to which numerical values can be assigned. An alternative to managing software quality during the course of development is to use a model which is amended incrementally. It enables closer control than the fixed model.

Programming language can affect software quality such as maintainability, execution efficiency, and reliability. In addition to that, the use of a well-designed language like Ada together with its validated tools can increase productivity.

Maintainability is important because software maintenance is costly and time-consuming. The estimation of this software quality needs to be built on historical data as well as the input of human experts. The Delphi Method, an established process for eliciting opinions from human experts, provides the means for reaching consensus.

The measurement of complexity is essential to the understanding of a body of software. Structural complexity, the most prominent of the four aspects of software complexity, can be decomposed into control-flow and data-flow.

The metrics of Halstead software science are based on program elements such as the number of operators and operands. They have fallen into disuse after object-oriented programming became established.

McCabe's cyclomatic number concerns control-flow structure and is very simple to extract from program code. It is still widely applied to procedural language programs and the lowest levels of object-oriented programs.

The Henry-Kafura metric deals with information flow between a group of related modules. It can be used to identify areas of high complexity, and hence potential future trouble spots early in the software lifecycle, i.e. at the time of design rather than after coding. However, its lack of a rigorous definition can sometimes result in anomalies. A better metric for information flow is the Shepperd complexity.

An alternative, high-level, and ready measure of system complexity is the degree or quality of coupling between modules and cohesion within a module. These two software properties are important as they are an influencing factor in about half of the McCall's software quality factors.

Some metrics cannot be used outside their programming paradigm. There are some concepts in object-oriented language which have no equivalents in procedural languages.

Not all the metrics described in this chapter can be used within the context of the research as stated earlier in Section 1.4. Those which are to be used to compare the agent-based model with the discrete-event model are set out later in Section 6.4.1.

Chapter 6
Case Study

6.1 Introduction

The subject of this research coincided with one of the areas of investigation in the IPAS Project (IPAS, 2007). As the aims of IPAS and this research were also closely similar, it was therefore considered reasonable for the work to become an integral part the project.

The central object of this research is to compare an agent-based model with a traditional discrete-event model, both of which have been implemented to the same functional specification. To start towards that goal, a non-trivial, representative, and realistic case study scenario needed to be selected and it was found in an existing study of an engine fleet global repair operation for the Trent 800 which had been carried out by Rolls-Royce plc (RR). A study such as this can help, early in the engine lifecycle, i.e. at the design stage, in understanding how to increase flying time, decrease maintenance cost, reduce stocking of spare parts, and thereby make Rolls-Royce's TotalCare® and CorporateCare® commercial arrangement of *'power by the hour'* more profitable.

In addition to a description of the case study scenario and a description of the Java agent-based model and the data required to run it, this chapter contains a further two sections which present the implementation of the traditional discrete-event model, and the empirical results from the agent-based model and the discrete-event model.

6.1.1 The case for a case study

Research into a software topic may typically be classified as one of the three techniques – a formal experiment, a case study, or a survey. As a survey is a retrospective study, the only action it can take is to record situations fixed in time so as to obtain a body of statistically significant data. Since it considers events which have passed, the factors of interest in a survey cannot be manipulated. By contrast, an experiment is a controlled investigation where factors of interest are manipulated with the intention of capturing information about all possible cases (Fenton and Pfleeger, 1997). It may also be noted here that a research method is reflected in its scale (Kitchenham *et al.*, 1995). As an experiment requires close and detailed control, it tends to involve small quantities. Again, by contrast, a survey tries to gather data over large groups of projects.

A case study is not as rigorous and controlled as an experiment nor is the amount of data gathered as large as that of a survey. Unlike an experiment or a survey, the subject of its investigation is a software project typically encountered in a specific area of application. In terms of scale, a case study lies in between an experiment and a survey. To describe it definitively, "*a case study is an empirical inquiry that investigates contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident*." (Yin, 2003) By focusing on the particular within its contextual conditions, it attempts to understand the general.

In order to compare two different modelling approaches, it is necessary to select and use two models which are as alike as possible. In this research, a meaningful survey on existing software cannot be carried out as it was noted in Section 2.5 that near-identical pairs of models for comparative study are very rare and, more specifically, none appear to exist for an ABM and DEM comparison.

An experiment is an appropriate method for comparing alternative modelling approaches but because it involves a high level of control, it is almost always used for investigating relatively small, self-standing tasks which can be isolated from its context, or the rest of a development process (Kitchenham *et al.*, 1995; Yin, 2003). Investigating alternative modelling approaches will require a series of experiments and it is likely to be too costly in time and effort. In the face of these constraints, the case study is judged to be the most appropriate research method, and therefore preferred before the experiment and the survey.

The case study described later in Section 6.2.1 contains a typical scenario for the lifecycle costing of an engineering product. Some of the factors which can contribute significantly to through-life cost, for example, the rules determining the inspection, repair, scrapping, transport, storage, and supply of a jet engine component are all present in the scenario. Also, the levels of details, ranging from flexible, high-level decision making to fixed, fine-grained processes, provide a good representation of a real-world engineering application.

6.1.2 Validity of methodology

The steps to be followed for the case study have been set out earlier in Section 1.5. It may be noted that the design of the case study presented there follows the long-established pattern for this research method (Fenton and Pfleeger, 1997; Yin, 2003), and it may be detailed as the following phases – preparing for data collection, collecting the evidence, analysing the evidence, and reporting the case study.

In the design of the case study to test the hypothesis that agent-based modelling is better than discrete-event modelling, it is a desirable aim to control as much of the modelling processes as possible (Fenton and Pfleeger, 1997). This may be achieved by making make them closely similar, and in doing so, greater confidence may be attributed to differences in the data collected to real differences in the modelling processes. Therefore, in preparation for data collection, both models are built using model builders of similar experience, to the same specification, validated as functionally identical, and run in the same computing environment.

To arrive at a system-wide view of a relatively broad subject such as a modelling paradigm, both qualitative and quantitative methods need to be employed. It has been emphasised by Boehm (1981) (see Section 5.3.2) that sometimes, to increase confidence in quantitative metrics, they may need to be supplemented with the opinions of human experts. Quantities such as code size and structural complexity provide a perspective of a model’s internal properties while qualities such as understandability, modifiability, and testability deal with its external properties. Further, these metrics need to be collected for smaller, identifiable subsystems of the model as the aggregated values for the whole model can conceal variations which may be significant.

The relative ease of post-implementation maintenance of the model is indicated by smaller code size, lower structural complexity, and code which is easier to understand, modify, and test. It is reasonable to expect that by drawing together these beneficial aspects from both modelling paradigms, a novel and better modelling approach can emerge.

6.2 The case study

This section starts with a description of the engine fleet maintenance case study scenario. The maintenance policy options and the data required are then presented. The section is concluded with a brief description of the agent-based model supplied by the Strategic Research Centre, Rolls-Royce, Derby.

6.2.1 The scenario

The scenario may be described broadly as a problem involving scheduling and logistics for the ongoing maintenance of a fleet of Trent 800 (Figure 6-1) high-bypass, three-spool, turbofan engines. The engine's initiation into commercial service occurred in April 1996 and has since become the dominant powerplant for a number of Boeing 777 aircraft variants in service worldwide. To illustrate this, Table 6-1 gives an overview of the engine fleet's operational status up to the end of September 2007. Its reliability while in operation by the various airline operators is evident as the average daily utilisation is almost 12 hours. Its operational record is also notable in that one of the engines has not been subjected to an off-wing overhaul even though it has done more than 5,000 flights.

6.2.1.1 Engine lifecycle

The engine lifecycle phase of particular interest to the case study extends from just after entry into fleet operation until just before retirement and disposal. As shown in Figure 6-2, this maintenance phase of the engine lifecycle corresponds to the following stages of the Rolls-Royce Derwent Cycle –

- ***Stage 4***, where the product is manufactured and in-service support is provided. It may also be modified to meet ongoing regulatory, customer, and business needs.
- ***Stage 5***, where customer operation of the product continues to be supported.

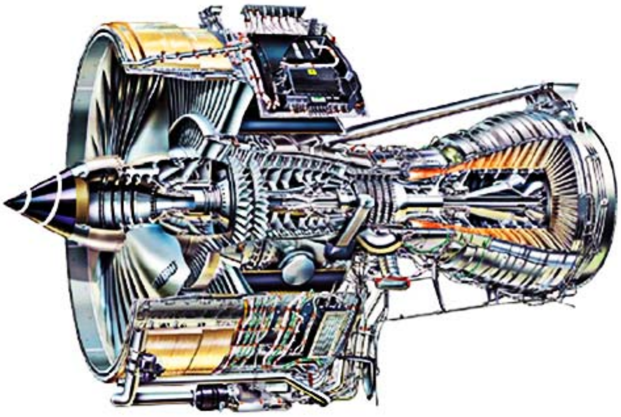


Figure 6-1: The Rolls-Royce Trent 800

Table 6-1: Rolls-Royce Trent 800 fleet operational summary (Rolls-Royce, 2008)

Item description	Quantity
Number of engines	502
Number of aircraft in service	221
Total fleet engine hours	11.72 million hours
Lead engine	42,931 hours
Lead engine without shop visit	30,169 hours
Total fleet engine cycles	2.28 million cycles
Lead engine	11,040 cycles
Lead engine without shop visit	5,004 cycles
Average daily utilisation	11.6 hours
Average stage length	4.8 hours (ranging between 2.4 to 10.3 hours)

Further, the scenario is concerned only with off-wing, full overhaul with repairs carried out either in an overhaul base (OHB) or a component repair vendor (CRV) and not with routine on-wing servicing, off-wing maintenance at an airport, or repairs carried out in a '*hospital shop*' (Rolls-Royce, 2005a). It is while in Stage 5 that an engine undergoes full overhauls, or shop visits, at OHBs spread strategically around the world (see Figure 6-3). The OHB at which work is to be carried out will vary according to the maintenance policy Rolls-Royce has agreed with the airline operator.

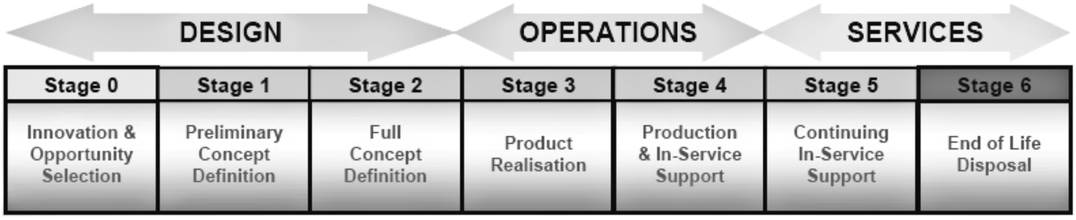


Figure 6-2 : The Derwent Cycle (Rolls-Royce, 2005b)

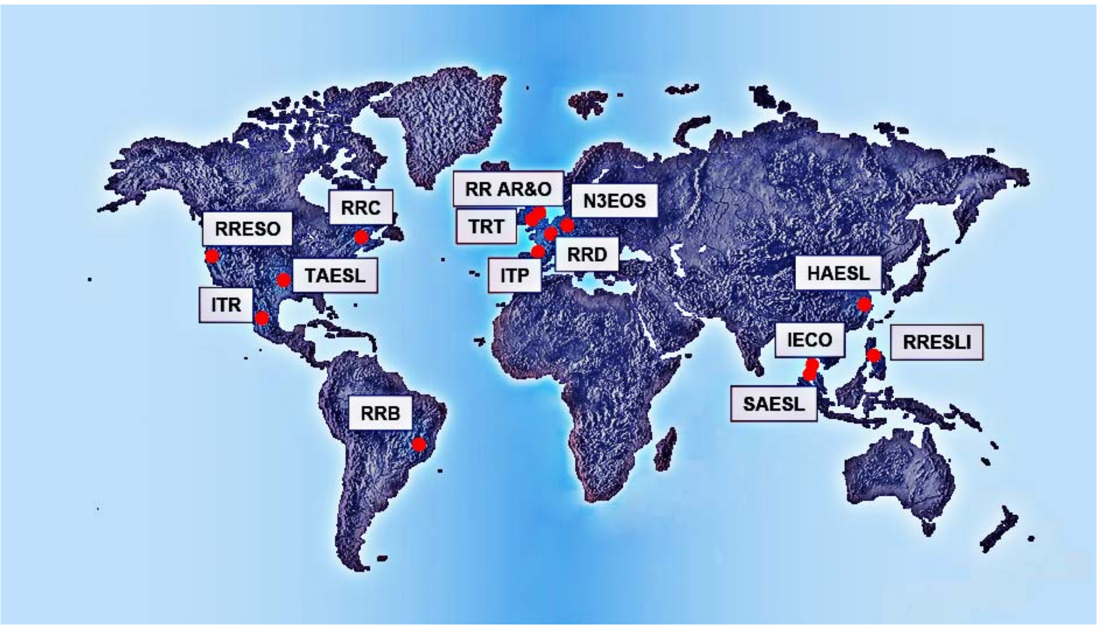


Figure 6-3: Network of repair and maintenance locations (Rolls-Royce, 2005a)

6.2.1.2 The engine maintenance process

It is assumed that after a new engine has been introduced into fleet service it will keep flying a fixed stage length to a simple schedule until it is withdrawn from normal operation. For each flight, the flight profile is not taken into account and it is also assumed that the engine performs within the parameters for normal operation.

An engine is withdrawn permanently for disposal once it has reached the number of flights defining its retirement limit. However, before that point is reached, an engine is subjected to either on-wing or off-wing maintenance according to the nature of the work required. In the event of off-wing maintenance, an engine is temporarily withdrawn from operation and inducted into a workshop for one or both of the following reasons –

- It is for a scheduled full shop visit at an OHB because it has reached, or is about to reach, the specified number of flights since the previous shop visit.
- It has sustained damage too severe or extensive that attention either in an airport workshop or in a hospital shop is not adequate. If such damage is close to a scheduled shop visit, the repair is done in conjunction with the planned overhaul.

A high level view of the typical workflow in a Rolls-Royce OHB is depicted in Figure 6-4. Details of the maintenance work needed to be carried out in a scheduled overhaul are described in the workflow plans or process maps laid out over the six diagrams shown in Appendix C as well as a textual functional specification. These six flowcharts or process maps may be concatenated at the places as indicated to form an overall process map.

When the Trent 800 engine arrives at the OHB for a scheduled shop visit, decisions are made to determine which of its eight modules need to be removed and stripped. The modules which require maintenance are stripped down to their individual parts all of which are inspected and assessed whether they should be refitted with no further work needed, sent for repair at a CRV, or replaced with a spare part.

Where a part has been assessed as suitable for refitting, it is set aside and no other work is done on it. It awaits module rebuild to commence once all the module parts are available.

If a part is deemed no longer fit for service, then a new or a previously repaired part is withdrawn from the stock of spares at the CRV. If a replacement is not available, a new part is ordered from the Parts Service Centre (PSC). The replacement parts are batched by part number and shipped to the OHB to await module rebuild.

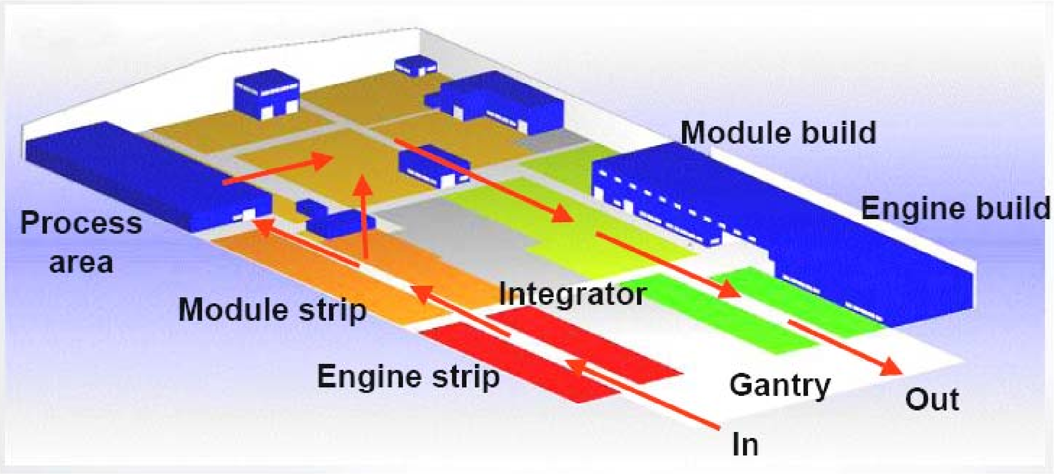


Figure 6-4: Typical workflow in an overhaul base (Rolls-Royce, 2005a)

Where a part needs to be repaired, the work is always attempted at a CRV. If the attempt at repair is not successful, the part is scrapped and a replacement is sought first from the CRV stores, or if none is available there, a new part is ordered from the PSC. If the repair is successful, the life of the part is always restored to its '*as-new*' value. The repaired and replaced parts are batched by part number and shipped back to the OHB.

Each module is rebuilt at the OHB once its full complement of parts is available. Finally, the modules are reassembled into engines, thus completing the overhaul process.

It is very likely that there will be occasions when two or more engines from different airline operators are overhauled at the same time in the same OHB. It is the normal practice that the stripped parts from different engines are not mixed up but are returned to their engines of origin. However, it is also possible that the maintenance policy of an airline operator may not demand this constraint to be exercised for its own engines. To ensure that the parts are returned to their originating locations, each part must be identified uniquely and tracked throughout the repair process.

6.2.2 Maintenance options

To enable different engine maintenance rules to be explored using the same simulation model, the model's functional specification as represented in the process maps (see Appendix C) may be configured according to options shown in Table 6-2.

From Table 6-2, a '*fixed*' workscope involves specifying a fixed list of module parts to be inspected whereas a '*customised*' workscope involves inspecting all the parts making up an engine module. A workscope can be implemented in conjunction with the type of inspection to be carried out. If the *'ship dirty'* rule is in force, the module will be stripped down to its individual parts at the OHB, batched by part number, shipped to the appropriate CRV and inspected there. Should the *'inspect at OHB'* rule be in force, a module part is inspected at the OHB and then determined if it may be refitted without further work, or if a repair may be attempted at a CRV, or if it should be scrapped. Where repair is required, the parts are shipped to a CRV in part-numbered batches.

Table 6-2: Configuration options for maintenance rules

Maintenance rule	Configuration option	
1. Workscope	Fixed	Customised
2. Inspection	Ship dirty	Inspect at OHB
3. Kitting	Yes	No
4. Module swap	Yes	No
5. Smoothing at component repair vendor (CRV)	Yes	No

In the maintenance approach which involves '*kitting*', a number of parts from a fixed parts list are marshalled in the Aftermarket Service Centre (ASC), or less probably in a CRV because of its limited range of parts. They are then sent as a kit in a single shipment to the OHB for rebuilding into a module. Kitting therefore implies a fixed workscope. Also, by extension, a '*module swap*' implies kitting is in operation for the reason that all the kits which make up a module can be sent in the same shipment. In normal kitting, the kits are sent in separate shipments once they have been marshalled.

The smoothing of workload at a CRV requires an appropriately large buffer of spare parts to be maintained in the CRV stores so that any temporary upsurge in workload will not result in a shortage of parts and hence a delay while waiting for the reordered parts to arrive. Although it is desirable to exploit resources effectively by maintaining a constant workload, an outcome of the requirement to smooth the workload is additional financial penalties because of the larger amount of stock which has to be carried, as well as having to run an automated stock control system to maintain the level of stock.

The maintenance options described in the preceding paragraphs are necessary since the ongoing cost of engine maintenance is influenced by policies related to inspection, shipping, repair, and scrapping of engine parts in operational service as well as the usage and stocking of spare parts. An engine part may be subject to a hard limit based on total number of flight cycles, flight hours, or number of repairs. Also, airlines do have their preferences about the past history of parts which can be fitted to their engines. For example, only repaired parts from their own engines, or from specific airlines but not from others, can be refitted.

6.2.3 Data for modelling the scenario

The granularity of a simulation model data inputs needs to be sufficiently fine if its results are to be of a quality good enough to be validated against historical data. Through a process of repeated refinement, the detail level of a model as well as the data needed to run it can be determined so as to satisfy such a level of detail and validity.

As this case study involves both engineering processes as well as aspects of logistics and high-level decision-making, data required include not only engine data down to component level but also operational data for time and costs for transportation, engine stripping and rebuilding, stocking and reordering of parts, and penalties for late delivery. Operational information gathered from other Rolls-Royce civil jet engines of similar design, has identified a set of 67 engine components (listed in Table D-1 in Appendix D) which contribute significantly to maintenance cost and effort over an engine’s in-service life. It is chiefly for this reason that only these components are considered for modelling global maintenance operation of the Trent 800. The components are drawn from the following engine modules (see Table 6-3) –

Table 6-3: Rolls-Royce Trent 800 module designations

Module number	Module description
1	Fan
2	Intermediate pressure compressor
4	High pressure compressor and turbine
5	Intermediate pressure turbine
8	Low pressure turbine

As the data is commercially-sensitive information, the numerical values supplied by Rolls-Royce have been amended and they bear little resemblance to the original. However, this adjustment is acceptable for the purpose of this research as the same set of data will be used by the agent-based model as well as the traditional discrete-event model. The results from the two models can therefore be directly compared.

6.2.4 The agent-based model

Before the commencement of this research, the scenario described earlier in this chapter (see Section 6.2.1) has been implemented as an agent-based model by the Strategic Research Centre (SRC) in Rolls-Royce.

The model is a multi-agent system which has been programmed in Java using the open source Eclipse software development kit (Eclipse, 2006). JADE, the Java Agent Development Framework (JADE, 2006), enables the agent system to be developed in compliance with the FIPA specifications (FIPA, 2002) and provides both the interface for managing the agents as well as the environment through which the agents communicate. Although the JADE platform can be distributed over several hosts, it runs on only one host in this instance. In a host, the Java Virtual Machine (JVM) provides a complete runtime environment which allows a multi-agent system to be executed as a multi-threaded process. In this instance, the JVM enables several agents to run concurrently and asynchronously, with each agent allocated to a separate thread of execution.

In this model, the top-level functions fulfilling all the requirements set out in the six process maps in Appendix C have been coded as individual agents while the functions within them have been implemented as agent behaviours. The program code of the model, whose agents are summarized in Table 6-4, has undergone thorough dynamic testing. A part of this programme of testing involved an end-to-end validation in which a selection of its outputs were validated against some relatively simple, manually worked out examples. This was aided by an animation of the model (see Figure 6-5) where the number of items in each location and the flow of items between locations can be clearly seen. Such testing is required to ensure that all functions have been implemented as set out in the process maps and the textual specifications.

Having validated this model against the functional specification, it then became the basis for dynamic validation of the functionally identical traditional discrete-event model built using the Extend™6 modelling tool (described later in Section 6.3).

Table 6-4: Agents and their functions

Agent	Instances	Functions
Fleet Manager	1	Manages engines between shop visits; sets states of engine components before they arrive at OHB.
Overhaul Base (OHB)	≥ 1	Manages engine overhaul; inspects engine components; coordinates shipping, ordering, repairing, and scrapping of engine parts.
Component Repair Vendor (CRV)	≥ 1	Inspects engine components; repairs engine parts, and supplies repaired engine parts.
Aftermarket Service Centre (ASC)	≥ 1	Marshals engine parts into kits or modules; ships kits to OHB
Parts Service Centre (PSC)	1	Supplies new engine parts to OHB and ASC.

6.2.4.1 Model categorisation

As described earlier in Section 2.2, a simulation model may be categorised either by the objective of the study or by its representation of state and time. In terms of the latter classification, this ABM is a time-driven model since it is executed at regular time-steps and events are assumed to occur not within the time-intervals but only at the time-interval boundaries.

Basically, the model addresses a problem in lifecycle costing and therefore elements of logistics, scheduling, and operational policies need to be present. The primary objective of the model is to use it as a design tool to determine the lifetime cost of maintaining an engine fleet when, for example, the design life of an engine part is changed or the rules governing its maintenance process are varied. Such design and maintenance policy changes will influence the stocking, supply, and scheduling of spare parts and hence, lifetime system cost.

6.2.4.2 Model rationale

The model enables the overall as well as the constituent costs for an engine fleet accumulated over a typical lifetime of 40 years to be determined. It may also be configured to run with any of the maintenance policies outlined in Section 6.2.2 and hence allow a credible, cost-effective solution to be selected. A simulation run of the model will also yield time histories of the demand for resources such as spare parts and transport, and thus enable availability planning to be managed.

Primarily, the use of this ABM is as a decision support tool because it can give a design engineer the ability to predict lifecycle costs early in the design process (for example, Stage 1 in the Rolls-Royce Derwent Cycle (see Figure 6-2)) and thus judge the desirability or otherwise of a design decision. The model can provide a holistic and more realistic view of a design by encompassing other influencing factors such as the cost of money, materials, and human resources each of which can fluctuate over the lifetime of the simulation run. Armed with such a tool, the design engineer can perform trade-off studies between cost and performance which are more accurately informed than currently available and thus arrive at better design decisions.

While model structural changes may be inevitable when implementing major features like maintenance and airline operation policies, minor but useful changes can be effected simply by modifying model input data. For instance, a change in the material specification of an engine component can result in improved performance by the extension of its life but can be accompanied by an increase in its initial acquisition cost. These new values of component life and initial cost then become the model inputs for determining its accumulated cost over its lifetime.

6.3 The discrete-event model

This section contains two sub-sections the first of which describes the implementation of the model, and the second presents a description of the model.

6.3.1 Implementation of the model

As discrete-event modelling has been used in industry for many years, several mature modelling products are available as commercial off-the-shelf (COTS) packages. Among them are well-established tools like Arena™, Enterprise Dynamics™, Extend™, Promodel™, Simul8™, and Witness™ all of which vary in quality and price. A reason for this variability is that, in addition to the tools' core competences for discrete-event modelling, they all possess other useful and user-friendly features which are intended to set each apart from the other.

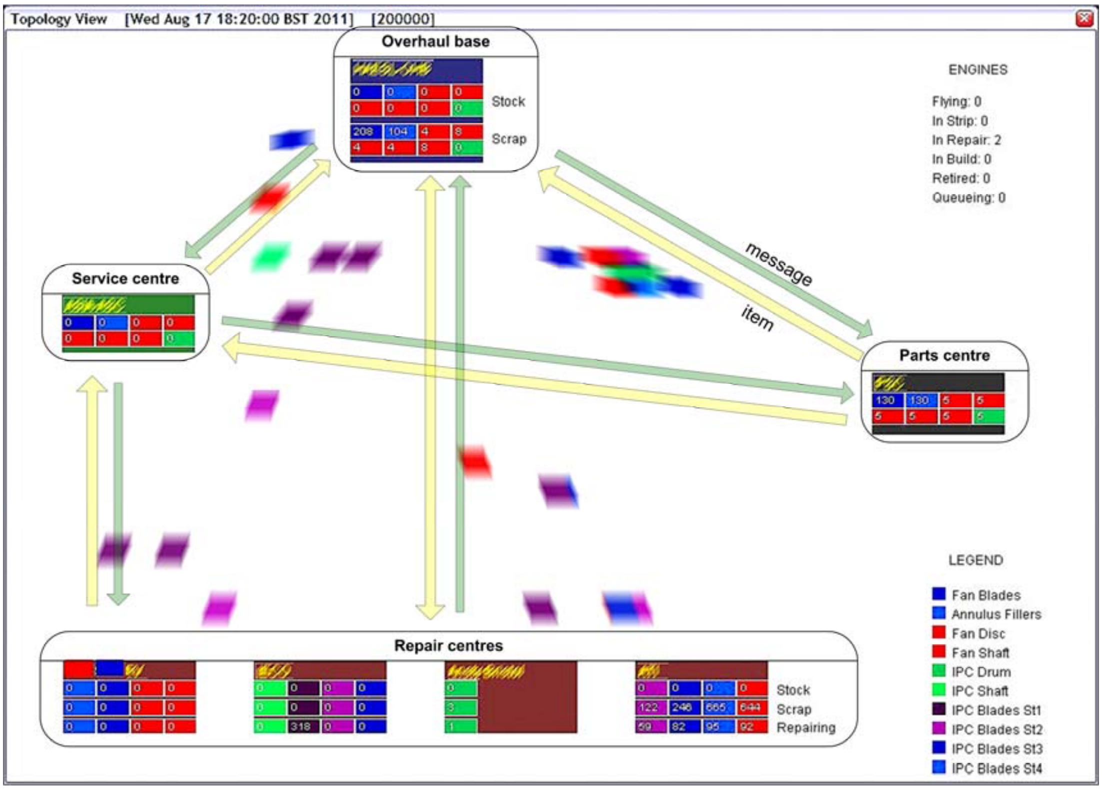


Figure 6-5: An annotated topology and animation of the agent-based model

The general requirements for a modelling tool to be used in this research were that it must –

- Have an intuitive user interface as the end-user is assumed to have no previous experience of such modelling tools.
- Be adequately flexible to enable the tool to be modified for experimentation in an academic research environment.
- Be fast and efficient in operation as the model can be large and complex.
- Be affordable and well-supported by the tool's developer.

Extend™6 which is developed by Imagine That, Inc. (ImagineThat, 2006) more than satisfies these general requirements by being open source and by giving free customer support. An evaluation of modelling tools carried out by Tewoldeberhan (2002) underscored not only the suitability of Extend™6 for developing models but also its cost effectiveness.

6.3.1.1 Model lifecycle

The processes used in the development of the DEM follow the typical model lifecycle sequence found in Law and Kelton (1999). This model lifecycle is shown on the left side of Figure 6-6. As shown in the same figure, it maps well onto the traditional systems development lifecycle (DoJ, 2003) commonly known as the '*Waterfall Model*' (Royce, 1987; Somerville, 2001). The reasons this methodology was adopted for the development of the DEM were that it was adequately rigid to ensure a disciplined approach was enforced and that testing was a requirement at every stage. The methodology is not overly rigid as it allows test results from one stage to be fed back to the previous stage if errors are discovered and have to be corrected. Also, the duration of implementation for the model was anticipated to be short so that the inflexibility of this method did not hinder the progress of model development significantly. A recent survey of software project managers, system designers, and developers (Neill and Laplante, 2003) showed that despite the availability of more flexible and up-to-date methodologies, the '*Waterfall Model*' was prevalent (about 40%) for projects lasting up to two years.

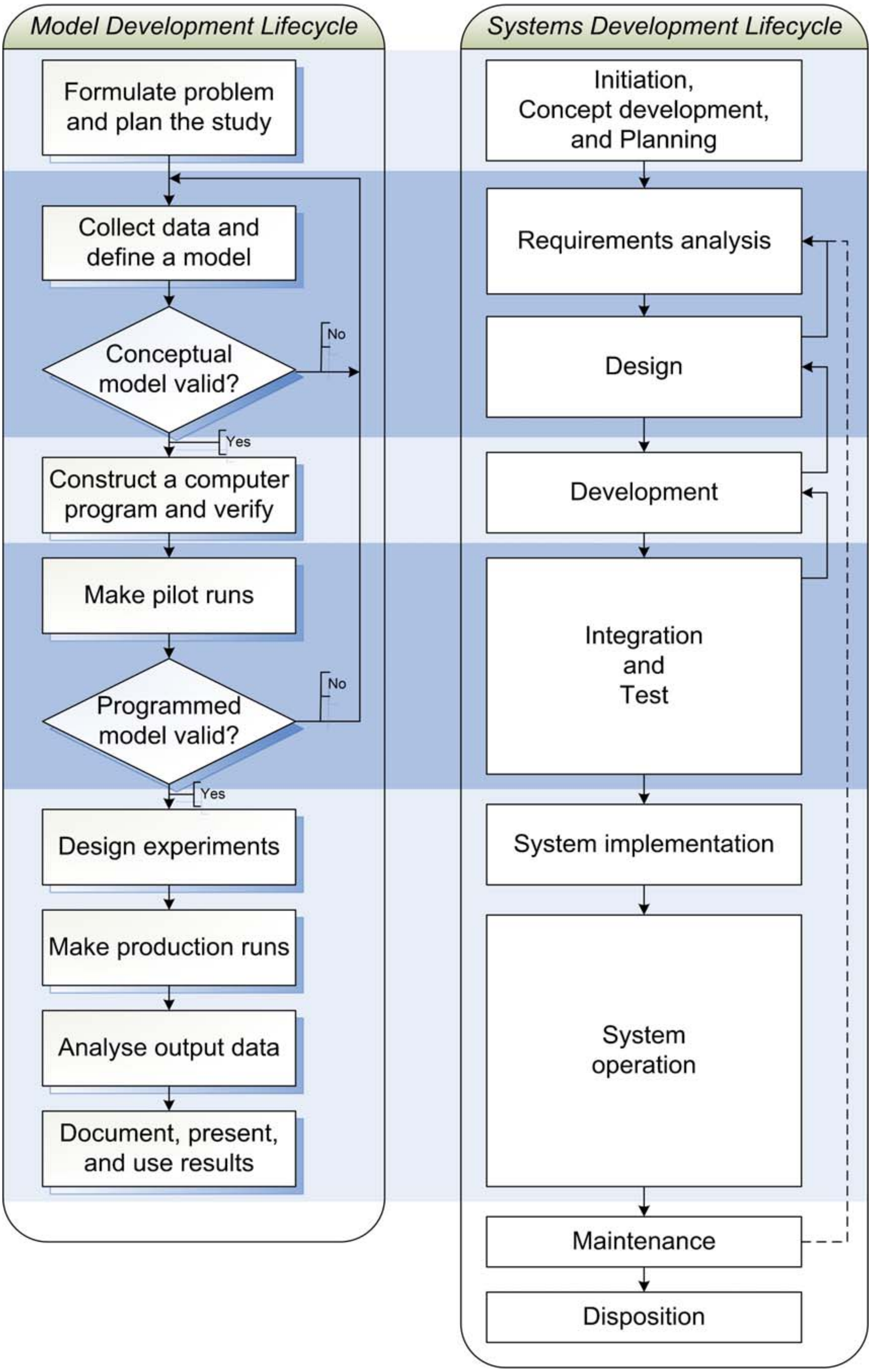


Figure 6-6: DEM development lifecycle (Law and Kelton, 1999) and the traditional systems development lifecycle (DoJ, 2003)

There are other development models, for instance, the Spiral (Boehm, 1988), and Rapid Application Development (Martin, 1991) models, but they were considered inappropriate for this task of producing a DEM which matched the ABM as closely as possible. These methodologies are better employed for implementing new and large software systems on their own.

6.3.1.2 Model verification and validation

Just how close a model is to reality may be revealed through the activities of verification and validation. It is necessary to note the distinction between these two activities as they address different aspects of model correctness. Briefly put, verification answers how close the model code is in relation to the written description of a problem while validation answers how close the model is in representing the reality of the problem being considered (Refsgaard and Henriksen, 2004). While code verification can be tackled methodically by direct comparison, model validation is a problem more difficult to surmount since it requires interpretation and judgement.

Code verification is a continuous activity carried out privately by the model builder during the coding phase (Balci, 1998) as it helps with the management of model complexity and it also enables good code quality to be produced. To introduce an independent view of the code, a formal code walkthrough is usually conducted on completion of the model code. The outcome of verification is that the code of the model satisfies the written specification.

The generally accepted definition of model validation is the "*substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model*" (Schlesinger *et al.*, 1979). From this, the notion is that model validity is not absolutely exact as it is acceptable for its results to fall within a specified band. This is largely due to the trade-off between accuracy, or model confidence, and cost during validation testing (Sargent, 2007). A way of obtaining high model confidence is to compare model results with historical data and this can be very costly and time consuming since the model has to be executed many times to ensure that the results are statistically significant. Any revision to the model will likewise require a repetition of these rigorous and time consuming validation tests.

Booch (1994) suggests that a way of handling complexity is through abstraction. A complex and finely detailed model which may be highly accurate, can sometimes be simplified and yet provide acceptably accurate results. For instance, the stripping of an engine involving numerous stochastic sub-processes can be often abstracted into a single, simple, stochastic time delay with little loss of precision.

It should be borne in mind that in this research where two models are to be compared, the greater part of the effort should be expended on code verification so as to ensure a like-for-like comparison. This is not to say that, in such a case, model validation can be neglected as it will impact adversely on model realism and hence, credibility. However, if a comparison between the model and its problem domain is required, then model validation should be given the highest priority.

6.3.1.3 Coding the model

Initial inspection of the functional specification and the process maps (see Appendix C) indicated that an incremental, top-down implementation would be suitable for the reasons that the major functions were clearly demarcated and they appeared to be almost self-contained. The major functions – engine construction, engine operation, the OHB, the CRV, and the PSC – were first coded as minimal hierarchy blocks. They were then decomposed further to implement the finer details. Wherever common combinations of Extend™6 library blocks formed a logical function, they were grouped into a hierarchical block and saved for reuse elsewhere in the model. In order to promote portability, none of the standard library blocks, as supplied with the Extend™6 Industry package, were modified.

During model development, code verification was performed as frequently as possible so as to increase confidence that the model was functioning as described by the process maps. This was aided in a large part by switching on animation to run the model at its lowest speed. The visual feedback enabled model debugging at a high level to be carried out quickly. In a simulation run for a completed model, animation would normally be switched off to minimise execution time. Other aids for model development included the outputting of numerical quantities to *'information'* blocks, and the partitioning of the area being developed from the rest of the model by the judicious placement of *'exit'* blocks.

As each major function of the model was completed, it was validated and then integrated with the other completed functions by removing the appropriate *'exit'* blocks. The model was then tested and debugged. By the use of this incremental development methodology, confidence in the model's correctness increased as implementation progressed. When the model was completed, the *'information'* blocks which were used specifically for model development were also removed.

6.3.1.4 Model walkthrough

After the model was completed and had successfully completed several simulation runs, a model walkthrough was conducted at the SRC at Rolls-Royce, Derby. This activity was to establish that –

- The model was complete when compared with the specification set out by the process maps (see Appendix C).
- Details of the engine overhaul process and the supplied data had been correctly interpreted and implemented.
- The model was functionally identical to the agent-based model.

A visual and structured inspection of the whole model was made and differences with the process maps, the engine and overhaul data, and the agent-based model were recorded and agreed. On completion of the walkthrough, even though the model was assessed as fully functional when compared against the process maps, corrective action was needed nevertheless on three areas which were assessed as significantly different from the agent-based model. To state it simply, it was very important that the two models were as nearly identical as possible because they were to be compared against each other. The discrepancies arose because of a misinterpretation of the supplied data and a lack of clarity in the details of the specification. The shortfalls were –

- A stock keeping and ordering function was to be implemented for the OHB and CRV so that a buffer of parts was always maintained. From earlier studies made by the SRC using the agent-based model, this function contributed significantly to the overhaul turnaround time. This important requirement was not explicitly stated but was implied in the functional specification.

- Whenever engine parts were to be shipped, they were to be batched by part number and then dispatched in a single shipment to either the OHB or CRV. As transportation cost was calculated from total weight and discrete price bands, shipping the parts in batches could be a lot cheaper than shipping each part individually. This requirement was absent from the functional specification and the process maps.
- The items which made up each of the 67 engine component were to be modelled individually. This requirement was not stated in the functional specification nor could it be readily deduced from the supplied data as the adjusted numerical values bore little resemblance to real data. The consequence of meeting this requirement was that the number of unique items in the model increased by more than 48 times.

After the model was modified and dynamically tested again, another formal walkthrough was conducted to ensure that the functions implemented in both models were the same. This activity was concluded with no further corrective action needed. A common set of input data was then agreed and prepared for both models in anticipation of the model performance measurements to be carried out later.

6.3.2 Description of the model

The portions of the model shown in this section exhibit some aspects of good modelling practice which were gained while developing the model. As the model is large, it is not the intention here to display the details for the whole model at its most detailed, or *'atomic'*, level but to give an overview first and then select a branch of activity in the OHB to demonstrate its hierarchical design.

6.3.2.1 Model overview

The topmost level of the model, i.e. Level 0, is shown in Figure 6-9. The layout convention adopted by the Extend™6 modelling tool is that the model starts in the top-left corner of the page and the items flow along the connectors, generally from left to right and top to bottom.

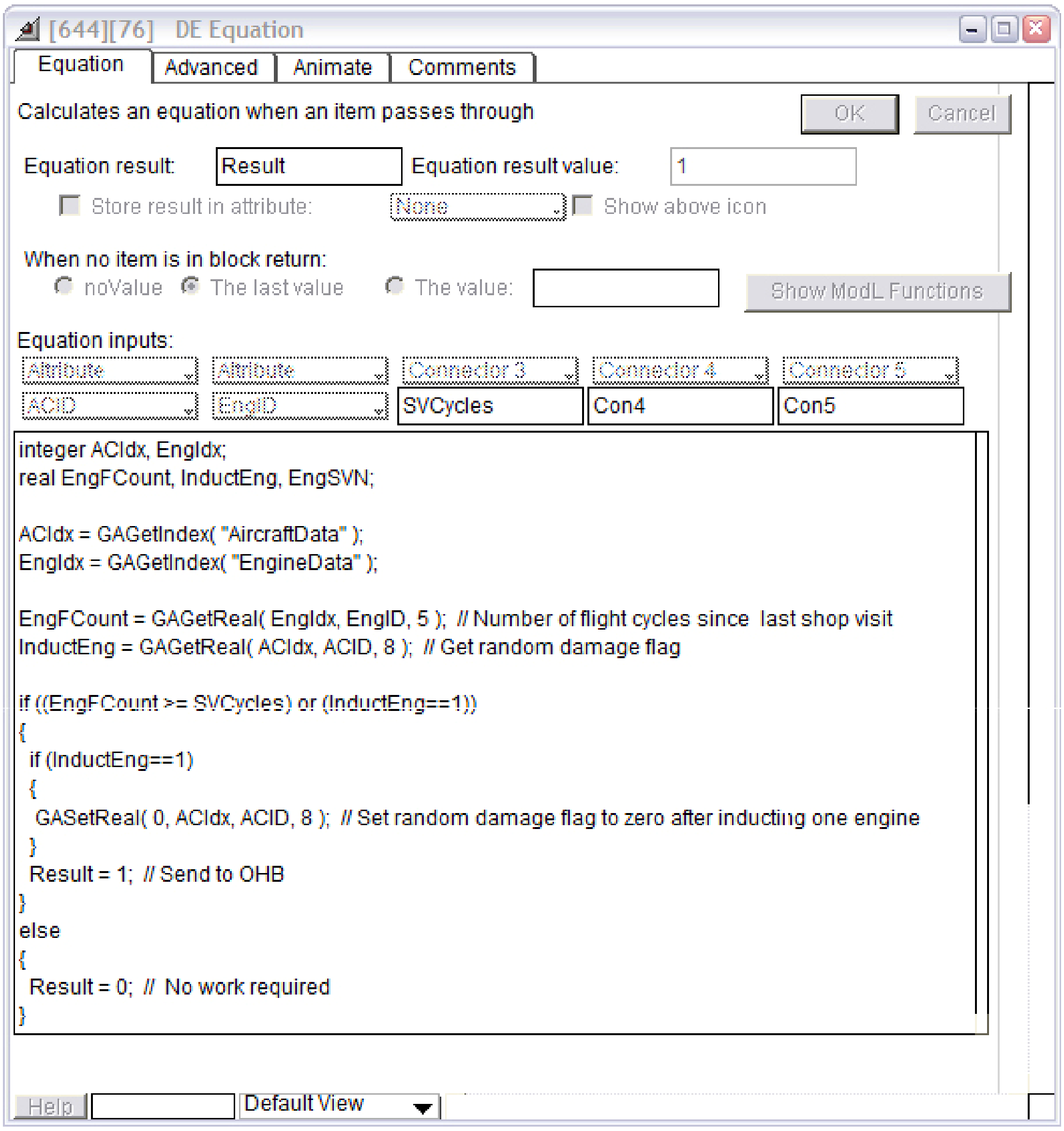
The engines are assembled in the '*Build Engines*' hierarchical block using the quantities and attributes for each of the 67 engine components supplied by SRC. Engines may be batched together to form an aircraft which enters service after a programmable delay. This forms the initialisation part of the model and is used only once for each simulation run.

The aircraft follows a fixed flying schedule which reflects the average values shown in Table 6-1 in terms of stage length and daily utilization. Flying continues until an engine either reaches the planned number of flights when it should be overhauled at the OHB or it has sustained damage serious enough to require off-wing repairs. An engine is checked after every flight cycle to determine if either of these two conditions has been satisfied. These rules, for inducting an engine into the OHB, are implemented in an Extend™6 '*DE Equation*' standard library block using ModL, the Extend™6 proprietary scripting language. The code is shown in Figure 6-7. This part of the model implements the operational phase of the engine and is used until the last engine retires from service.

A check is made to ensure that there is spare capacity in the OHB to handle the new work before engine stripping can commence. The engine modules which are assessed as requiring attention are stripped down to their individual component items. Whether inspection and sentencing takes place at the OHB or CRV depends on the maintenance policy then in force for the component. The items making up the component may be scrapped, replaced, or repaired based on a matrix of probabilities which has been compiled from operational statistics. During the rebuilding process in the OHB, the repaired items and items for refitting are returned to their engine of origin. The 3,236 unique parts for each engine are re-assembled first into components and then into modules. Subsequently, these modules are re-assembled into an engine, attached to an aircraft so that it can resume its flying service. This part of the model implements the overhaul process and constitutes the main loop of the model.

6.3.2.2 Constructing the engines

The internal structure of the ‘*Build Engines*’ Level 1 hierarchical block is presented in Figure 6-10. Adhering to the Extend™6 layout convention of modelling item flow from left to right, it shows the logical sequence of processes which are to be executed to construct an engine.



Calculates an equation when an item passes through

Equation result: Result   Equation result value: 1

□ Store result in attribute: None   □ Show above icon

When no item is in block return:
○ noValue ● The last value ○ The value:

Equation inputs:

Attribute	Attribute	Connector 3	Connector 4	Connector 5
ACID	EngID	SVCycles	Con4	Con5

```
integer ACIdx, EngIdx;
real EngFCount, InductEng, EngSVN;

ACIdx = GAGetIndex( "AircraftData" );
EngIdx = GAGetIndex( "EngineData" );

EngFCount = GAGetReal( EngIdx, EngID, 5 );  // Number of flight cycles since  last shop visit
InductEng = GAGetReal( ACIdx, ACID, 8 );  // Get random damage flag

if ((EngFCount >= SVCycles) or (InductEng==1))
{
 if (InductEng==1)
 {
  GASetReal( 0, ACIdx, ACID, 8 );  // Set random damage flag to zero after inducting one engine
 }
 Result = 1;  // Send to OHB
}
else
{
 Result = 0;  // No work required
}
```

Figure 6-7: Code to determine if an engine is to be inducted into the overhaul base

A principle followed in the course of building of the model is not to hard-code numerical values but to enable such variables to be read in, wherever possible, from an external source like a plain ASCII text file, a spreadsheet, or a relational database. Flexibility in the engine construction phase of the model has been designed in so that the number of components, modules per engine, and the number of engines per aircraft can be adjusted readily. This data-based approach allows a limited combination of engines and aircraft to be constructed without having to modify the structure of the model. In this instance, all necessary items of data for engine construction are imported from an external Microsoft Excel spreadsheet into a number of internal global arrays once only at the start of a simulation run with the goal of increasing the speed of execution.

All *'atomic'* items as well as batched-up items are tagged with automatically generated unique identifiers so that they can be identified and tracked throughout their simulation lifetimes at any point in the model. Therefore, it is essential that all items retain their unique identifiers during all batching and unbatching operations.

6.3.2.3 The overhaul base

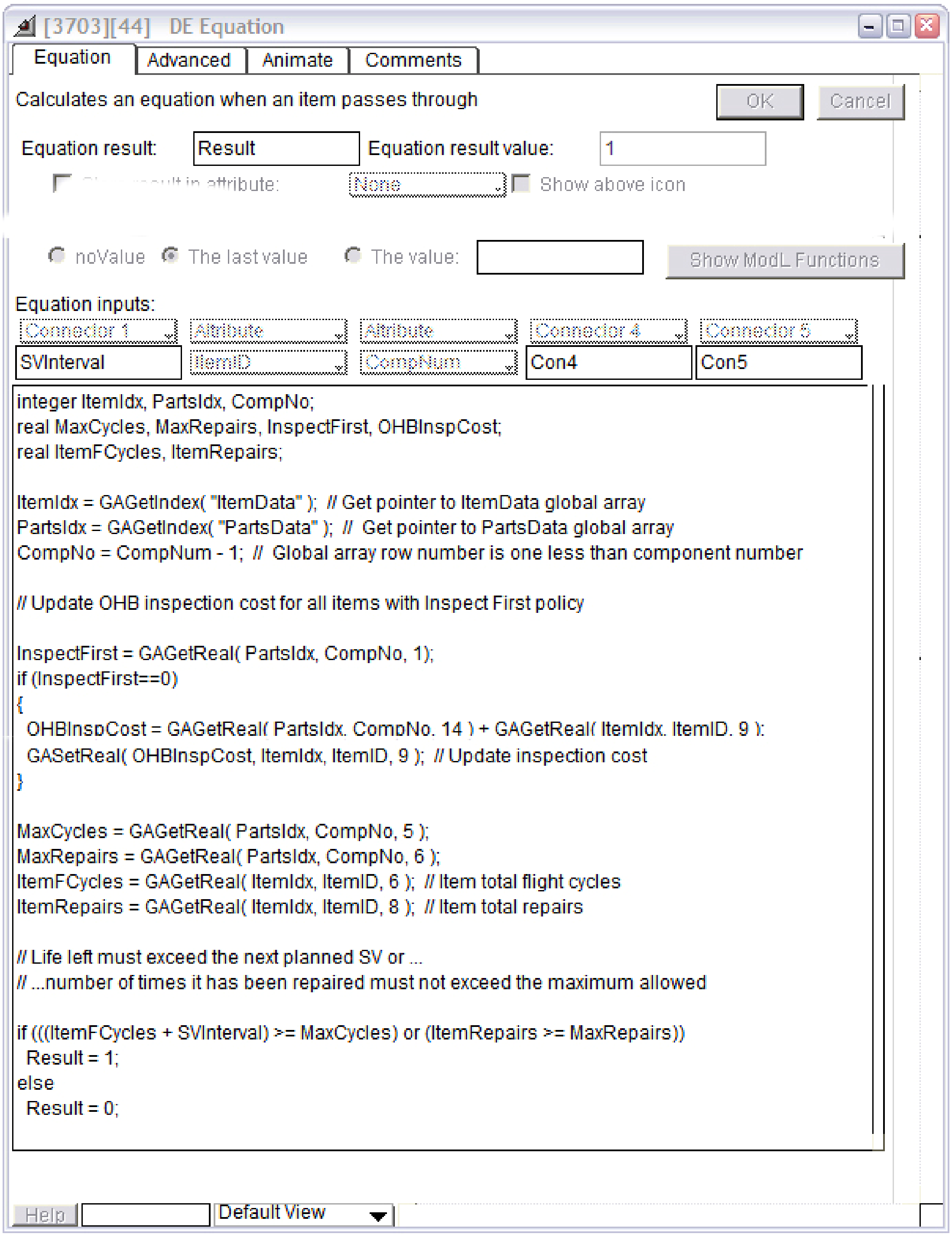
The functions of the OHB are implemented in the '*Overhaul Base Frontend*', '*Rebuild Components*', '*Rebuild Modules*', and '*Rebuild Engines*' hierarchical blocks (see Figure 6-9). The last three blocks implement the activities to rebuild an engine with its original parts as well as with new parts from the PSC and repaired parts from the CSV.

In Figure 6-11 an engine enters the '*Overhaul Base FrontEnd*' hierarchical block at the top left of the window through the '*Con1In*' input label. The engine undergoes decomposition progressively as it is first stripped down to its constituent modules and then into their components. Finally, the components are by disassembled into individual items.

Figure 6-12 and Figure 6-13 are lower level hierarchical blocks and they illustrate the procedure of unbatching the items while retaining their unique identification. This is a method which is also used in other parts of the model where unbatching occurs. Once an item is exposed, it may then be determined which of the following category it falls into –

- it may be refitted into the engine with no further work needed
- it has to be sent to a CRV for repair to be attempted
- it has to be scrapped and a replacement ordered

The last '*if*' statement in the block of code in Figure 6-8 shows how the item-scrapping rules are implemented.



Calculates an equation when an item passes through

Equation result: Result Equation result value: 1

□ Show result in attribute: None □ Show above icon

○ noValue ● The last value ○ The value:

Equation inputs:

SVInterval ItemID CompNum Con4 Con5

```
integer ItemIdx, PartsIdx, CompNo;
real MaxCycles, MaxRepairs, InspectFirst, OHBInspCost;
real ItemFCycles, ItemRepairs;

ItemIdx = GAGetIndex( "ItemData" );  // Get pointer to ItemData global array
PartsIdx = GAGetIndex( "PartsData" );  //  Get pointer to PartsData global array
CompNo = CompNum - 1;  //  Global array row number is one less than component number

// Update OHB inspection cost for all items with Inspect First policy

InspectFirst = GAGetReal( PartsIdx, CompNo, 1);
if (InspectFirst==0)
{
  OHBInspCost = GAGetReal( PartsIdx, CompNo, 14 ) + GAGetReal( ItemIdx, ItemID, 9 );
  GASetReal( OHBInspCost, ItemIdx, ItemID, 9 );  // Update inspection cost
}

MaxCycles = GAGetReal( PartsIdx, CompNo, 5 );
MaxRepairs = GAGetReal( PartsIdx, CompNo, 6 );
ItemFCycles = GAGetReal( ItemIdx, ItemID, 6 );  // Item total flight cycles
ItemRepairs = GAGetReal( ItemIdx, ItemID, 8 );  // Item total repairs

// Life left must exceed the next planned SV or ...
// ...number of times it has been repaired must not exceed the maximum allowed

if (((ItemFCycles + SVInterval) >= MaxCycles) or (ItemRepairs >= MaxRepairs))
  Result = 1;
else
  Result = 0;
```

Figure 6-8: Code to determine if an engine item is to be scrapped

In this model, this categorization is based on the probabilities applicable to their current full shop visit numbers (see list of attributes in Table D-2 of Appendix D). Where the parts need to be dispatched to a CRV for repair, their destination is determined first; they are then batched by component number; their batch weight calculated; their shipping costs calculated from a scale of charges; and finally, the shipment is made. The attributes for each item are updated at the lowest level of the model using program code (see Figure 6-14) written in the ModL language. The attributes are stored in global arrays as it is easier and faster to manipulate them in code than through a combination of Extend™6 standard library blocks.

6.3.2.4 Shipping items

In the course of a simulation run, it is required that various quantities of different engine items are batched together by component number and shipped to their respective destinations. Shipment occurs once each day. Figure 6-15 shows how this has been implemented using Extend™6 standard library blocks to schedule, queue, and group engine items into component-number batches. This hierarchical block, labelled ‘*Batch up items for shipping*’, implements a common process which is also used elsewhere in the model.

6.4 Results and evaluation

After successfully verifying that both the DEM and the ABM possess the same functions, the next step is to make a quantitative comparison between them. The following sub-sections describe the collection of code metrics and the measurement of model runtimes. The results are then presented and evaluated.

6.4.1 Metrics to be collected

The metrics which may be used to characterise the models are size, complexity, and connectedness. As described previously in Chapter 5, program size may be measured as LOC, number of methods, or number of classes. They are almost invariably compiled to monitor the progress of software projects and because they have been used so widely they may be considered as valid measures for program size. The models have been implemented in different programming languages (Java and ModL), but Table 6-5 lists three language constructs which may be treated as equivalent because of the way they are used. In the ABM, methods are local modules which are invoked within a class and in the DEM, procedures and handlers are local modules which are invoked within a library block.

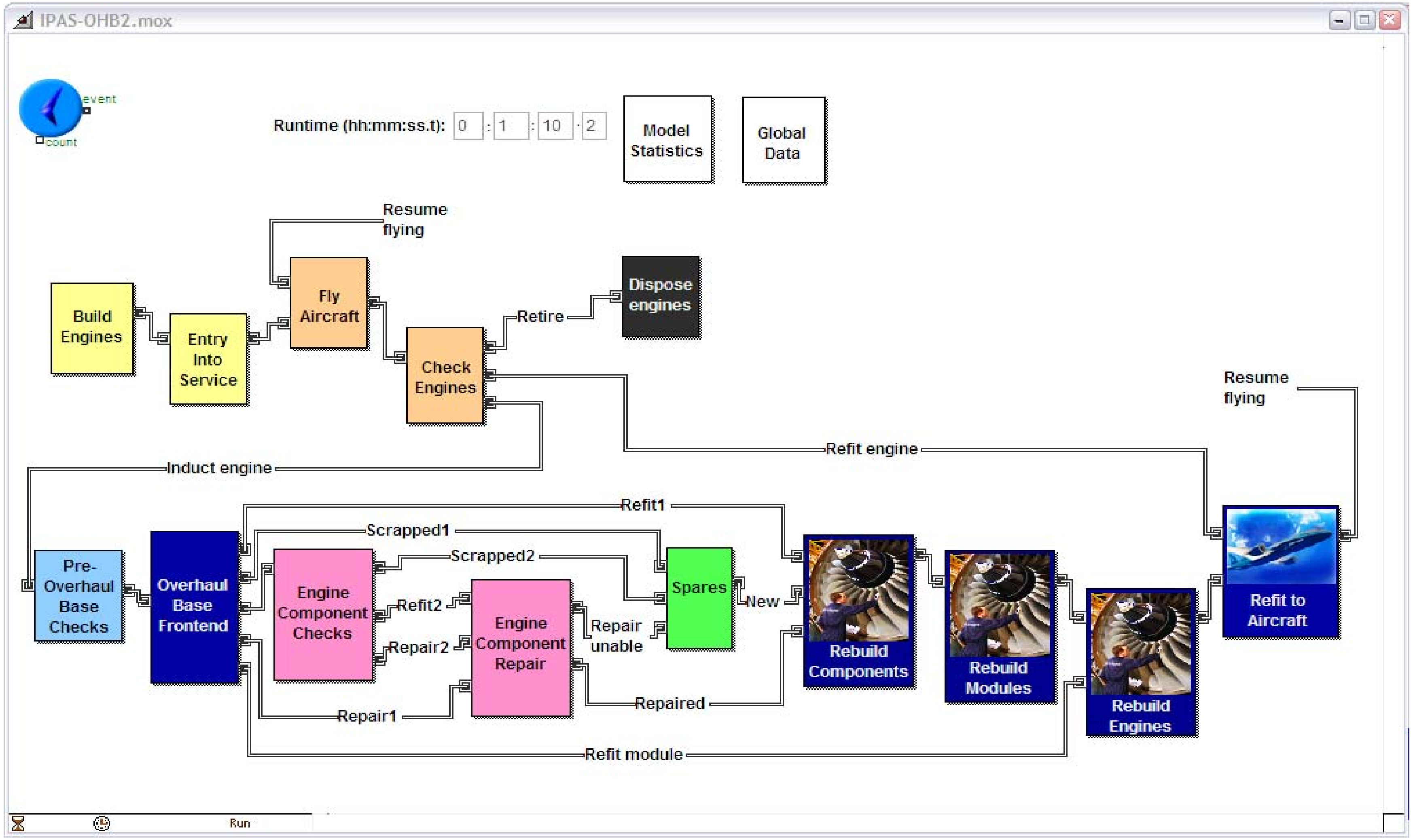


Figure 6-9: Top level (Level 0) of traditional discrete-event model

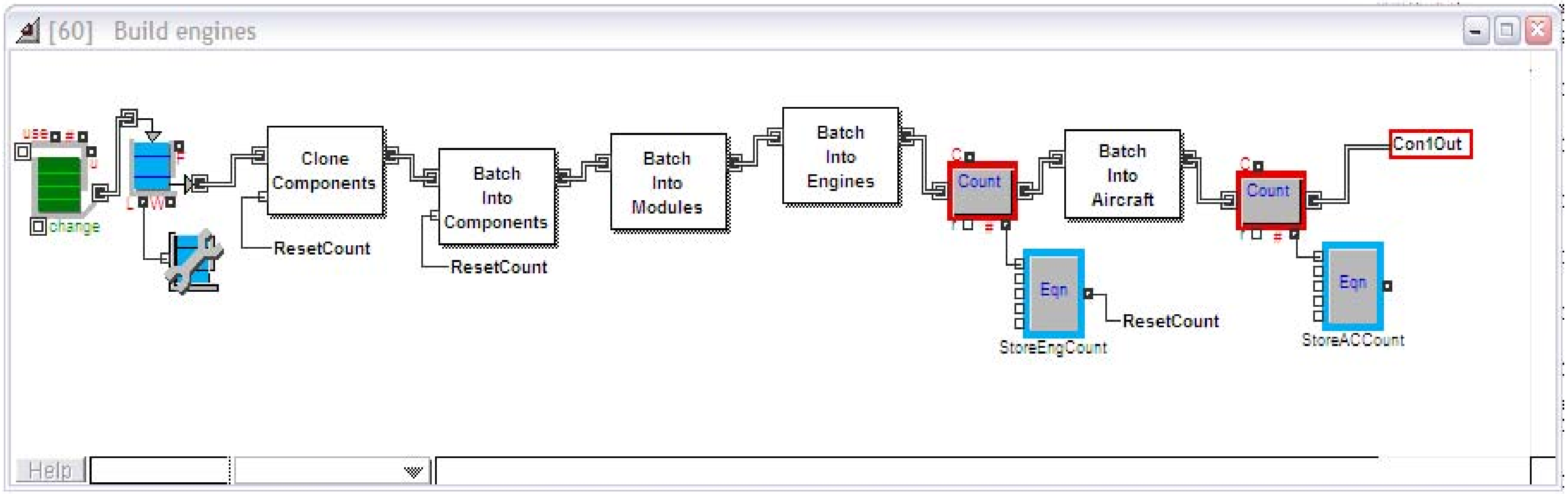


Figure 6-10: Level 1 hierarchical block – Build Engines

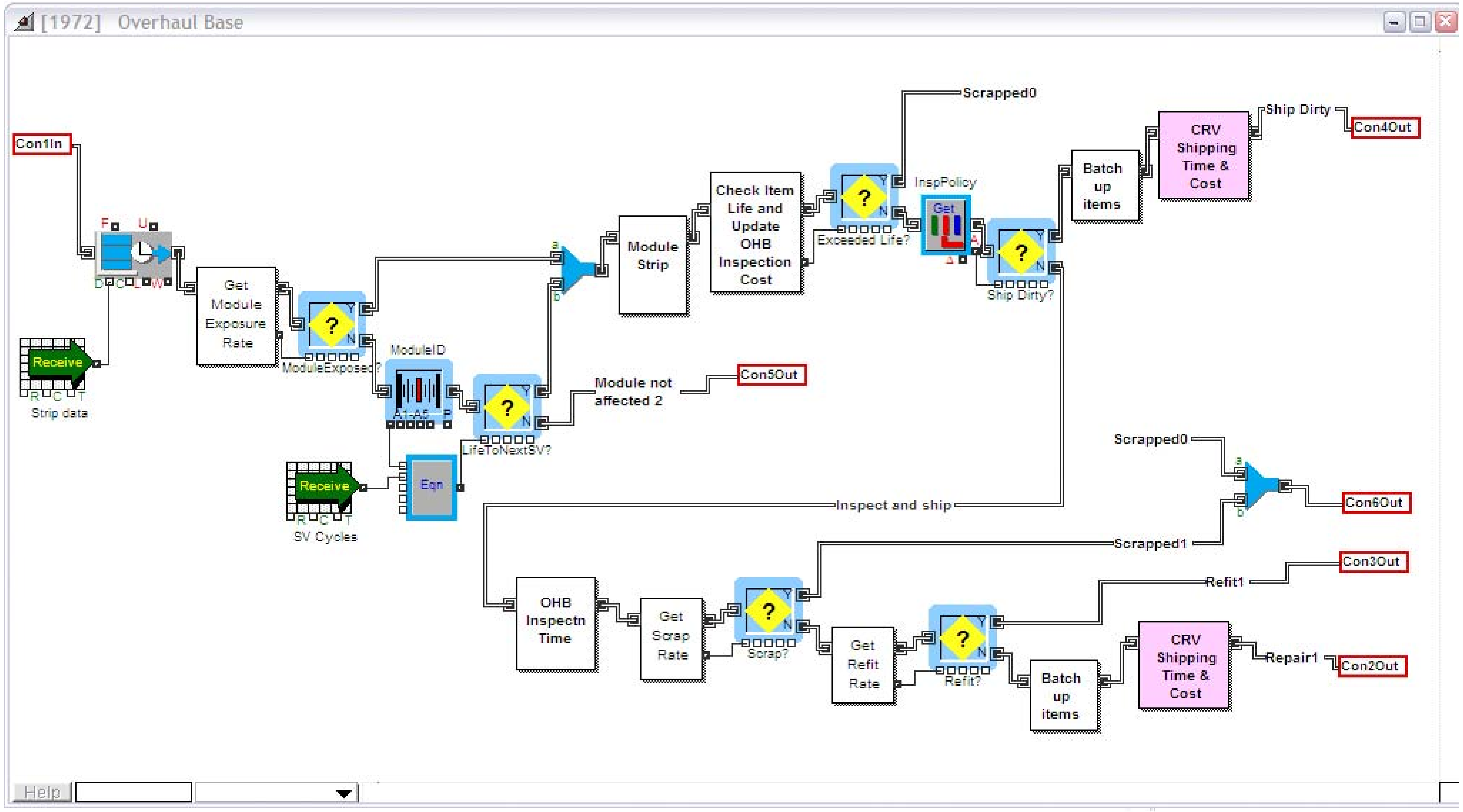


Figure 6-11: Level 1 hierarchical block – Overhaul Base

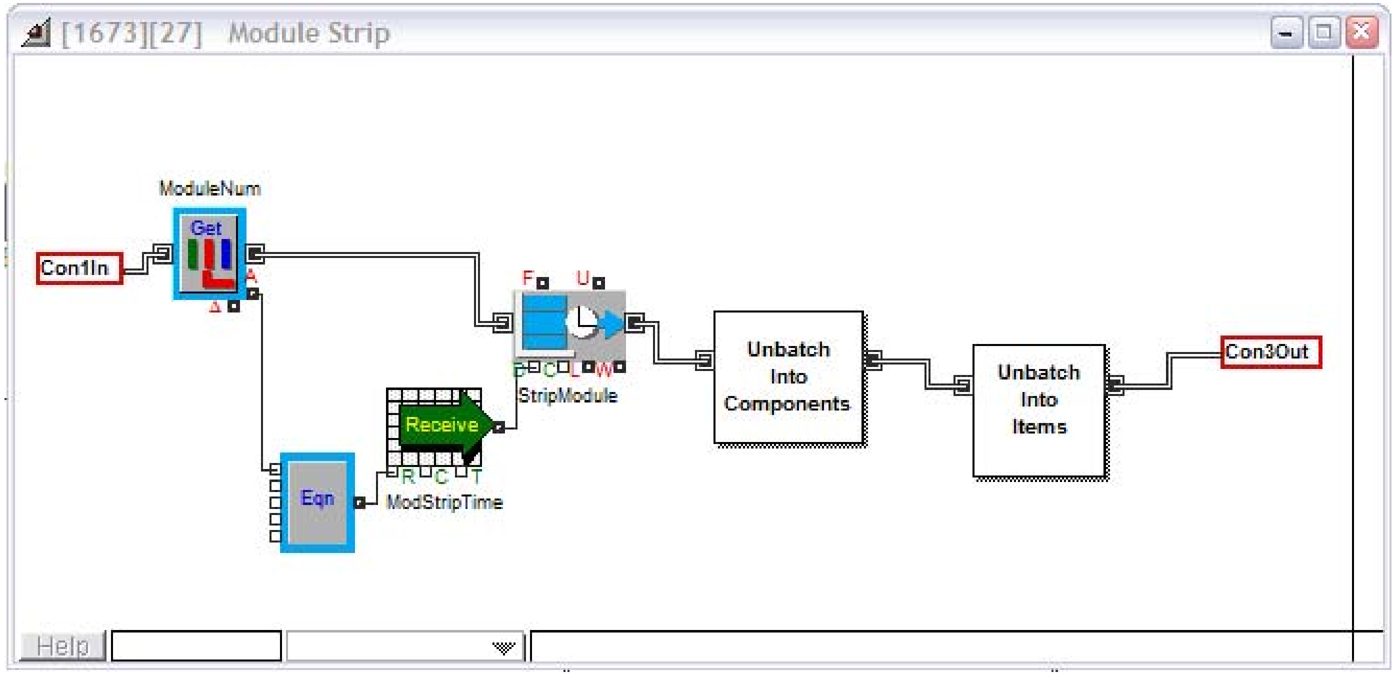


Figure 6-12: Level 2 hierarchical block – Module Strip

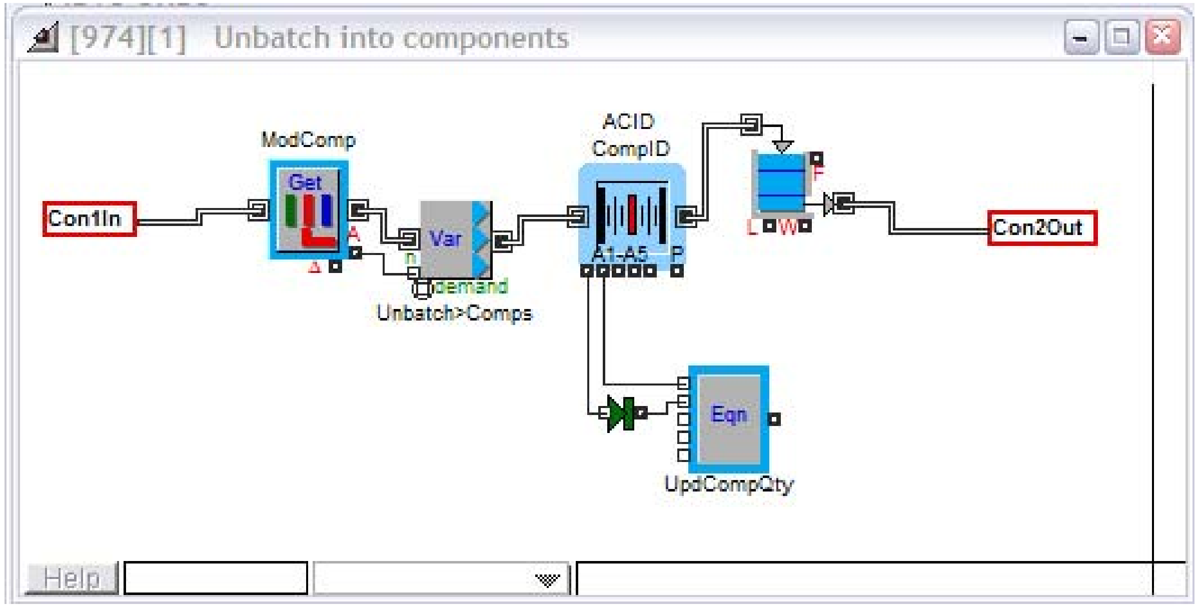
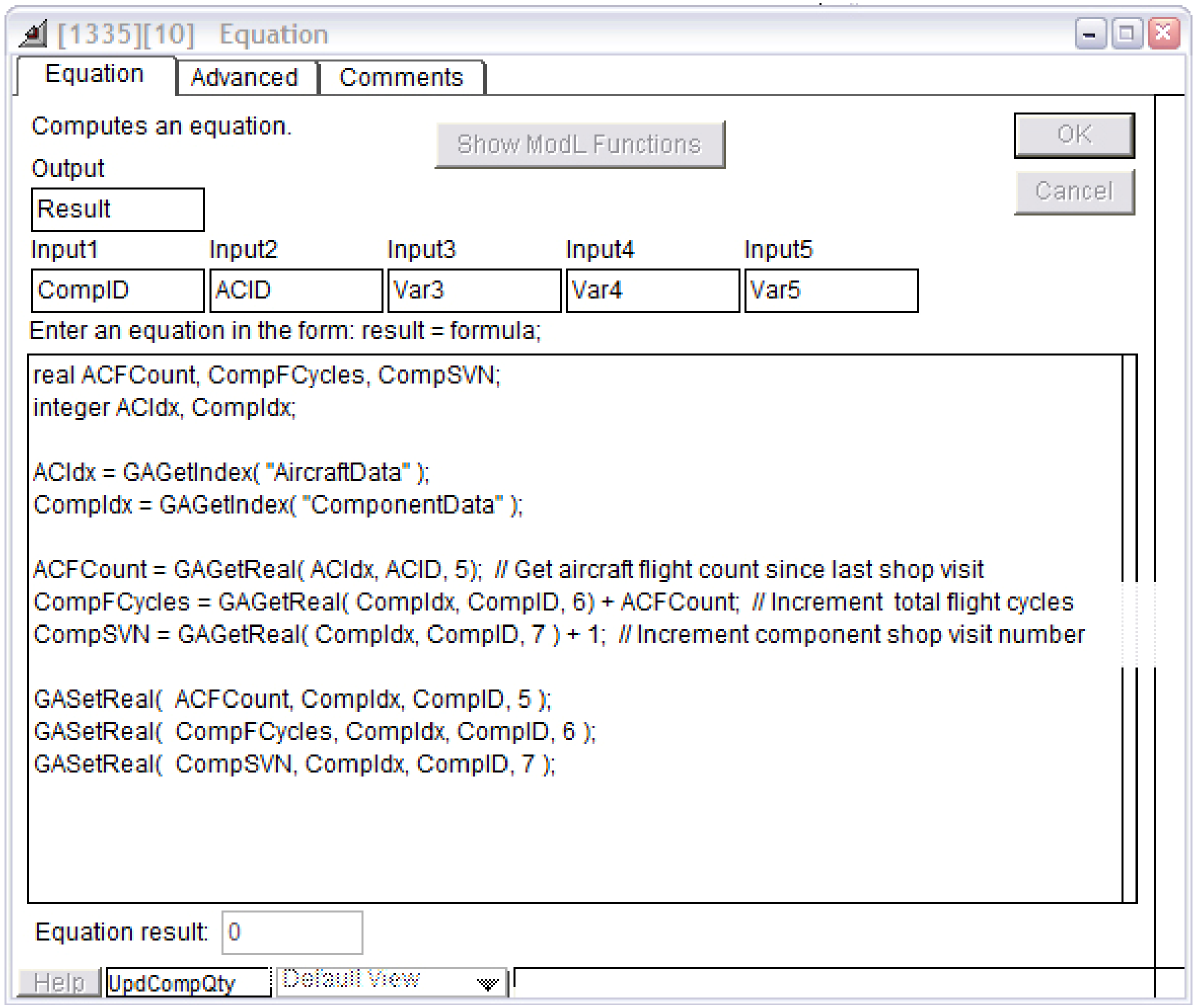


Figure 6-13: Level 3 hierarchical block – Unbatch Into Components



Computes an equation.

Output: Result

Input1: CompID
Input2: ACID
Input3: Var3
Input4: Var4
Input5: Var5

Enter an equation in the form: result = formula;

```
real ACFCount, CompFCycles, CompSVN;
integer ACIdx, CompIdx;

ACIdx = GAGetIndex( "AircraftData" );
CompIdx = GAGetIndex( "ComponentData" );

ACFCount = GAGetReal( ACIdx, ACID, 5);  // Get aircraft flight count since last shop visit
CompFCycles = GAGetReal( CompIdx, CompID, 6) + ACFCount;  // Increment  total flight cycles
CompSVN = GAGetReal( CompIdx, CompID, 7 ) + 1;  // Increment component shop visit number

GASetReal(  ACFCount, CompIdx, CompID, 5 );
GASetReal(  CompFCycles, CompIdx, CompID, 6 );
GASetReal(  CompSVN, CompIdx, CompID, 7 );
```

Equation result: 0

UpdCompQty

Figure 6-14: Level 4 block containing ModL code – Update Component Quantities

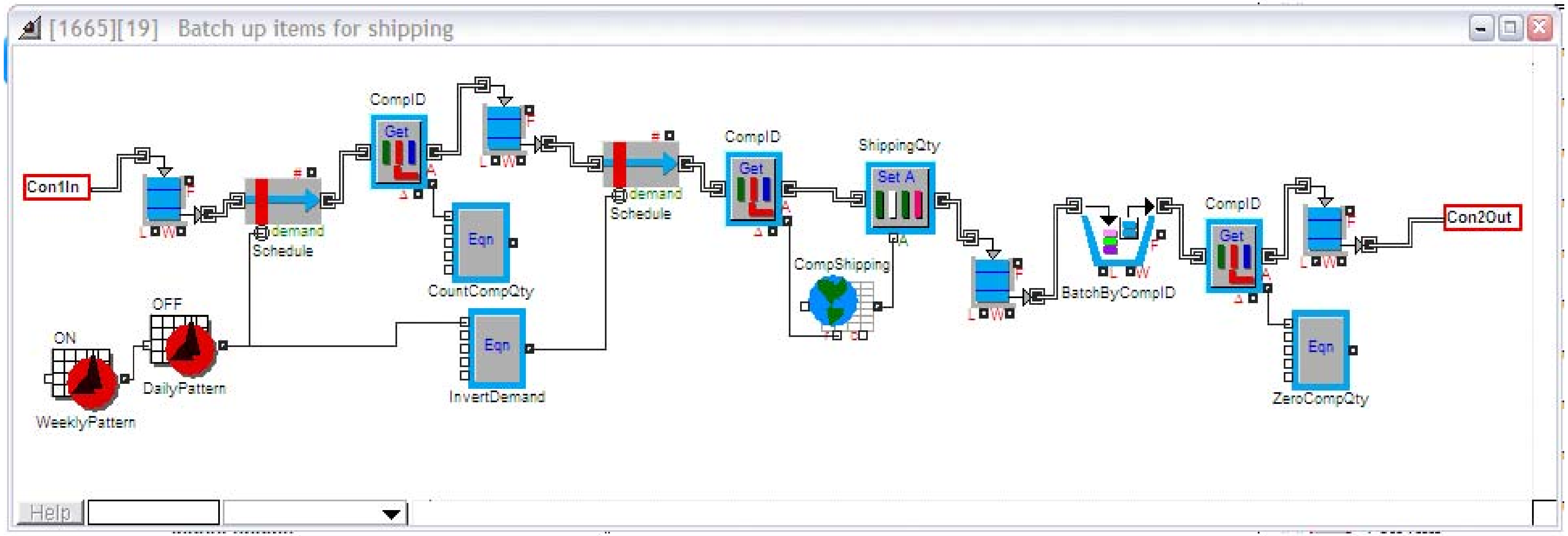


Figure 6-15: Level 2 hierarchical block – Batch up items for shipping

The LOC metric may be too close to the human programmer in the sense that the effect due to programmer idiosyncrasy may result in significant variation in LOC. A count of the number of methods or procedures is a readily accessible metric and probably provides some insulation from the effect of programmer variation. A count of the number of classes or blocks can be achieved by inspection but its granularity may be too coarse for meaningful comparison.

Table 6-5: Equivalence of constructs in object-oriented and procedural languages

ABM / Object-oriented language	DEM / procedural language
Statement	Statement
Method	Procedure or handler
Class	Model library block

Complexity for a method or procedure is calculated using the McCabe Cyclomatic Number. Similarly, the measure of complexity for a class or library block is the Weighted Methods per Class which is just the arithmetic sum of the complexity of all methods in the class or procedures and handlers in the library block. This metric is used to estimate the degree of structural complexity for both models.

Connectedness for a model is assessed as the worst method of coupling between classes or library blocks as set out in the scale given in Section 5.4.4.

Collectively, these metrics enable the software quality of maintainability to be determined for each of the models. The metrics also indicates a model’s degree of understandability, modifiability, and testability since they are the factors which contribute towards the quality of maintainability (see Section 5.3.1).

6.4.2 Collecting the metrics

The DEM was partitioned into five subsystems which corresponded to five of the agents in the ABM. This was achieved without difficulty as the top-down hierarchical implementation of the DEM followed the modules set out in the ABM. These agents or subsystems are labelled as Subsystems A to E in Table 6-7 and Table 6-8.

Metrics for size and complexity had to be compiled manually for the DEM as no automated code measurement tool could be found for the Extend™6 ModL programming language. For the ABM, a plugin for Eclipse was used to extract the required metrics.

6.4.2.1 Adjustments to DEM code metrics

The LOC metric was refined to include only executable statements thus excluding data declarations and comments. Also, in an attempt to compare like with like, code used specifically in Extend™6 for model construction were excluded. For example, code for handling parameter setting, block creation, and block report generation were left out in the statement counts. Lastly, the number of statements were factored using data from Jones (1996) to allow for the difference in expressive power of the two programming languages. Jones' database of programming languages showed that the average number of statements per function point for Java was 53 while that for C was 128.

Also, only unique blocks were counted in each of the DEM subsystems. This adjustment was made because, in a visual programming system like Extend™6, each additional use of a library block meant an explicit, repeat occurrence of all the code for the block at compile time. In contrast, in a Java program, each additional use of a class is accomplished in a single statement which instantiates the class code only at run time. Counting only unique Extend™6 blocks, and hence procedures and LOC, results in a more equitable comparison.

6.4.2.2 Complexity

The MCN is the number of linearly independent paths through a program and it therefore indicates the upper bound of test cases required to ensure full test coverage. From empirical evidence, McCabe (1976) suggested an MCN greater than 10 pointed to reduced program modifiability and testability and hence increased risk of error when making changes to a program. Also, Grady (1994) concluded from a large-scale study that an upper limit of 15 should be enforced. Table 6-6 shows typical ranges of MCN for programs of different levels of complexity.

Table 6-6: Program complexity and maintainability (SEI, 2000)

Cyclomatic complexity, MCN	Program complexity and risk evaluation
1 to 10	Simple program, low risk
11 to 20	More complex program, medium risk
21 to 50	Complex program, high risk
Greater than 50	Untestable program, very high risk

Although MCN is usually used for procedural languages, it is nevertheless a valid measure for object-oriented languages at the lowest method level. Figure 6-16 shows that the cyclomatic complexity, or structural complexity, for both models is well managed. This is markedly so in the ABM as only 0.24% of the modules had a MCN greater than 20 while it was 4.13% for the DEM. Further, the maximum MCN for an ABM module was 27 while it was 49 for the DEM. Lastly, neither model had any module with a MCN greater than 50, i.e. a module considered too complex for rigorous testing.

The *WMC* metric, suggested by Chidamber and Kemerer (1991), is the sum of the weights of all the methods within a class. The weight of each method is represented by its MCN.

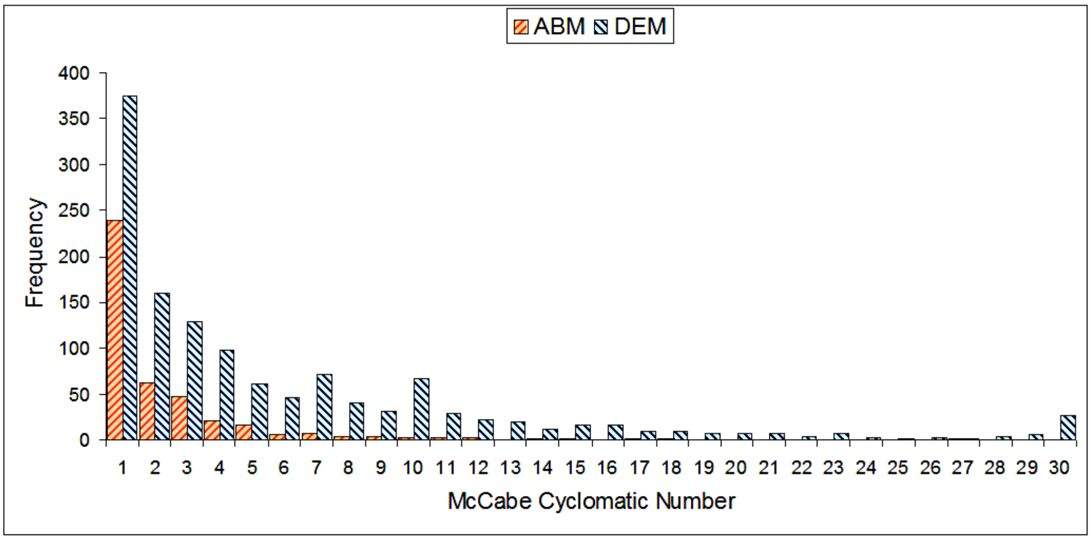


Figure 6-16: Distribution of MCN in the ABM and the DEM

The code metrics collected for both models are summarized in Table 6-7. The quantities for the DEM have been adjusted in the manner described in Section 6.4.2.1 as the model was implemented using a different programming language from that of the ABM. To present a comparative overview of these metrics, the multiples of DEM to ABM quantities are shown in Table 6-8. It may be noted that the quantities in this table illustrate the stark differences between the two models as all the values are much larger than unity. They indicate that the DEM is significantly larger and more complex than the ABM in terms of program code.

Table 6-7: Code metrics for the ABM and the DEM

Sub-system	Number of executable statements, *LOC*		Number of methods or procedures		Number of classes or modelling blocks		Weighted methods per class, *WMC*	
	ABM	DEM	ABM	DEM	ABM	DEM	ABM	DEM
A	740	2685	70	304	5	16	35	135
B	187	1838	15	211	2	11	19	129
C	130	1516	12	177	2	10	12	118
D	487	2928	28	344	2	19	55	122
E	88	1843	8	197	3	11	5	128

Table 6-8: Multiples of DEM to ABM metrics

Sub-system	Number of executable statements, *LOC*	Number of methods or procedures	Number of classes or modelling blocks	Weighted methods per class, *WMC*
A	3.63	4.34	3.20	3.86
B	9.83	14.07	5.50	6.79
C	11.66	14.75	5.00	9.83
D	6.01	12.29	9.50	2.22
E	20.94	24.63	3.67	25.60

The process for conducting a Delphi estimation session to obtain high quality group opinion as described by Stellman and Greene (2005) was employed with a minor modification, i.e. the use of email as the medium of communication, to compare the maintainability aspect of both models. Details of this exercise are shown in Appendix E. As the ABM code contained information which was commercially sensitive, only three software and modelling experts were allowed to view it. The results of this exercise are displayed in Table 6-9. When considering model maintainability, i.e. consisting of understandability, modifiability, and testability, from the view of a software engineer or model implementer, the ABM is more maintainable on balance. The result reinforces the tendency shown by code metrics in Table 6-8. It was noticeable that the DEM was judged to be more readily understood than ABM by a design engineer for the reason that a traditionally trained engineer will tend to have a better mental grasp of a problem if it was framed as processes rather than activities. This is a result which is not apparent by considering code metrics alone.

The ABM was also considered to be more suitable for modelling at a high level of abstraction and this view is supported by the score that the ABM is easier to understand by someone in business and commerce but not so readily by a design engineer or software engineer where attention to technical detail is important. However, to implement an ABM, which was viewed by the Delphi session participants as marginally simpler than a DEM in concept, required someone with a higher level of formal education. The reasons were that the current availability of modelling tools for implementing an ABM required a higher level of expertise and that formal training in agent technology occurred only at postgraduate level.

Table 6-9: Summary of results of Delphi estimation of ABM and DEM

	ABM	DEM
Understandability of model as considered by a software engineer or model implementer	+	++
Understandability of model as considered by a design engineer (e.g. a jet engine designer)	-	+++
Understandability of model as considered by a person in business and commerce	++	+
Modifiability of one or more sub-systems in the model	+	-
Testability of one or more sub-systems in the model	--	--
Suitability of modelling at high level of abstraction	++	+
Suitability of modelling of very fine details	-	++
Ease of expansion by adding one or more existing sub-system	+	++
Level of formal education needed to implement the model	+	-
KEY: +/- nominally; ++/-- moderately; +++/--- very		

6.4.2.3 Coupling and cohesion

Agents in the ABM interact not by invoking each other directly but by message passing via the JADE agent environment. The messages are processed by the receiving agents in their own time. Further, the messages may sometimes contain a small number of integer, floating point, or character string parameters. For the additional reason that the message passing is asynchronous, coupling between agents may be judged to fall in between stamp coupling (relation R_2 in Table 5-1) and data coupling (relation R_1). As a software system, the ABM may be classified as loosely coupled.

Inspection of the ABM code showed that the methods within a class implemented a single well-defined function with small number of parameters passed between methods. As set out previously in Table 5-2 the ABM code may be described as highly cohesive since it displays functional cohesion.

A visual inspection of the DEM library block code revealed that while most procedures within a block displayed data coupling (relation R_1), the undesirable feature of coupling through a small, fixed number of system global data (common coupling relation R_4 in Table 5-1) was nevertheless commonly used for control as well as for communication between blocks. Because of this lower classification, the DEM subsystems may be classified initially as tightly coupled. However, this drawback is mitigated by the fact that it is a characteristic of a stable and well tested modelling package and not of the DEM. It has a direct and adversely effect on the package developer but not the model implementer. It is reasonable to classify the DEM less severely than R_4 and considerably closer to R_1.

All procedures and handlers in the DEM standard library blocks performed single well-defined functions. Therefore, the DEM exhibit functional cohesion (see Table 5-2).

The qualities of inherently low coupling and high cohesion may make the agent approach marginally more suitable for distributed modelling than traditional discrete-event modelling but it may be accepted that both modelling approaches are suitable for distributed modelling.

6.4.3 Model runtimes

The performance of both models was measured in the same computing environment under identical model initial conditions. In the case of the ABM, the complete initial engine fleet data was prepared by a separate Java program, stored in an XML file, and subsequently used for initialising the ABM. In contrast, initialisation of an engine fleet for the DEM was carried out by the DEM itself one engine at a time using a Microsoft Excel spreadsheet which contained the data for a single engine. This method was adopted because of its simplicity in the main and also the assumption that all new engines in a fleet may be reasonably expected to be identical. However, initialisation carried out in this manner took up a significant portion of the DEM runtime while the time required for ABM initialisation was negligible.

Two timing marks were needed to measure the elapsed time for a simulation run – the initial timing mark was set when the first engine entered service (the service entry point was described in Section 6.3.2.1 in the overview of the DEM model) and the final timing mark occurred when the fifth shop visit of the last engine was due. This was done to exclude the effect the initialisation methods had on the models. The runtimes between the timing marks for both models using different engine fleet sizes are presented in Figure 6-17. The effect of DEM initialisation is evident as it forms about 17% of total elapsed model runtime.

The maximum ABM engine fleet size was 220 as that was the largest number of engines which could be accommodated by the Java runtime environment without an abnormal termination through an '*out of memory*' error. The largest DEM engine fleet was selected as 310 because it involved just more than 10^6 individual engine items. Such a quantity was thought to be representative of real engine fleets and it was not necessary to expand the fleet beyond that. Although it was possible to complete a DEM run with a larger fleet, it took too long for practical purposes because of intensive memory swapping between the main memory and the virtual memory in the hard-disk drive.

For both models, it is possible to continue enlarging the engine fleet size by the addition of computer memory until the limit imposed by the operating system is reached. In the case of the Microsoft Windows 32-bit operating system used for this research, the Windows XP operating system limit is 3.3GB of main memory. It may be estimated by linear extrapolation that a fleet of about 360 engines can be accommodated by the ABM before it fails with an '*out of memory*' error.

The ABM is a time-driven model and, as shown in Figure 6-17, its runtimes with respect to fleet size was almost constant up to about 100 engines but beyond this the gradient of this plot increased noticeably. For fleets larger than 100 engines, the OHB capacity limit of five engines meant that it could not finish overhauling all the engines before the first ones became due for another overhaul. This is illustrated in Figure 6-18 and Figure 6-19 where the peaks of computer processor utilisation coincide with the peaks of overhaul activity. In the former figure, the OHB is able to satisfy the demand to complete the required work within the agreed timeframe and there are lengthy intervals of relative inactivity between shop visits. For a time-driven model under such conditions, the time taken to complete a simulation run is expected to be constant. In the latter figure, OHB activity is kept at a high level throughout the simulation run when work occurring at the end of the previous shop visit runs into the start of the next. As the engine fleet size increased, the longer the OHB pre-overhaul queue became. Consequently, the time when an engine was out of normal airline operation was extended, thus prolonging the elapsed time of a simulation run. It should be remembered that the criterion for terminating a simulation run is satisfied when the last engine is due for its fifth shop visit and not after a fixed simulated period.

The DEM is an event-driven model with events managed by a variant of the calendar queue (described previously in Section 3.3.1.3). Theoretically, the best performance which may be obtained from such a priority queue algorithm is $O(1)$. Based on this idealised relationship, the run times will be in direct proportion to engine fleet size. In practice the performance of the calendar queue is likely to fall between $O(1)$ and $O(n)$ (see Table 3-2), where n is the length of the queue. Hence, it is reasonable to expect the model run times with respect to fleet size to be longer than those predicted by a straight-line relationship.

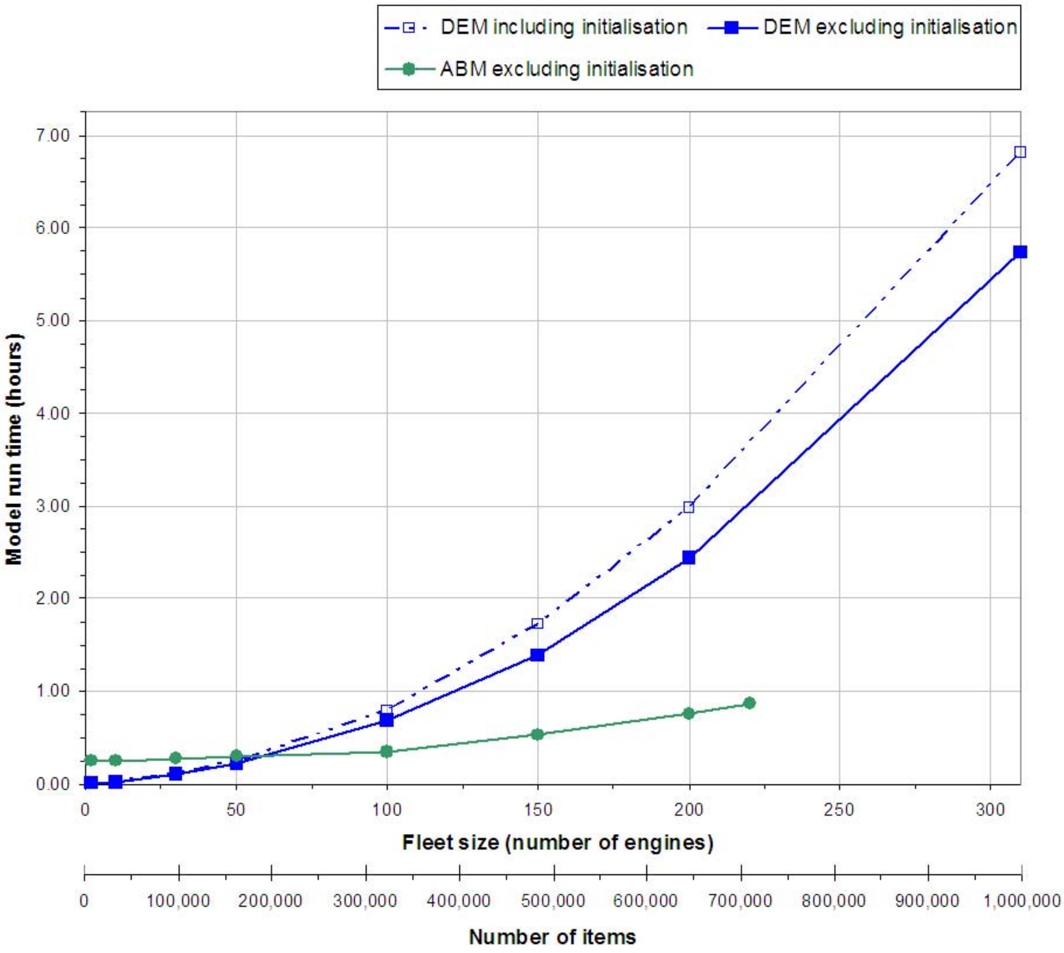


Figure 6-17: Runtimes for DEM and ABM for different engine fleet sizes

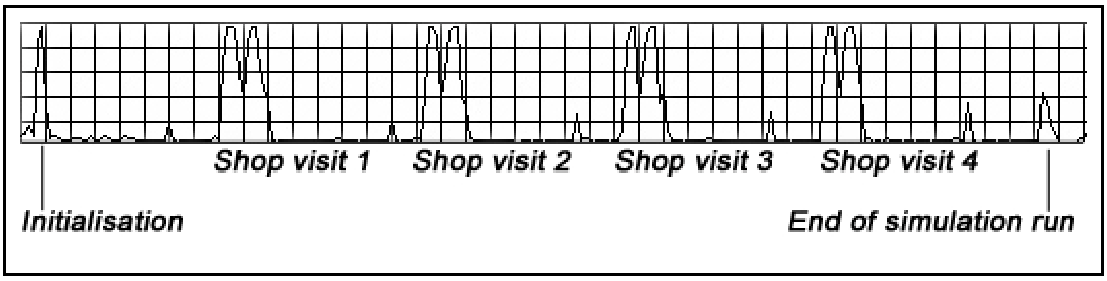


Figure 6-18: ABM CPU load for a 10-engine fleet

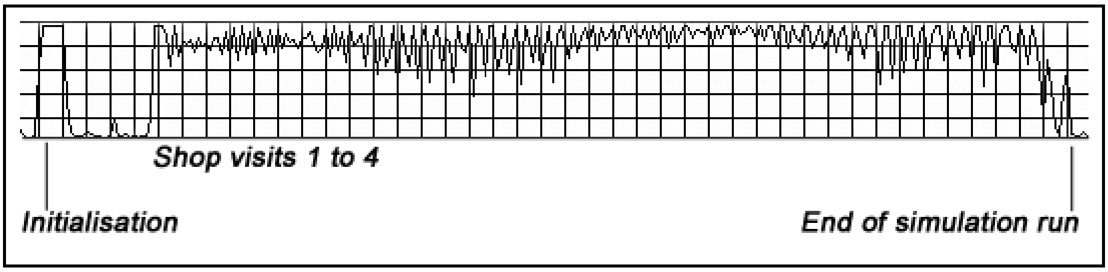


Figure 6-19: ABM CPU load for a 100-engine fleet

6.5 Summary

The method of investigation selected for comparing agent-based modelling and traditional discrete-event modelling is the case study. Neither the formal experiment nor the survey is appropriate for investigating the differences. A suitable non-trivial scenario was provided by the RR Trent 800 global repair operation and a validated ABM, which was used as a design tool, existed for that scenario.

The case study scenario was described and particular attention was given to the engine maintenance process, the ABM, and the data required for modelling the scenario. The ABM and the data were provided by SRC.

To enable a comparison to be made between the two modelling paradigms, a functionally identical traditional discrete-event model had to be built. It was built using the Extend™6 modelling tool and validated against the process maps, textual specification, the ABM, as well as by formal code walkthroughs. Lastly, the ABM and DEM were run using a common set of input data and their outputs compared.

The Extend™6 modelling tool enabled the DEM to be implemented using the top-down approach and this hierarchical structure emulated the topology of the ABM. Consequently, sub-systems within the DEM could be clearly identified for comparison with agents in the ABM.

The comparison was carried out using code metrics as well as human expert input in the form of Delphi sessions – a method advocated by Boehm (1981) in his COCOMO software project costing methodology. This approach, which required the additional judgement of human experts, was reinforced more recently by Welker (2001).

Code metrics for model size and complexity all pointed to the superiority of the ABM relative to the DEM in that it was more understandable, modifiable, and testable. The good maintainability of the ABM is indicated both by its smaller code size and lower structural complexity. These results were supported by the independent assessment of the Delphi sessions.

The Delphi results also indicated that the process-centric DEM may be more suited to the way traditionally trained design engineers work. Fine details of engineering processes may be better modelled as DEMs rather than ABMs. Moreover, the participants were of the view that implementing an ABM required greater software skills.

For large engine fleets, the runtime performance of the ABM exceeded that of the DEM. However, the ABM required more computing resources and in the same computing environment managed to complete a simulation run with a fleet size 29% smaller than that managed by the DEM.

Chapter 7
An Agent-like Discrete-event Model

7.1 Introduction

In this chapter, the motivating reasons for emulating an ABM are discussed first and the definition of an agent is revisited. Following that, consideration is given to the management of complexity in software and finally, drawing from the knowledge gathered and results obtained in this research, a novel method of structuring a DEM is presented. A model is built using this new architecture and results obtained to compare it against an ABM and a traditional DEM.

7.2 Rationale for making a DEM agent-like

In the following sub-sections the reasons for making a traditional DEM more like an ABM are presented and discussed. They are derived from the results of the case study.

7.2.1 Starting from the discrete-event modelling paradigm

It was considered more appropriate to make a DEM more like an ABM than *vice versa*. In the former instance, the effort would be logical, reasonable, and worthwhile since the boundary of usefulness for the discrete-event modelling paradigm would be extended. Making an ABM more like a DEM defies reason as that, in effect, shrinks the boundary of usefulness for the agent-based modelling paradigm. Another reason for basing the model on the discrete-event paradigm is that it is a mature, established, and well-understood approach which has a large body of active practitioners. Finally, there are a number of excellent COTS tools to support and speed up model implementation while agent-based modelling in its current state of development still depends almost entirely on text-based, manual coding.

7.2.2 Code size and complexity

In the case study described in Chapter 6, the quantitative results described in Section 6.4 pointed to the benefits that ABM had over the DEM. The significantly smaller code size, in terms of lines of non-comment code, as well as the lower complexity, in terms of weighted method per class (see Table 6-8), are metrics which indicate less mental effort required, and hence lower cost to be incurred, in the software maintenance phase. Although the Delphi results showed the DEM to have a higher understandability than the ABM by model implementers (see Table 6-9), nevertheless both models were considered to be easy to understand. ABM code is easy to understand and easier to modify than DEM. Therefore modifications to an ABM stand a higher probability of being implemented correctly in a shorter time.

As software maintenance costs typically varies between 50% to 65% of overall lifetime costs (Somerville, 2001) and enhancement and adaptation activities can make up more than 80% of the maintenance effort (Krogstie *et al.*, 2006), having higher maintainability is a desirable goal. The difficulty of understanding any program code may be alleviated to some extent by adhering to good programming practice such as the consistent use of descriptive names for variables, the indenting of code sections, and the reuse of modules. Although such practices can result in code which is read more easily, they are not likely to result in code which is structurally less complex. It is mainly the lower structural complexity which makes the ABM more maintainable and therefore the more attractive modelling paradigm.

7.2.3 Model performance and multi-threaded operation

In contrast to the DEM which is event-driven, the ABM is time-driven. The agent model runs as a multi-threaded computer process with each agent executed as a separate concurrent thread. This is in keeping with one of the three founding concepts of agent technology, i.e. OO programming and concurrent object-based systems (Jennings *et al.*, 1998). The necessity of multi-threaded execution of multi-agent systems was also forcefully reiterated in Wooldridge and Jennings (1999) where failure to do so was identified as a serious pitfall to avoid.

Figure 6-5 shows that the ABM is relatively insensitive to computing load and it holds a significant runtime advantage over the DEM as the demands of computation increased with increasing engine fleet size. In other words, the ABM is more scalable, where scalability may be described as the ability of the model to continue performing acceptably even when the workload has been increased greatly. Despite this advantage, it should be noted that under identical runtime conditions, the ABM managed to complete a normal simulation run with an engine fleet which was considerably smaller than that managed by the DEM. While the ABM experienced a sharp engine fleet cut-off size beyond which a simulation run could not be completed normally, the DEM continued with much larger fleet sizes even though they took much longer to complete their runs.

While the scalability of the ABM may be largely attributed to its time-driven design and fleet operation scenario, its concurrent multi-threaded execution also contributes towards its efficiency in operation. Multi-threading seeks to exploit parallelism at instruction level and the simplest variant is block multi-threading. In such a scheme, a thread runs until it is blocked by an event which might take hundreds of processor cycles to be resolved. While waiting for that event to be serviced, another thread can be initiated within a few processor cycles to take up the slack. This makes better use of computing resources but it introduces two undesirable side-effects – mutual exclusion of shared resources and an element of unpredictability because of the loss of thread synchronization. The management of shared resources is usually tackled by locking the resource so that only the active thread has exclusive access to it. Thread synchronization may be handled by manual coding to define points where execution may be safely passed on to another thread, and by ensuring that a thread cannot be interrupted between these points.

It should be noted that switching execution from one thread to another incurs processing overheads. Excessive switching is therefore to be avoided as it consumes computing time for an unproductive end. In an ABM containing a large number of agents, running each agent as a separate thread is likely to be impractical because of the switching overheads involved. It is not unusual to execute the model as a single thread in order to minimise this overhead. However, this runs against the agent paradigm (Wooldridge and Jennings, 1999) and begs the question – "Is ABM the most appropriate modelling approach for such a problem?"

It may be mentioned here that DEM does not suffer from either of these drawbacks since it is executed essentially as a single sequential process. This is the result of using a single event list (see Section 3.2) to control the execution of all events in a DEM. Its outputs are predictable and repeatable but, as shown in Figure 6-5, a DEM is not as scalable as an ABM.

7.2.4 Partitioning into sub-models

In the context of the case study, agent-based modelling tends strongly towards a top-down design approach by constraining a model implementer to consider high-level roles within a system at the outset. The resulting high-level modules, or sub-models, closely reflect the processes where high-level human decision making are dominant. In this part of the problem domain, the role-based agent approach (Kendall, 2000; Kendall, 2001; Parunak *et al.*, 1998) has the benefit of reducing the conceptual gap between the model and the problem being modelled.

Errors can inadvertently be introduced during system requirements capture as the transfer of an idea between two people is itself a complex process involving different personal assumptions, abilities to articulate abstractions, and levels of experience and knowledge. Therefore, the capability to map easily from the real-world system to the model is valuable, especially during the requirements analysis and design phases of the software lifecycle.

Although this attribute of agent technology is desirable in general, it may not continue to be so when carried out to its logical end, i.e. to apply the agent paradigm to all levels of detail in a model. This is indicated by the Delphi results (Table 6-9) in which the expert view was that the agent concept was likely to be understood more readily by business people (those whose work activities are centred on organizational relationships and roles) than by design engineers (those whose work activities are centred on processes). An inference to be drawn from this result is that, in order to minimise the conceptual gap between problem and model, it may be necessary to combine the agent-based approach with the traditional discrete-event approach in the same model. Where processes predominate, discrete-event modelling should be employed, and where there is high-level decision-making, possibly where information to define the problem is incomplete, the agent-based approach should be utilised. Another inference is that the agent-based approach better enables non-technical people to participate directly in the building of an ABM and so potentially shorten the implementation time. This applies similarly to the discrete-event modelling approach which can enable engineering designers who are not modelling specialists to implement a DEM using a commercially available visual programming tool.

A principle which may be drawn out from this discussion is that, in the context of this case study scenario, it is better to segregate parts of a problem in such a way as to minimise the mismatch between problem and model. This division of a problem occurs at the concept level of abstraction which is a higher level than the functional level of abstraction in existing design methods. It has the potential to speed up initial model development as well as to lower subsequent maintenance effort.

A good conceptual match is good not only for a model executing on a single processor but when applied together with a modular design offers other benefits, more pertinently, distributed modelling should that be required when a model becomes too large to be handled by a single processor.

7.2.5 Flexibility in operation

In theory, an agent in a multi-agent system can discover any other agent in its community and communicate with each other by passing messages through a common agent environment like JACK (JACK, 2007), JADE (JADE, 2006) and Lost Wax (LostWax, 2005). However, if this is allowed to take place without restraint, there is a high probability that chaos will result.

In practical systems, inter-agent communication can be quite restrictive and not all possible links are permissible. In a typical logistics system like the one used in the case study, the topology of permissible links in the agent network needs to be defined explicitly before the start of a simulation run. Depending on the environment and the internal state of an agent, the strength of its links can be adjusted during runtime. A simulation may use only a subset of the defined set of links but the rules embedded in the agents can decide which links to activate and which links are preferred over the others. This attribute of an ABM gives it an additional degree of flexibility over a traditional DEM.

Despite the restriction imposed by a limited topology, an ABM remains more flexible than a DEM because the configurable links between the agents are established at run time. There is still the possibility of loosening the restriction by allowing the ABM to implement its own links at runtime. In a traditional DEM all possible valid links have to be present at compile time and this characteristic makes it less flexible than an ABM. The structure of an ABM can change in the course of a simulation run while the structure of a DEM is fixed at the time it is constructed.

7.2.6 A natural design metaphor

In the context of the supply chain/logistics case study, there is distinct correspondence between an agent and an identifiable human role at a high level of decision making. In a model which spans a wide range of detail levels, it is more natural to employ this paradigm than process sequences to emulate human decision making. The large overlap between agent and human behaviour makes agents suitable for modelling problems which are centred on the human role (Kendall, 2000; Kendall, 2001; Parunak *et al.*, 1998; Wooldridge *et al.*, 1999). This is supported by the results of the Delphi sessions (see Table 6-9) where the experts were of the view that an ABM is easily understood by a person whose normal daily work is in business and commerce. Inter-personal relationships are dominant in these areas of employment. Further, Newell (1982) described conceptual agent interactions as occurring at knowledge level, i.e. above the computer program or symbol level and nearer the degree of abstraction at which humans communicate. Knowledge level consists of data structures and the processes for extracting knowledge from them while the symbol level contains data which may be outputs from sensors.

The DE paradigm, which is founded on processes and tends to work at data level, provides a better match to the parts of a model where processes predominate. This view is supported by the Delphi result (see Table 6-9) which is emphatic that the DEM, rather than the ABM, is more readily understood by a traditionally trained design engineer and also that the DE paradigm is better suited to the modelling of very fine details.

The relationship between processes and roles is illustrated in Figure 7-1. In the DEM, the unit of abstraction is the process which consists of a logical sequence of activities. In the ABM, the unit of abstraction is the role which contains a segment of a process. To emulate the real world, the roles form a network of relationships and are executed in parallel as concurrent threads. A conclusion which may be drawn from the evidence is that no single approach is suitable under all conditions. Therefore, based on the observations of earlier work and on the results of the case study described in Chapter 6, it can be said that the agent is a natural metaphor for role-dominant problems while traditional discrete-event is better for process-dominant ones.

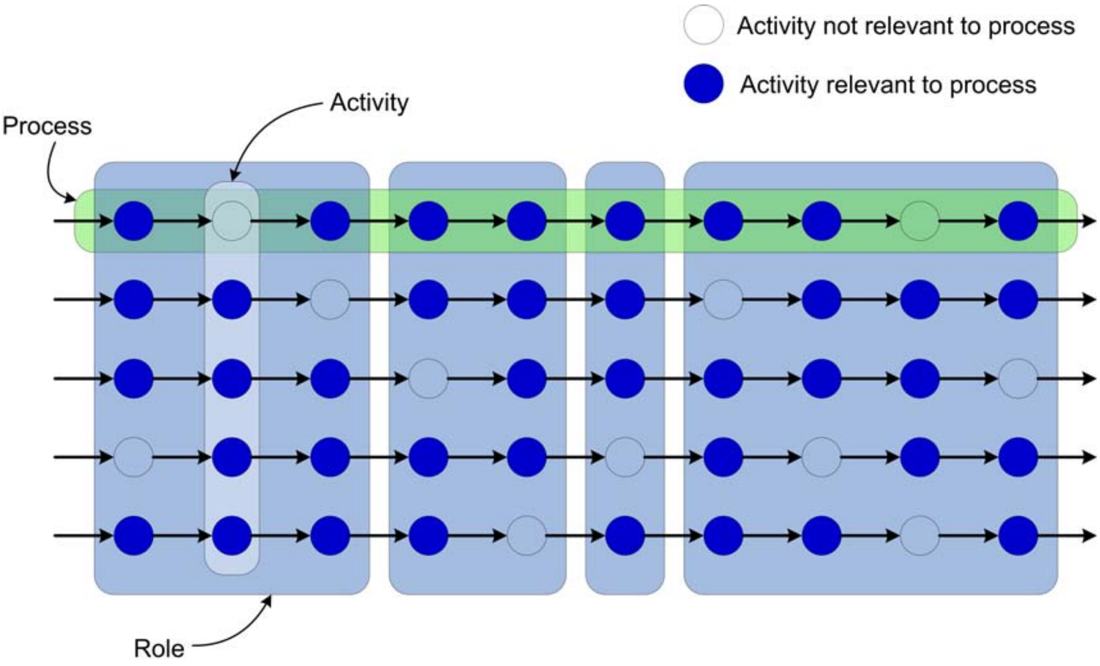

○ Activity not relevant to process
● Activity relevant to process
Activity
Process
Role

Figure 7-1: The relationship between processes, activities, and roles

7.3 Management of complexity

In the continuing enhancement and normal maintenance of a large model, a process which rightly belongs to one sub-model can sometimes be implemented elsewhere. This is sometimes done for expediency, or it may be because a modeller is not quite able to maintain a clear mental picture of the finer details of a large part of the model. Recently, it has been demonstrated clearly that the short-term, episodic or working memory is limited to maintaining a few high resolution, rather than many fuzzier, impressions for a few seconds (Zhang and Luck, 2008). Earlier, in drawing from cognitive science results, Henderson-Sellers (1996) noted that the short-term memory of an average programmer has the capacity to read and analyze between five and nine chunks (or logical groupings) of code for 20 to 30 seconds. In order to lessen the impact of this innate human limitation, there is therefore a need to manage the complexity inherent in a large model.

Booch (1994) advocates three techniques – decomposition, abstraction, and hierarchy or organization – for managing complexity in analysis and design. These are standard practice in object-oriented (OO) software development and as agent-based models are almost invariably implemented in that programming paradigm, they traditionally follow Booch’s three principles.

A consequence is that the objects they contain possess the desirable qualities of high cohesion and thus, a low degree of coupling. In such a design the amount of intra-object communication is considerably higher than inter-object communication and a change made in one object is likely to affect only code lying within the confines of the object. Localising the code helps to contain the inherent human limitation for handling large amounts of complex details concurrently and thus makes the software easier to maintain.

In a model implemented in a non-OO programming language, i.e. a procedural language in this research context, a similar design can also be reached by applying a structured design methodology like the one propounded by DeMarco (1979) or Yourdon and Constantine (1979), or a particular implementation of such a structured methodology, for example, SADT (Structured Analysis and Design Technique) and SSADM (Structured Software Analysis and Design Method). It is an established and widely practised methodology which is well supported by mature commercial off-the-shelf packages.

7.4 Criteria for agenthood

It was seen in Section 4.3 that there is a broad spectrum of software which may be described as possessing the attributes of agents. At one end of the spectrum, purely deliberative agents can be clearly identified as software with intelligence which can be so sophisticated as to make them indistinguishable from humans. At the other end, purely reactive agents are not much different from a conventional piece of software for a control system where inputs are quickly transformed into outputs through simple, fixed rules. Nevertheless, its characteristics satisfy the definition for an agent and may be considered to have crossed the boundary into the area of agent technology.

Revisiting the definition given by Wooldridge (1997) and restated by Jennings (2000) (see Section 4.2), which has been viewed as a currently and generally accepted statement of agenthood, five essential characteristics may be identified and they are –

- It is *'an encapsulated computer system'*. In this software engineering originated definition, while it is possible to admit hardware as well as software, a 'computer system' is generally taken to mean a software system with clearly defined boundaries and interfaces. Hence, in publications about agents in the context of this research, unless directed otherwise, a wholly software agent is implied whenever the term 'agent' is used. However, in the field of robotics, '*an encapsulated computer system*' may contain software as well as the hardware like image sensors, electric motors, and specialised computer processors to enable response to occur within a useful timeframe, i.e. in real-time.
- It is *'situated in some environment'*. The encapsulated computer system actively seeks inputs from, and sends outputs to the software environment it is embedded in.
- It is *'capable of flexible action... within that environment'*. The computer system is both reactive and proactive in seeking to achieve its design goal.
- It is *'capable of autonomous action... within that environment'*. In contrast to a conventional software object, which is totally obedient to an external demand, the computer system has control over its choice of action. It may choose not to respond if it perceives that doing so may be to its own, or its system's, detriment.

- The actions it takes within that environment are performed *'in order to meet its design objectives'*. Hence, the flexible, autonomous actions are executed only within the context, and in pursuit, of the agent's goal.

These characteristics were used to gauge to what degree the proposed agent-like DEM (ADEM) design meets these criteria for agenthood.

7.5 Re-visiting the discrete-event model

Based on the reasons set out in Section 7.2, this section describes the concept of a combined layered architecture and communication environment to make a DEM agent-like. It also describes how the existing traditional DEM was modified using the attributes which endues an ABM with superior characteristics. The same set of model metrics used for the case study was used to compare the traditional DEM with the ADEM.

7.5.1 Architecture for ADEM

In order to embody the desirable characteristics of an ABM described earlier in Section 7.2, it is proposed that a traditional DEM be conceptually structured as shown in Figure 7-2.

There are four components making up this architecture model and they are described in the following sub-sections.

7.5.1.1 Layer 1

This layer enacts the rules which determine the flow of items at a high-level in the ADEM. In a typical logistics problem, these rules may be implemented as a consistent body of Boolean '*if-then*' condition-action statements, fuzzy rules, or neural networks. In such a simulation model where the emphasis is on the speed of response, it is necessary to keep this layer operating as efficiently as possible. Its inputs are the data stored in the model's communication environment, and its outputs are configuration commands to the control structure in Layer 2. This layer emulates an agent with reactive architecture (see Section 4.3.2).

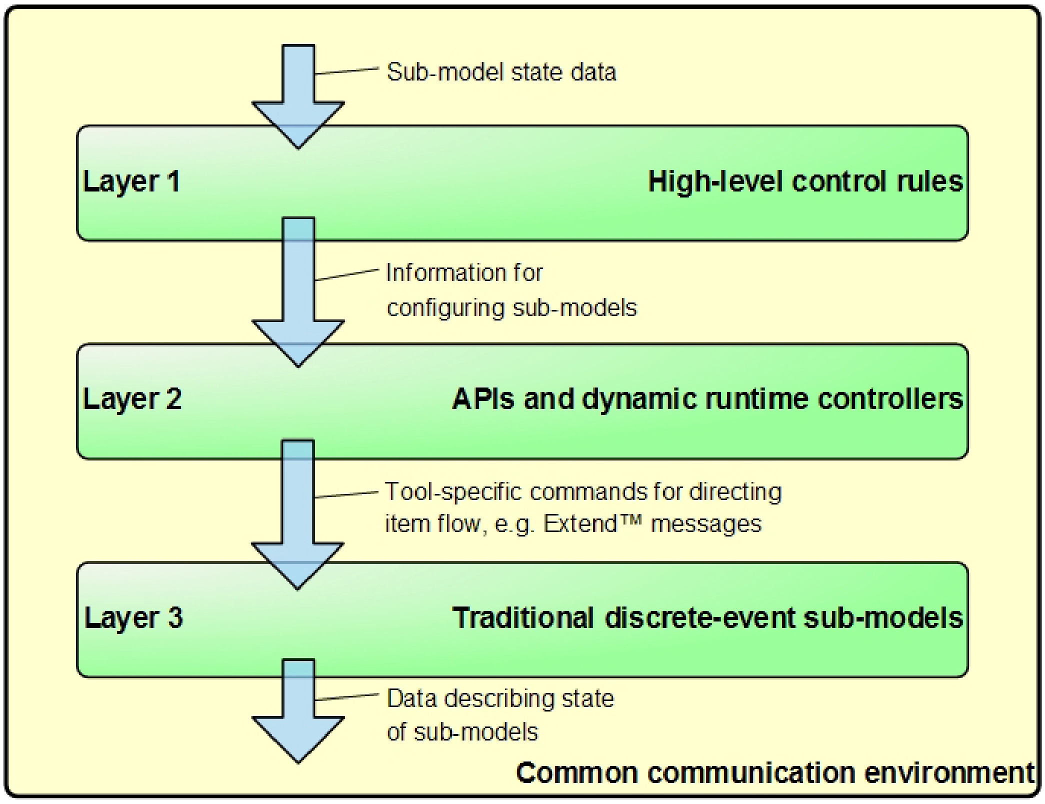

Sub-model state data
Layer 1
High-level control rules
Information for configuring sub-models
Layer 2
APIs and dynamic runtime controllers
Tool-specific commands for directing item flow, e.g. Extend™ messages
Layer 3
Traditional discrete-event sub-models
Data describing state of sub-models
Common communication environment

Figure 7-2: The three-layer architecture for an ADEM

7.5.1.2 Layer 2

This is the layer which implements the outputs of Layer 1. To enable this, it processes the information from Layer 1 in the following sequence –

- The intentions of Layer 1 are interpreted as the logical source and destination addresses for an item, for example, from OHB1 to CRV3.
- These logical addresses are translated into a form specific to the model and the modelling tool, for example, from Block 123 to Block 456.
- The information is then packaged as a message and sent to the source DEM sub-model, i.e. Block 123 in this example, using the application programming interface (API) functions provided by the modelling tool.

Hence, by modifying the source and destination address of each item, the flow of items between the traditional discrete-event sub-models in Layer 3 can be redirected. In effect, this dynamic control structure, which may be described as a configurable software switch, defines the high-level topology of the model. It enables redirection to take place at run time thus increasing model flexibility.

7.5.1.3 Layer 3

This layer represents the level at which the traditional discrete-event models operate. It can consist of several traditional DE sub-models which are linked dynamically by way of Layer 2 as the flow of items between these sub-models is controlled by Layer 2. At intervals, these sub-models update their states in the data structures allocated to them in the communication environment. The sub-model states form the inputs for the rules in Layer 1 and may contain information such as resource availability, rate of throughput, and queue length.

7.5.1.4 The common communication environment

This consists of data structures both to store information from the DE sub-models in Layer 3 as well as to act as an efficient information conduit between Layer 3 and Layer 1. Depending on the modelling tools used, the communication environment can be implemented in a variety of ways, e.g. as time-stamped message queues or global data records. The updating of information in the communication environment occurs asynchronously, which is to say that the updating software continues execution immediately after it has deposited the data. This is contrasted against synchronous operation where the updating software deposits the information and then waits until the control of execution is relinquished by the communications environment. The latter is wasteful of computing resources and can sometimes result in deadlocks thus preventing further execution of the model.

7.5.1.5 Maintainability

By separating the high-level model configuration rules as well as the control structure from the DE sub-models, maintainability is likely to be improved because this design process constrains the modeller to implement loosely coupled and highly cohesive sub-models. The nature of supply chain and logistics systems, where data is distributed and processes are local, lend themselves well to the implementation of self-contained sub-models. The rules and model control structure which, in combination, provides the agent component of this agent-like DEM, are collocated and this is likely to lead to greater understandability and modifiability.

This three-layer closed-loop architecture which is based on the discrete-event paradigm emulates an agent-based model. In the context of the case study where the Extend™6 modelling tool was used, it may be contended that by adopting the global data structure as the medium for message passing, the inherent latency due to conventional message passing and its associated processing are avoided. In this aspect of model operation, the efficiency of a traditional DEM is retained.

7.6 Implementing the architecture

The following sub-sections outline the actions taken to implement the case study scenario again but applying the architecture for an agent-like DEM described above.

It may be noted here that although the new model is intended primarily as a demonstration of implementing an ADEM, it may also be considered as a precursor to a distributed model. A natural consequence of segregating the traditional DEM into the four architectural components described in the previous section also prepares it for execution in a distributed computing environment, for instance using the HLA framework (IEEE, 2000; Kuhl *et al.*, 1999).

7.6.1 Apply a structured methodology

In Section 7.2.4, it was noted that agent-based modelling naturally constrained a model implementer to adopt a top-down design approach. There is no reason why a DEM implementer cannot take the same approach.

Taking a high level view of model design, a DEM for a supply chain or logistics problem can very frequently be partitioned naturally into sub-models according to different criteria like human role and capability, but two of the more common ones are site function and geographical location. For example, from the data description supplied for the case study, the component repair function is situated in four geographically separated locations. Hence, in sketching out an initial design of the DEM, it is reasonable to partition a component repair function into a sub-model. Similarly, the engine overhaul and the spares supply functions fall naturally into sub-models.

Working from the DEM written previously for the case study, the model was partitioned according to the well-established principles of structured design (Yourdon and Constantine, 1979) so as to minimise coupling between sub-models and at the same time maximise cohesion within them. This is achieved by strictly limiting the activities to those of the site function alone. For example, component repair was restricted to activities immediately related to repairs and was not mixed with the restocking of spare parts.

By separating the functions, the resulting main sub-models, i.e. the overhaul base, component repair centre, and parts supply centre, can be stored and reused. The Extend™6 modelling tool allows them to be stored as library hierarchical blocks and reusing them is just a matter of replicating them by dragging them onto the model design sheet. They integrate easily into the model since their interfaces are small and well-defined.

A consequential benefit of applying this software engineering methodology is the low volume of data flow between the sub-models when compared with the flow of data within each sub-model. A model structured in this manner can form the basis for a distributed model in the future.

7.6.2 Emulate an agent environment

In an ABM, information between agents is communicated asynchronously through messages passed via a software environment such as that provided by the open-source JADE (JADE, 2006) platform or the commercially available JACK (JACK, 2007) and Aerogility™ (LostWax, 2005) agent frameworks. Information may be actively sought by an agent or provided unsolicited by other agents. In a busy system with many agents, the information may sometimes not be provided within a useful timeframe. Although asynchronous message passing is versatile in that it can enable models implemented by disparate modelling tools to communicate by acting as middleware, it can nevertheless suffer from high overheads because the messages, which can sometimes be long character strings, have to be queued, detected, extracted, processed, and executed.

Using the Extend™6 modelling tool, an agent environment can be emulated in a simple and limited way. It provides the means of communication but not the agent management services (for instance, to create and terminate agents) and directory facilitator (‘*Yellow Pages*’ service for finding other agents) found in fully-fledged agent environments. A model-wide global area can be provided so that any of the sub-models can write to it and read from it. For instance, information from the component repair centres advertising their current turnaround time, availability of skill types, and the cost of services they are prepared to offer is updated at regular intervals in this global area. A sub-model wishing to know the state of a repair centre has just to read from the relevant memory location. Communicating by way of a shared area of memory is faster, simpler, and more efficient than message passing as it has few of the overheads associated with message handling. The overheads of updating the global information are small in comparison.

7.6.3 Improved flexibility

It was recognized in Sections 7.2.4 and 7.2.6 that the agent paradigm is better suited to modelling high-level decision making while the DE paradigm gives a better match to engineering processes. Agents work with information at the high level end of a model where problem details and data flow volume are low while DE modules work with data at the low level end where details and data flow volume are high. Further, an impediment to flexible modelling is that the structural links of a DEM are fixed at the time the model is assembled or compiled. To improve flexibility an innovation to the model was introduced. A higher layer of control logic was added to the model in the form of two DE blocks to coordinate the flow of components – the ‘*OHB Controller*’ for the two OHBs and the ‘*Repair Controller*’ for the four component repair centres (see Figure 7-5). These are role-based blocks and they act as surrogate human controllers, i.e. agents.

The functions of these controllers are twofold –

- To define the valid links among the sub-models and thus implement the model structure. This function is also present in the ABM where the links are defined as data during model initialisation. It may be noted that in order to move an item from one sub-model to another in the

ADEM during a simulation run, the destination of the item was determined at run time. This is different from the traditional DEM where the route of an item is hard-coded at compile time.

- To control the flow of components according to embedded rules and current sub-model states. In Figure 7-3 and Figure 7-4, the '*OHB Controller*' and the '*Repair Controller*' respectively, the addresses of the source sub-model and destination sub-model were implemented as attributes of an item. As the item passed through the '*Source*' and '*Destination*' decision blocks in these controllers, their embedded decision rules were executed to determine the item's destination. Depending on the states of the sub-models, the initial destination address may be modified by these decision blocks. For example, these sub-model states, implemented as global variables, can help in deciding how to route items for OHB or repair centre load balancing, for special skills required to inspect and repair items, for shortest turnaround time, or for cheapest repairs.

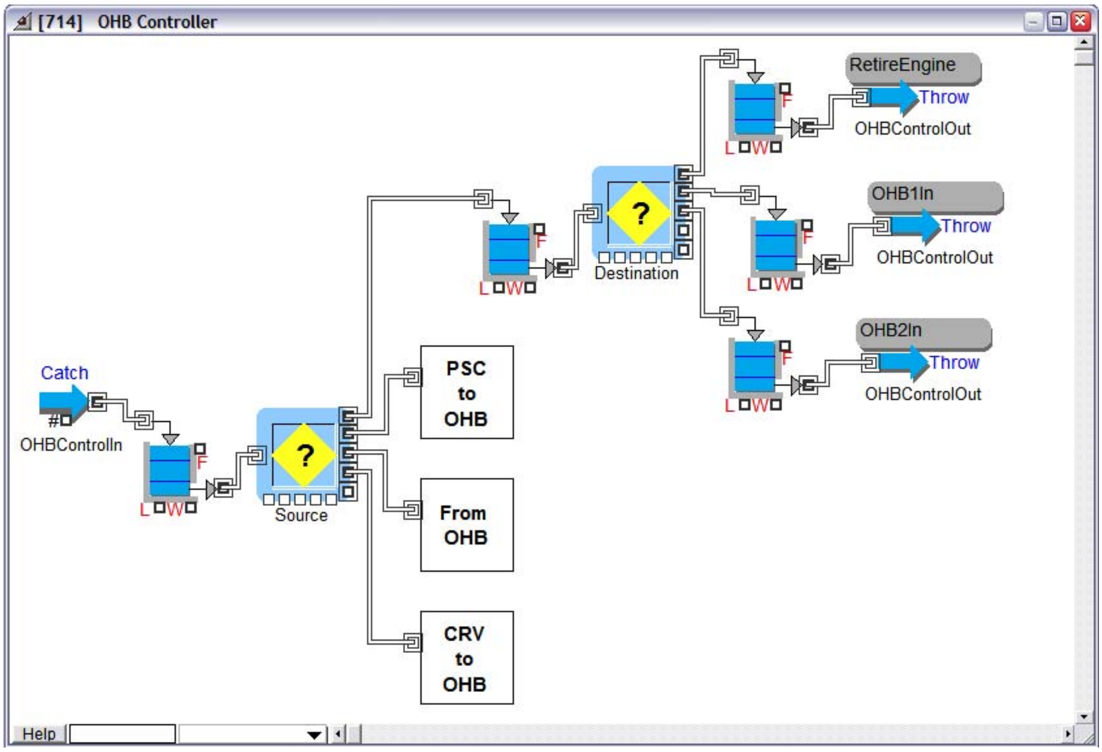


Figure 7-3: OHB Controller

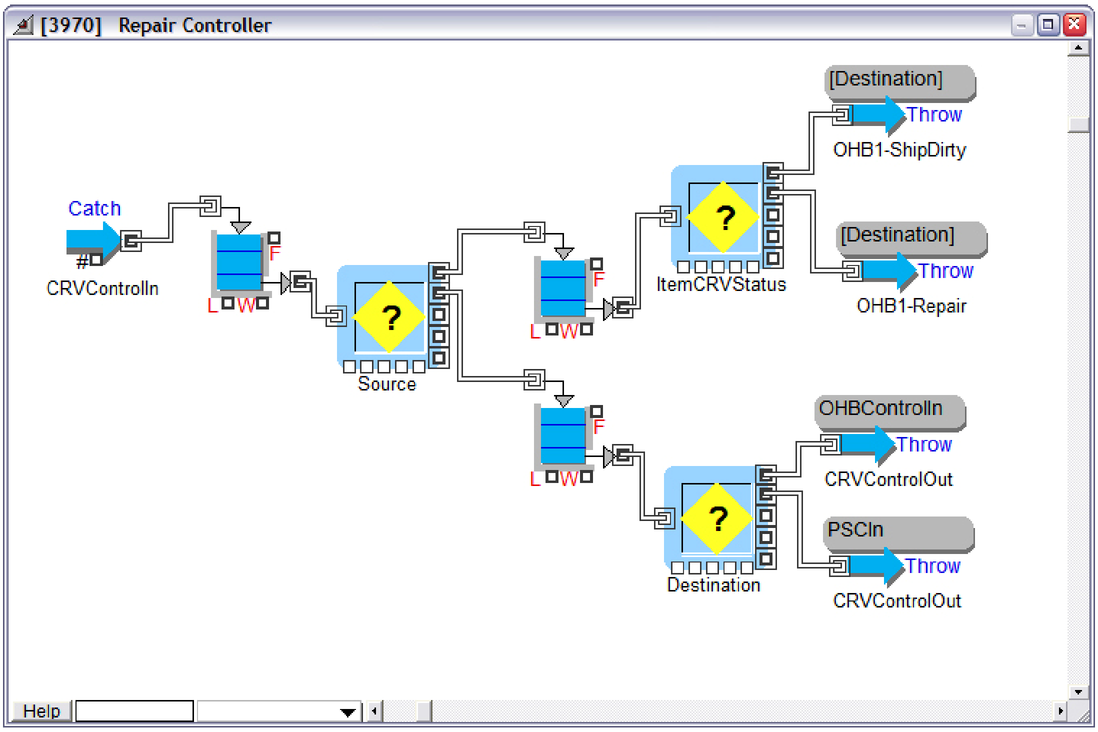


Figure 7-4: Repair Controller

7.6.4 Results

Working from an understanding of discrete-event modelling, it was anticipated that the execution times of the ADEM would be a little longer than those for the traditional DEM under identical conditions. The underpinning feature of discrete-event modelling, i.e. the event list or event calendar for the whole model, still applies, and together with the additional processing overheads needed to maintain a flexible model structure meant that the execution times would be extended.

To ascertain whether reconfiguring the workflow in the model would make any difference in the execution time, alternative routing rules were formulated at the controller blocks (the '*OHB Controller*' and the '*Repair Controller*' blocks in Figure 7-5). The rules in the controller blocks enabled the repair workload to create one of the following scenarios – recreate the workload of the traditional DEM, be evenly spread amongst the four repair centres, be evenly spread between the two OHBs, be even spread between the OHBs and repair centres, or be directed to one repair centre for the duration of a simulation run. The execution times were almost identical in each of these cases.

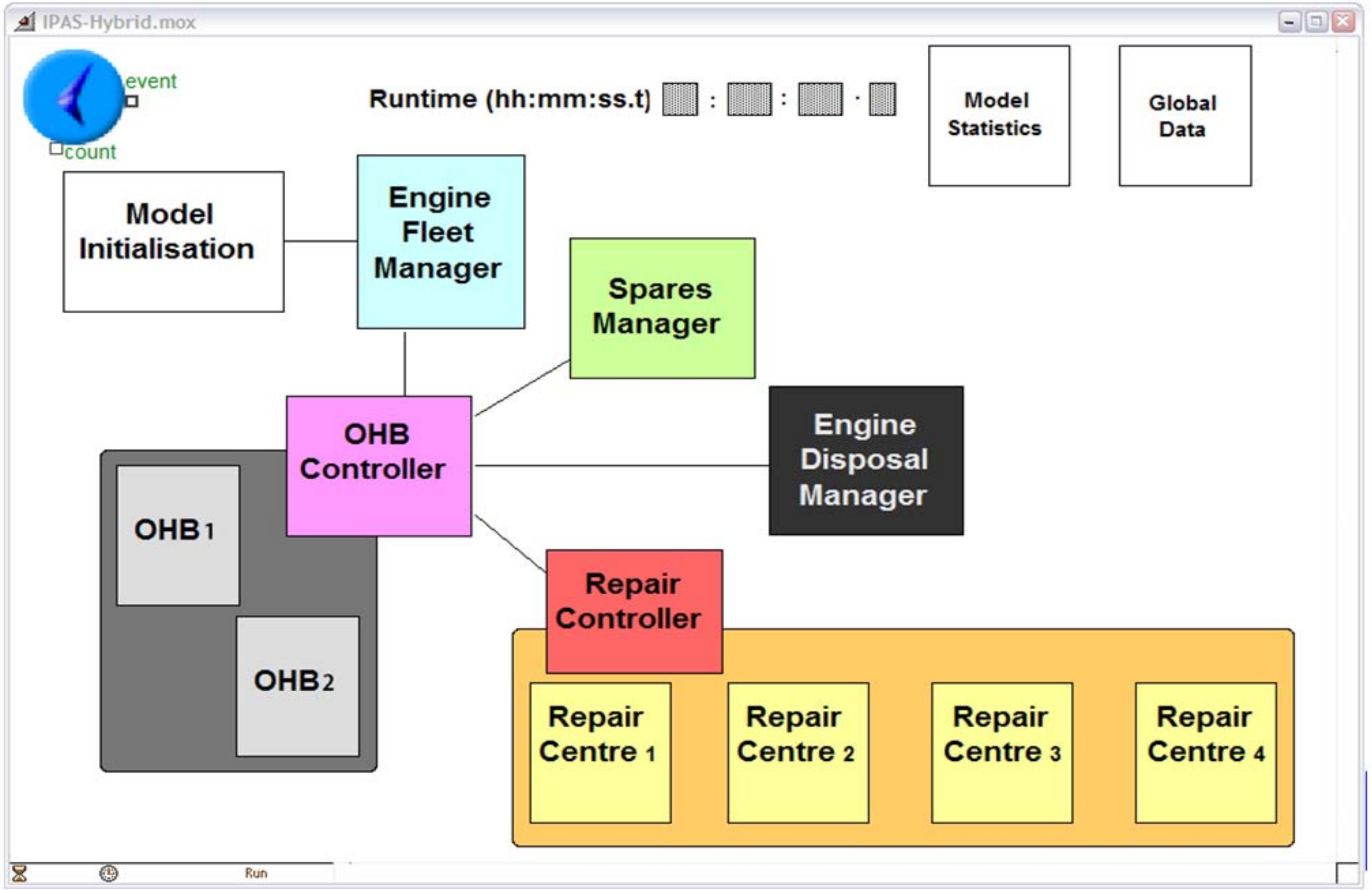


Figure 7-5: An overview of the agent-like DEM (ADEM). The '*OHB Controller*' and the '*Repair Controller*' are role-based hierarchical blocks functioning as agents to control the flow of items. All other blocks are traditional DEM sub-models.

For comparison, the run times of the ABM, ADEM, and DEM are shown in Figure 7-6. As described in Section 6.4.3 and also shown in Figure 6-17, the run times for the ADEM and DEM are shown with and without model initialisation. It is clear from the graphs that model initialisation in both these Extend™6 models occupy a significant portion of total run time. The ABM is initialised differently and the time taken for this activity is negligible when compared with total run time.

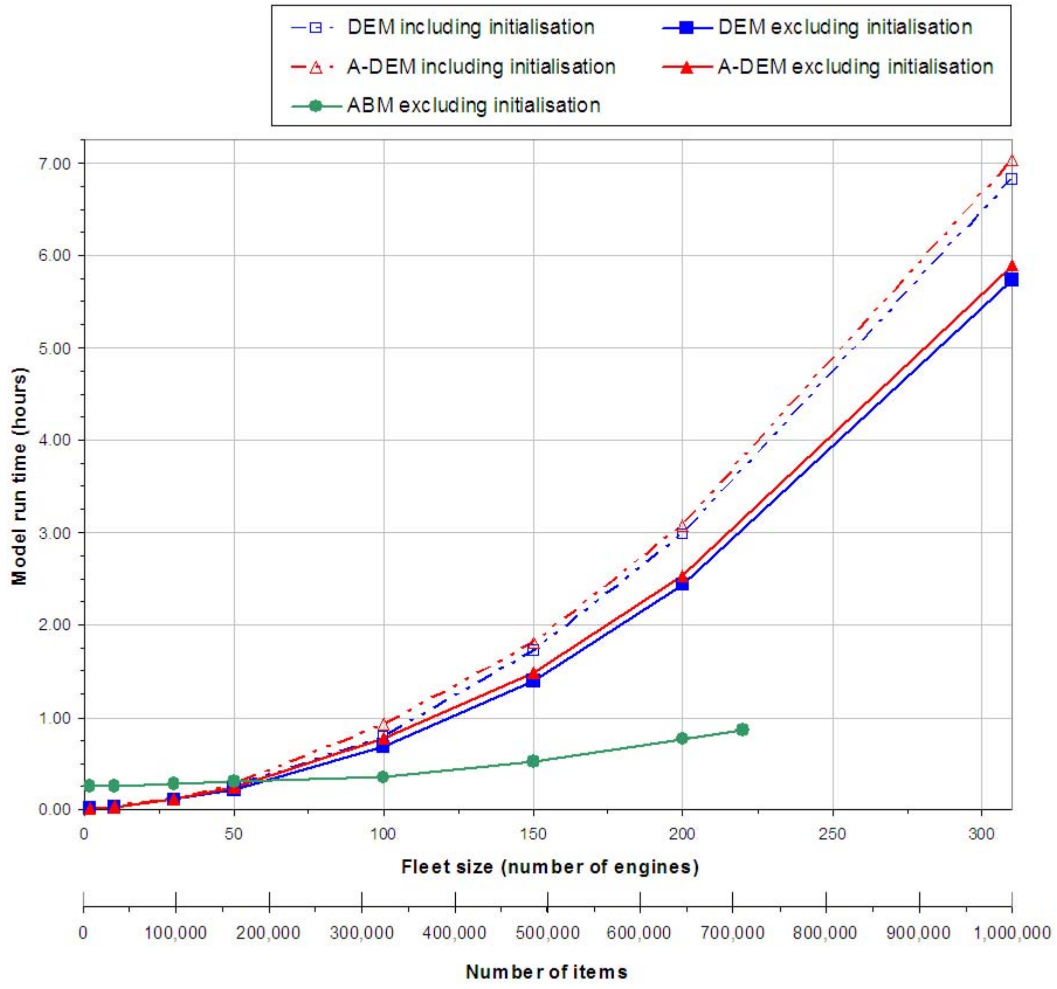


Figure 7-6: Runtimes for ADEM, DEM, and ABM for different engine fleet sizes

When compared with the performance of the earlier traditional DEM, the ADEM execution times were longer by between 0.8% and 3.4%. This is a slight degradation in performance and should be considered together with other software qualities. It may be an acceptable trade-off for making the model more flexible, understandable, and modifiable thus better satisfying the demands of a dynamic model development environment typically encountered in the lifecycle of a model. As mentioned in Section 7.2.2, cost incurred in the post-implementation or maintenance phase of a model forms a significant proportion of total software lifecycle cost.

Table 7-1 shows that when compared with the earlier traditional DEM, the size and complexity metrics for the ADEM were larger by between 1% and 28%. By making the sub-systems self-contained, more code than before was required to handle their interaction with other sub-systems thus increasing the numerical values of these metrics. Although the ADEM sub-systems were larger, they were easier to understand since the modular code resulting from applying a structured design method made the model clearer and easier to follow. Both Boehm (1981) and Welker (2001) strongly recommend the use of code reviews by expert programmers to assess program maintainability. They are of the opinion that while code metrics provide an automated, objective measure, it is but one view which however needs to be confirmed by another means. Welker (2001) provides an example which showed that a larger and more complex piece of code, as measured by LOC and McCabe cyclomatic number respectively, was judged to be more maintainable because of the helpful comments it contained. This result points to the importance of adhering strictly to best programming practice in order to ensure the production of high maintainability software.

Table 7-1: Multiples of ADEM to DEM metrics

Sub-system	Number of executable statements, *LOC*	Number of methods or procedures	Number of classes or modelling blocks	Weighted methods per class, *WMC*
A	1.3	1.3	1.3	1.1
B	1.3	1.2	1.1	1.2
C	1.0	1.0	1.0	1.2
D	1.0	1.0	1.0	1.1
E	1.3	1.3	1.2	1.1

Table 7-2 summarizes the extent to which Layers 1 and 2 meet the five-part definition of an agent as provided by Wooldridge (1997) and Jennings (2000). Considered together, Layers 1 and 2 emulate an agent since all five parts of the definition are satisfied. However, it should be stressed that this consideration is based only a single agent in isolation. It is more pertinent to compare models and it is very likely that an ABM will be implemented as a multi-agent system.

According to Wooldridge and Ciancarini (2001) such a multi-agent system is inherently a concurrent multi-threaded process. Each agent has its own thread of execution and is continually actively engaged in an infinite loop sensing its environment, updating its internal state, and performing the appropriate action. As the ADEM is based on the discrete-event paradigm, all actions in the model are managed by the event list and can only be carried out sequentially. It has been discussed in Section 7.2.3 that for practical reasons, a very large multi-agent system is usually executed as a single-threaded process, i.e. the agents are executed sequentially. Therefore, the ADEM does not comply with the definition of an ideal multi-agent system but compares well with some practical, large multi-agent systems. Thus, it is justified in describing the ADEM as an agent-like discrete-event model rather than a hybrid of an agent-based and a discrete-event model.

Table 7-2: Summary showing how ADEM satisfies the definition of an agent

	Constituent elements of an agent (Jennings, 2000; Wooldridge, 1997)	**ADEM architecture**
1	'*An encapsulated computer system*'	The modular design of the model, enforced by a structured design methodology, ensures that the model configuration rules and code in Layers 1 and 2 have clearly defined boundaries and interfaces.
2	'*Situated in some environment*'	The rules in Layer 1 seek input from the common communication environment. The outputs from Layer 1 act on the Layer 3 DEM sub-models which outputs information to the common communication environment. This completes the agent input/output loop.
3	'*Capable of flexible action… within that environment*'	In reaction to changes in its environment, code in Layer 1 work in concert with the dynamically configurable switch in Layer 2 to modify the model structure and to redirect the flow of

	Constituent elements of an agent (Jennings, 2000; Wooldridge, 1997)	ADEM architecture
		model items. The case study logistics scenario did not give the occasion to implement a deliberative agent but the condition-action rules present in the '*OHB Controller*' and '*Repair Controller*' blocks satisfy the description of a reactive agent.
4	'*Capable of autonomous action… within that environment*'	The architecture is not based on message passing but on Layer 3 sub-models publishing information in the communication environment. The body of code and rules in Layer 1 decides on what, if any, action needs to be taken.
5	'*In order to meet its design objectives*'	The rules and code in Layer 1 act on information in the communication environment and respond in such a way as to meet its design objectives which may be, for example, achieving lowest cost or shortest overhaul time.

7.7 Summary

The benefits of an ABM over a traditional DEM may be summed up by its smaller size and lower complexity; its higher scalability and ability to exploit parallel execution; its natural tendency to segregate into loosely coupled sub-models; its better conceptual match at a high level of decision-making; and its flexibility in operation due to its ability to reconfigure structurally at runtime.

Agent technology enables the management of problem complexity to be made easier because it is object-oriented and the established techniques of decomposition, abstraction, and organization can be applied to it. However, the same benefit can be obtained for non-OO languages by the disciplined use of a structured design methodology.

The five-part definition for an agent was considered in greater detail so that an informed judgement could be made later on how well the agent-like DEM (ADEM) fitted the criteria.

A layered architecture comprising four components was proposed for the ADEM to enable a model with both role-dominant and process-dominant parts to work together.

The traditional DEM used in the case study was converted into an ADEM and code metrics as well as model run times were obtained for comparison. The ADEM was slightly larger and executed slightly slower than the traditional DEM. However, it was considered that these minor drawbacks were acceptable tradeoffs against the benefits of greater understandability of the code and increased flexibility in operation.

Lastly, the controller blocks in the ADEM were compared against the agent definition and it may be concluded that they satisfy the five parts of the definition. Despite that, the ADEM is unlike an ideal multi-agent system since it does not execute as a concurrent multi-threaded process.

Chapter 8
Conclusions and Future Work

In conclusion, this chapter presents a précis of the main conclusions reached through the research which has been carried out. It also outlines the particular contributions towards furthering the understanding of the agent-based and the traditional discrete-event modelling paradigms. Finally, as a result of the work completed, other areas of research are suggested where further investigation may be worthwhile.

8.1 Conclusions

The case study, which involved the modelling of the Rolls-Royce Trent 800 fleet repair operation using an ABM and a functionally identical traditional DEM, demonstrated the differences between the two modelling approaches. The H_1 hypothesis stated in Section 1.4, that '*agent-based modelling is better than traditional discrete-event modelling*', has to be qualified since the benefits of the agent-based modelling are not universal but are evident only under certain conditions. These conditions will be made clear in the conclusions contained in the following sub-sections where applicable.

8.1.1 Code metrics for models

The ABM was shown to be smaller, less complex, and consequently more maintainable than the traditional DEM. This result applies specifically within the context of the case study scenario, i.e. for a logistics problem containing level of details ranging from flexible, human decision-making to detailed, fixed process sequences, and after the '*system implementation*' activity in the development lifecycle (see Figure 6-6 for Systems Development Lifecycle). Code metrics which are normally compiled during the implementation of a software project – LOC, number of methods, classes, and procedures, as well as MCN and WMC – all emphasise the superiority of ABM in terms of program size and complexity.

8.1.2 Loose coupling and high cohesion

Agents in the ABM and DEM blocks are loosely coupled, and since communication amongst the agents was achieved through asynchronous message passing the ABM holds a slight advantage. In a well-designed system, loose external coupling implies high internal cohesion, thus resulting in related code which is likely to be located within the same module. As such, the effect of a change in one module will not ripple far out to other parts of the system thus amplifying its effect. Based on this software quality of maintainability alone, the mental effort required to understand, modify, and test a change to the ABM will be less than that required for the DEM, thus lowering cost.

8.1.3 Wider aspects of modelling

The superiority of the ABM is less obvious when the '*requirements analysis*' and '*design*' activities are taken into consideration (see Figure 6-6 for Systems Development Lifecycle). The considered views of software and modelling experts from the Delphi sessions were that the ABM was more easily understood by people whose normal work activities centred on organizational relationships while the DEM was more easily understood by people whose activities centred on processes. Information gathering for requirements analysis will be optimized if the agent and the traditional discrete-event paradigms are used appropriately when interviewing people who are involved in different levels of the problem domain. While the conclusions in Section 8.1.1 are objective, they form but an element of the comparison between the two modelling approaches. The enlarged scope of the systems development lifecycle considered here is closer to reality and is more useful in practice.

8.1.4 Scope of an agent

The scope of an agent is not confined just to the activities of code implementation and subsequent maintenance. It encompasses the system development lifecycle activities of requirements analysis and design as well. As a body of programme code, an agent is not greatly different from conventional programme code. Its benefits are realised when the agent concept is applied in a disciplined manner early in the model lifecycle, i.e. from the requirements analysis stage onwards, where high level agent roles are initially identified. The early application of effort to reduce the conceptual gap between the real world and the written specification provides a good foundation for all subsequent cost reduction since less mental effort will be needed to understand, modify, and test the model software.

8.1.5 Modelling best practice

Both ABM and DEM benefit from the disciplined application of modelling best practice. A top-down structured design methodology helps to make the complexity of a system manageable to a model implementer through the design techniques of hierarchy, decomposition, and abstraction. A consequence applying such a methodology is the natural segregation of a model into appropriate sub-systems or agents. It was recognised during the implementation of the DEM that designing it using one of the commercially available, mature, and proven structured design methodologies can help to reduce the initially perceived gap between the two modelling approaches significantly.

8.1.6 Matching problem with model

A multi-agent system used as a model is better suited to role-dominant problems as the AI aspect of agent technology is a better match than discrete-event model for situations where information may not be complete. Also, in contrast to the strict sequential activities of a DEM, a multi-threaded multi-agent process simulates parallel activities and is a closer approximation to human communication in the real world.

8.1.7 Flexibility in operation

A traditional DEM requires all information and organisational relationships to be present at the time the model is compiled for execution thus fixing the model structure for the duration of a simulation run. In comparison, a multi-agent model has the ability to create and terminate links in the course of a simulation run so endowing the ABM with greater flexibility. However, the freedom to restructure a model in such a manner has to be controlled by rules to ensure that only valid links can be made.

8.1.8 Model performance

The elapsed, or '*wall-clock*', time of an ABM simulation run is not as sensitive to changes in model workload as that of a DEM. The flatter ABM response time graph is due largely to it being a time-driven model with a fixed simulated time step while the DEM is event-driven.

In the ABM, the passage of time is modelled and therefore the elapsed time of a simulation run does not vary by much even when there are no items to process and the CPU is idle most of the time. As the number of items for processing continues to increase, CPU idle time will continue to decrease. Up to the point where there is still some idle time during a simulation run (a fleet size of about 100 engines in Figure 7-6), thus indicating the presence of spare capacity in the CPU, the simulation run time remains about the same. As the number of items increases beyond this, the simulation run time will increase because the ability of the CPU to respond to the demands of the model continues to diminish.

In the event-driven DEM, where the notion of a fixed time step is absent, the model is always executed as quickly as possible progressing from one event to the next. Hence, the CPU is always fully loaded during a DEM simulation run. This is in contrast to the ABM which runs with CPU idle time up to the point mentioned in the previous paragraph. It is clear from Figure 7-6 that, because of the difference in CPU utilisation between the two modelling paradigms, the DEM has a shorter simulation run time when the fleet has fewer than 60 engines while the ABM is quicker for engine fleets larger than that.

However, the ABM is a heavy user of computing resources and it failed catastrophically when it ran out of memory. In contrast, the DEM was light on computing resources and successfully completed simulation runs with engine fleet sizes which exceeded the largest managed by the ABM (see Figure 7-6).

8.1.9 Repeatability and predictability

Simulation runs with the DEM can be guaranteed to be repeatable and predictable. Although stochastic processes are present, the DEM may nevertheless be considered to be deterministic for the reason that the seeds for the random inputs (they are pseudo-random in reality) can be left unchanged between model executions. In addition, there is only one event list in a DEM and since this data structure determines the event sequence, the results of different runs with the same input data will be the same. However, repeatability and predictability cannot be guaranteed with the ABM because other unrelated but essential computer processes, e.g. scheduling and network processes, can introduce undesirable perturbations into the execution of the concurrent threads making up the process. In an ABM, where it is usual for each agent to be run in its own separate thread, the delayed output in one thread caused by the execution of other processes can periodically result in unpredictable and undesirable consequences. This is an inherent problem of multithreaded execution the solution to which sometimes is to avoid multithreaded operation altogether.

8.1.10 Multithreaded execution

Although multi-threaded execution is an inherent component of the agent paradigm and is beneficial to it, it is nevertheless a significant drawback of the ABM. To ensure repeatable simulation results, the threads have to be synchronized by manual coding. Similarly, to prevent deadlocks, shared resources have to be locked for exclusive access by each thread by manual coding. Complete test coverage can be difficult, or impossible, to achieve in multi-agent systems with a large number of threads of execution.

8.2 Contributions of research

Although various modelling approaches have been compared before (see Section 2.5), the comparison of an ABM with a traditional DEM has not been attempted to date. Where the other published comparisons have been wholly qualitative, a large part of this research involved the quantitative and objective measurement of model characteristics.

To minimise the conceptual gap between problem and model, it is better to partition a model according to the relative dominance of roles or processes in the problem domain. Roles may be implemented using the agent paradigm while processes may be implemented using the discrete-event paradigm. The logistics problem in the case study scenario has processes fully occupying the lower, detailed level while roles predominate at the higher, strategic levels. A problem such as this is more suitably addressed as a combination of agents and discrete-event processes where each modelling paradigm can be applied where appropriate. This combined approach is proposed in the thesis.

Building on the results of the case study, a layered architecture is presented as the framework for implementing such a combined model in the discrete-event paradigm. The three conceptual layers represent the sub-systems of traditional DEMs, intelligent switches which enable the model to be restructured dynamically, and agents which enable goals of the model to be achieved. The resulting model exploits the strengths of both modelling paradigms and in doing so, extends the range of practical usefulness of traditional DEMs. This combined-paradigm model is described as an agent-like DEM (ADEM). The reason it is not described as an agent/discrete-event hybrid in the full sense of that term is because the agents present in the model cannot be implemented as traditional multi-threaded agent processes but have to be executed as a part of a single-threaded sequential DEM.

An important outcome of the research is the observation that most of the published benefits of an ABM can be achieved by a DEM through the disciplined adherence to a well-established structured design methodology. By reinforcing the desirable qualities of high modularity, loose coupling, and high cohesion in the design of a DEM, the gap between the two modelling paradigms is reduced.

8.3 Future work

8.3.1 Extension of model comparison

Although software maintainability from the perspectives of the purchaser and the model implementer has been a significant component of this research, other software qualities such as reliability, efficiency, correctness, and usability are equally important to the model user. As agent technology is increasingly being adopted by model implementers and becomes more prominent in the mainstream of modelling, these other software qualities will assume greater importance. Maintainability addresses the primary issues of software cost and effort, both of which are pre-eminent questions when seeking justification to migrate to a new technology, the other software qualities address matters of user confidence in a new technology which will be expected to be better than the technology it will replace and be in constant operation. In the present paucity of quantitative information about agent technology, a more holistic view of the technology will be obtained by widening the scope of such a comparison.

8.3.2 Distributed modelling

Another natural extension of the modelling paradigm comparison carried in this thesis is to consider the important problems related to distributed modelling. As the demand for larger and more realistic models continues to grow, the computing capacity offered by a single processor will be unable to satisfy that. This is evident from the results of the performance measurements carried out in this research (see Figure 7-6) where both models were hindered, for different reasons, from successfully simulating the operation of a realistically large engine fleet. The ABM experienced a hard failure because it ran out of memory while the time required by the DEM to complete a run may be considered too long in a commercial environment.

It is likely that the various existing civil and military engine fleets RR has to maintain will continue to expand. Although the existing memory constraint imposed by a 32-bit hardware and software system can be addressed directly by migrating to a 64-bit system, it is unlikely that a simulation run will complete in an acceptable time. A promising way to model a large engine fleet and yet give a reasonably short response time is to exploit parallelism in modelling. There are techniques for the synchronization of model time (Section 3.3.2) and a standardised framework (see Section 3.3.2.3 High-Level Architecture) to enable a model to be distributed to geographically dispersed processors for parallel execution. They can be used for the seamless integration of models implemented by collaborating enterprises.

8.3.3 Optimization

It may be argued that the primary motive for building a simulation model is to be able to arrive as quickly as possible at the optimal solution for a problem. At present, a common method of exploring the solution space is to conduct numerous sensitivity studies by manually varying one or two input variables at a time. This is unlikely to be adequate for large and complex models such as those typically required by RR to simulate the logistics for engine fleet maintenance. Specialized external optimizers will be needed to operate in conjunction with these models and thus enable multivariate optimization to be carried out. The clamour to improve on an existing solution will be driven largely by the very large financial returns which can be accumulated over the lifetime of an engine fleet.

8.4 Concluding remarks

The work carried out in this research has identified some of the strengths and weaknesses of agent-based modelling when compared with discrete-event modelling. The outcomes of the comparison have been deployed in a novel model architecture which combines the strengths of both modelling paradigms. Although discrete-event modelling has been widely used in industry for many years, the new agent-like discrete-event model may be considered to have contributed to its continuing development, specifically in the class of problems exemplified by the logistics model used in this research.

As modelling tools become more powerful, sophisticated, and user-friendly, the production of models or model parts becomes more likely to be devolved from specialist modellers to end-users like engine designers and cost engineers. It is the hope that this research has begun to help this migration on its way.

Appendix A

A.1 Technology Readiness Levels

The definitions of the levels of technology readiness recommended by the Carnegie Mellon Software Engineering Institute (Graettinger *et al.*, 2002) are presented in Table A-1. This nine-point scale is as a metric for assessing the maturity of new technologies. Although applicable to all technologies, this version of the Technology Readiness scale has been adapted for assessing new software technologies.

Table A-1: The SEI Technology Readiness Levels

TECHNOLOGY READINESS LEVEL	DESCRIPTION
1. Basic principles observed and reported	**Hardware/Subsystem:** Lowest level of technology readiness. Scientific research begins to be translated into applied research and development. Examples might include paper studies of a technology's basic properties. **Software:** Lowest level of software readiness. Basic research begins to be translated into applied research and development. Examples might include a concept that can be implemented in software or analytic studies of an algorithm's basic properties.
2. Technology concept and/or application formulated	**HW/S**: Invention begins. Once basic principles are observed, practical applications can be invented. Applications are speculative and there may be no proof or detailed analysis to support the assumptions. Examples are limited to analytic studies. **SW**: Same as **HW/S**

TECHNOLOGY READINESS LEVEL	**DESCRIPTION**
3. Analytical and experimental critical function and/or characteristic proof of concept	**HW/S**: Active research and development is initiated. This includes analytical studies and laboratory studies to physically validate analytical predictions of separate elements of the technology. Examples include components that are not yet integrated or representative. **SW:** Active research and development is initiated. This includes analytical studies to produce code that validates analytical predictions of separate software elements of the technology. Examples include software components that are not yet integrated or representative but satisfy an operational need. Algorithms run on a surrogate processor in a laboratory environment.
4. Component and/or breadboard validation in laboratory environment	**HW/S**: Basic technological components are integrated to establish that they will work together. This is relatively "low fidelity" compared to the eventual system. Examples include integration of ad hoc hardware in the laboratory. **SW**: Basic software components are integrated to establish that they will work together. They are relatively primitive with regard to efficiency and reliability compared to the eventual system. System software architecture development initiated to include interoperability, reliability, maintainability, extensibility, scalability, and security issues. Software integrated with simulated current/legacy elements as appropriate.

TECHNOLOGY READINESS LEVEL	DESCRIPTION
5. Component and/or breadboard validation in relevant environment	**HW/S**: Fidelity of breadboard technology increases significantly. The basic technological components are integrated with reasonably realistic supporting elements so it can be tested in a simulated environment. Examples include "high fidelity" laboratory integration of components. **SW**: Reliability of software ensemble increases significantly. The basic software components are integrated with reasonably realistic supporting elements so that it can be tested in a simulated environment. Examples include "high fidelity" laboratory integration of software components. System software architecture established. Algorithms run on a processor(s) with characteristics expected in the operational environment. Software releases are "Alpha" versions and configuration control is initiated. Verification, Validation, and Accreditation (VV&A) initiated.
6. System/subsystem model or prototype demonstration in a relevant environment	**HW/S**: Representative model or prototype system, which is well beyond that of TRL 5, is tested in a relevant environment. Represents a major step up in a technology's demonstrated readiness. Examples include testing a prototype in a high-fidelity laboratory environment or in a simulated operational environment. **SW**: Representative model or prototype system, which is well beyond that of TRL 5, is tested in a relevant environment. Represents a major step up in software demonstrated readiness. Examples include testing a prototype in a live/virtual experiment or in a simulated operational environment. Algorithms run on processor of the operational environment are integrated with actual external entities. Software releases are "Beta" versions and configuration controlled. Software support structure is in development. VV&A is in process.

TECHNOLOGY READINESS LEVEL	**DESCRIPTION**
7. System prototype demonstration in an operational environment	**HW/S**: Prototype near, or at, planned operational system. Represents a major step up from TRL 6, requiring demonstration of an actual system prototype in an operational environment such as an aircraft, vehicle, or space. Examples include testing the prototype in a test bed aircraft. **SW**: Represents a major step up from TRL 6, requiring the demonstration of an actual system prototype in an operational environment, such as in a command post or air/ground vehicle. Algorithms run on processor of the operational environment are integrated with actual external entities. Software support structure is in place. Software releases are in distinct versions. Frequency and severity of software deficiency reports do not significantly degrade functionality or performance. VV&A completed.
8. Actual system completed and qualified through test and demonstration	**HW/S**: Technology has been proven to work in its final form and under expected conditions. In almost all cases, this TRL represents the end of true system development. Examples include developmental test and evaluation of the system in its intended weapon system to determine if it meets design specifications. **SW**: Software has been demonstrated to work in its final form and under expected conditions. In most cases, this TRL represents the end of system development. Examples include test and evaluation of the software in its intended system to determine if it meets design specifications. Software releases are production versions and configuration controlled, in a secure environment. Software deficiencies are rapidly resolved through support infrastructure.

TECHNOLOGY READINESS LEVEL	**DESCRIPTION**
9. Actual system proven through successful mission operations	**HW/S**: Actual application of the technology in its final form and under mission conditions, such as those encountered in operational test and evaluation. Examples include using the system under operational mission conditions. **SW**: Actual application of the software in its final form and under mission conditions, such as those encountered in operational test and evaluation. In almost all cases, this is the end of the last "bug fixing" aspects of the system development. Examples include using the system under operational mission conditions. Software releases are production versions and configuration controlled. Frequency and severity of software deficiencies are at a minimum.

Appendix B

B.1 A nominal definition of Complex Adaptive Systems

The notion of Complex Adaptive Systems has originated from a number of contributors. In the absence of a concise definition, Dooley (1996) has proposed a description of a complex adaptive system summarised from works which are considered as seminal. First, his method is presented and that is followed by the definition and the literature sources.

B.2 The method

The nominal definition to be put forth is forged from the works of Gell-Mann (1994), Holland (1995), Jantsch (1980), Maturana and Varela (1992), and Prigogine and Stengers (1984). The essential principles of a CAS, as defined in each work, were carefully noted. These conceptual lists were then merged into one master list of concepts. Common themes were noted and an abbreviated list was developed. This aggregate list of concepts was then put into a structural model that synthesized the concepts into a single description.

B.3 The definition

The basic elements of a CAS are agents. Agents are semi-autonomous units that seek to maximize their fitness by evolving over time. Agents scan their environment and develop schema. Schema are mental templates that define how reality is interpreted and what are appropriate response for a given stimuli. These schema are often evolved from smaller, more basic schema. These schema are rational bounded: they are potentially indeterminate because of incomplete and/or biased information; and they differ across agents. Within an agent, schema exist in multitudes and compete for survival via a selection-enactment-retention process.

When an observation does not match what is expected, an agents can take action in order to adapt the observation to fit an existing schema. An agent can also purposefully alter schema in order to better fit the observation. Schema can change through random or purposeful mutation, and/or combination with other schema.

When schema change it generally has the effect of making the agent more robust (it can perform in light of increasing variation or variety), more reliable (it can perform more predictably), or more capable in terms of its requisite variety (in can adapt to a wider range of conditions).

The fitness of the agent is a complex aggregate of many factors, both local and global. Unfit agents are more likely to instigate schema change. Optimization of local fitness allows differentiation and novelty/diversity; global optimization of fitness enhances the CAS coherence as a system and induces long term memory.

Schema define how a given agent interacts with other agents surrounding it. Actions between agents involve the exchange of information and/or resources. These flows may be nonlinear. Information and resources can undergo multiplier effects based on the nature of interconnectedness in the system. Agent tags help identify what other agents are capable of transaction with a given agent; tags also facilitate the formation of aggregates, or meta-agents. Meta-agents help distribute and decentralize functionality, allowing diversity to thrive and specialization to occur. Agents or meta-agents also exist outside the boundaries of the CAS, and schema also determine the rules of interaction concerning how information and resources flow externally.

B.4 The sources

Gell-Mann, M. (1994). *The Quark and the Jaguar*. New York: Freeman & Co.

Holland, J.H. (1995). *Hidden Order, Reading*, MA: Addison-Wesley.

Jantsch, E. (1980). *The Self-Organizing Universe*. Oxford: Pergaman Press.

Maturana, H. and F. Varela (1992). *The Tree of Knowledge*. Boston: Shambhala.

Prigogine, I., & I. Stengers (1984). *Order Out of Chaos*. New York: Bantam Books.

Appendix C

C.1 Process maps for engine maintenance

The following six process maps form part of the ABM's functional specification from which the traditional DEM was implemented.

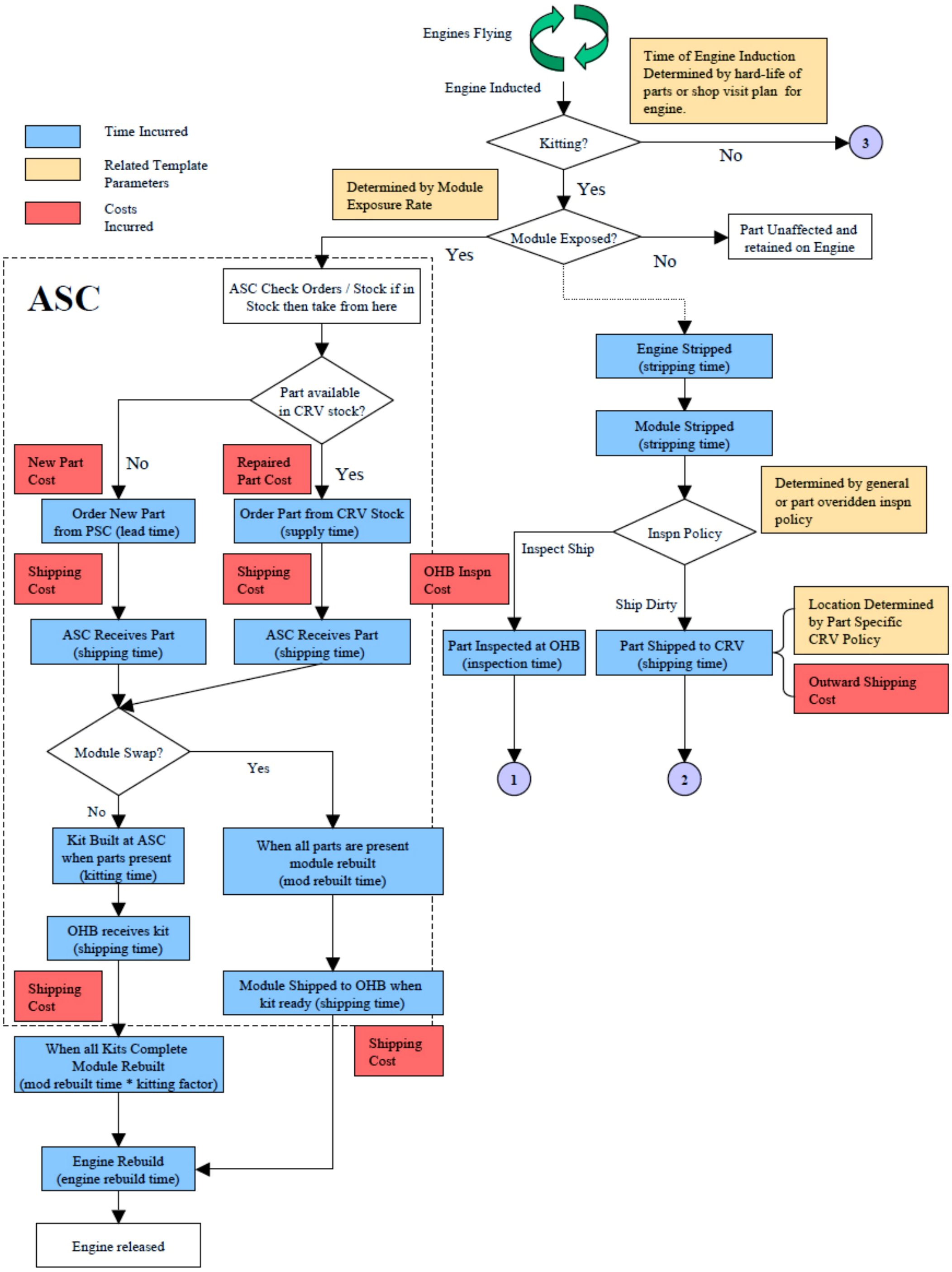

Engines Flying
Engine Inducted
Time of Engine Induction Determined by hard-life of parts or shop visit plan for engine.
Time Incurred
Related Template Parameters
Costs Incurred
Kitting?
No
3
Yes
Determined by Module Exposure Rate
Module Exposed?
Part Unaffected and retained on Engine
Yes
No
ASC
ASC Check Orders / Stock if in Stock then take from here
Part available in CRV stock?
New Part Cost
No
Repaired Part Cost
Yes
Order New Part from PSC (lead time)
Order Part from CRV Stock (supply time)
Shipping Cost
Shipping Cost
ASC Receives Part (shipping time)
ASC Receives Part (shipping time)
Engine Stripped (stripping time)
Module Stripped (stripping time)
Determined by general or part overidden inspn policy
Inspn Policy
Inspect Ship
OHB Inspn Cost
Ship Dirty
Location Determined by Part Specific CRV Policy
Part Inspected at OHB (inspection time)
Part Shipped to CRV (shipping time)
Outward Shipping Cost
1
2
Module Swap?
Yes
No
Kit Built at ASC when parts present (kitting time)
When all parts are present module rebuilt (mod rebuilt time)
OHB receives kit (shipping time)
Shipping Cost
Module Shipped to OHB when kit ready (shipping time)
Shipping Cost
When all Kits Complete Module Rebuilt (mod rebuilt time * kitting factor)
Engine Rebuild (engine rebuild time)
Engine released

Figure C-1: Process map 1 – Initial OHB processes and the ASC (Rolls-Royce plc)

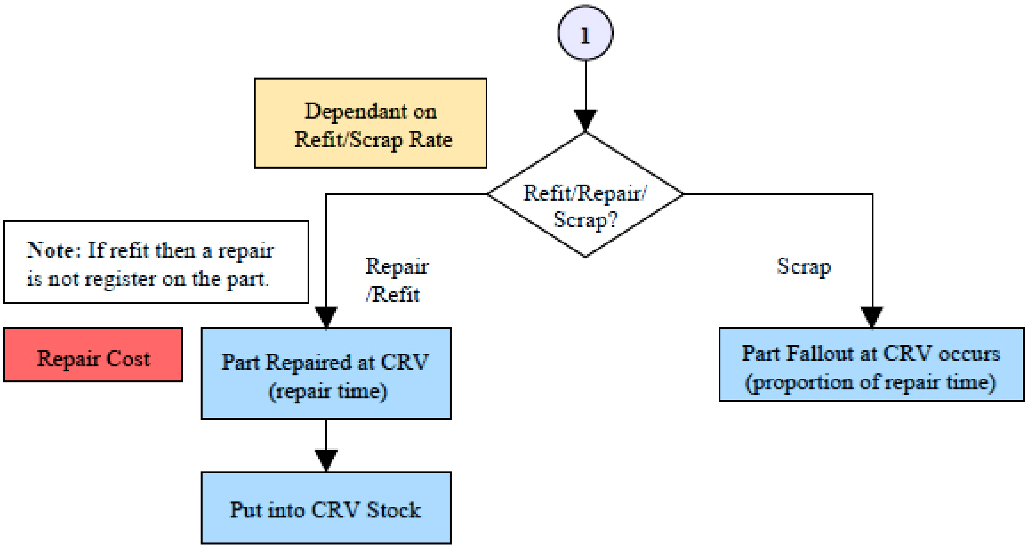


Figure C-2: Process map 2 – OHB processes (Rolls-Royce plc)

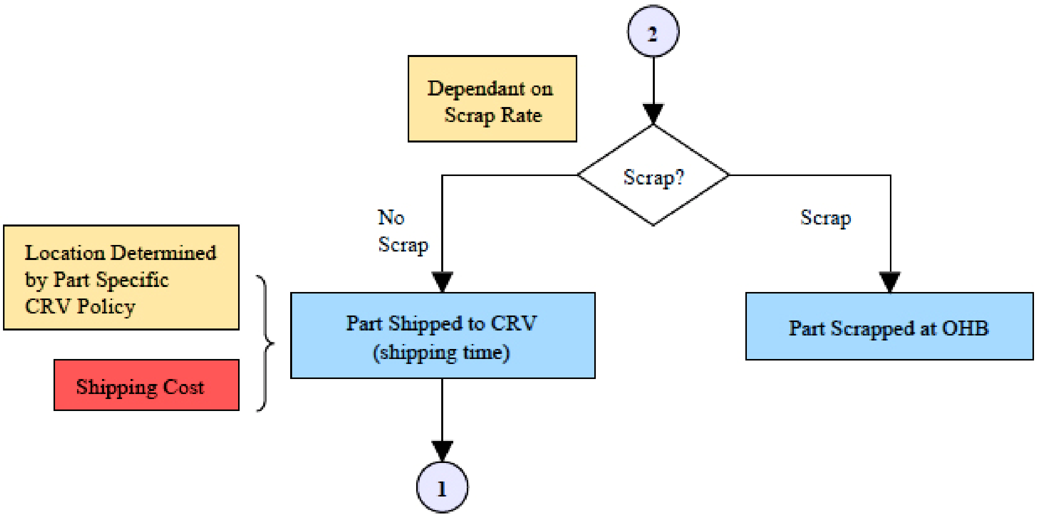


Figure C-3: Process map 3 – CRV processes (Rolls-Royce plc)

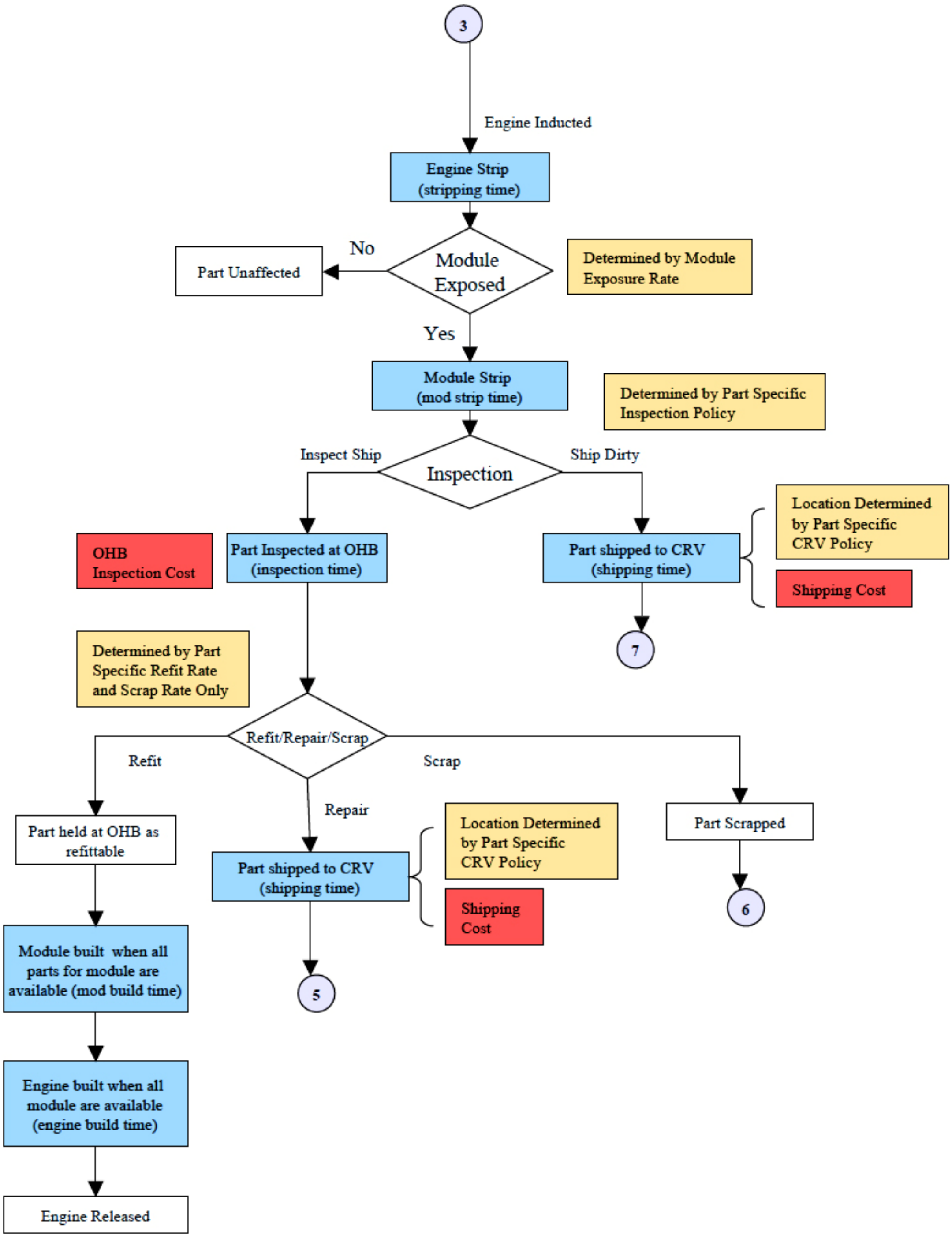

3
Engine Inducted
Engine Strip (stripping time)
Module Exposed
No
Part Unaffected
Determined by Module Exposure Rate
Yes
Module Strip (mod strip time)
Determined by Part Specific Inspection Policy
Inspection
Inspect Ship
Ship Dirty
OHB Inspection Cost
Part Inspected at OHB (inspection time)
Part shipped to CRV (shipping time)
Location Determined by Part Specific CRV Policy
Shipping Cost
7
Determined by Part Specific Refit Rate and Scrap Rate Only
Refit/Repair/Scrap
Refit
Scrap
Repair
Part held at OHB as refittable
Part shipped to CRV (shipping time)
Location Determined by Part Specific CRV Policy
Shipping Cost
Part Scrapped
6
5
Module built when all parts for module are available (mod build time)
Engine built when all module are available (engine build time)
Engine Released

Figure C-4: Process map 4 – 'No kitting' OHB processes (Rolls-Royce plc)

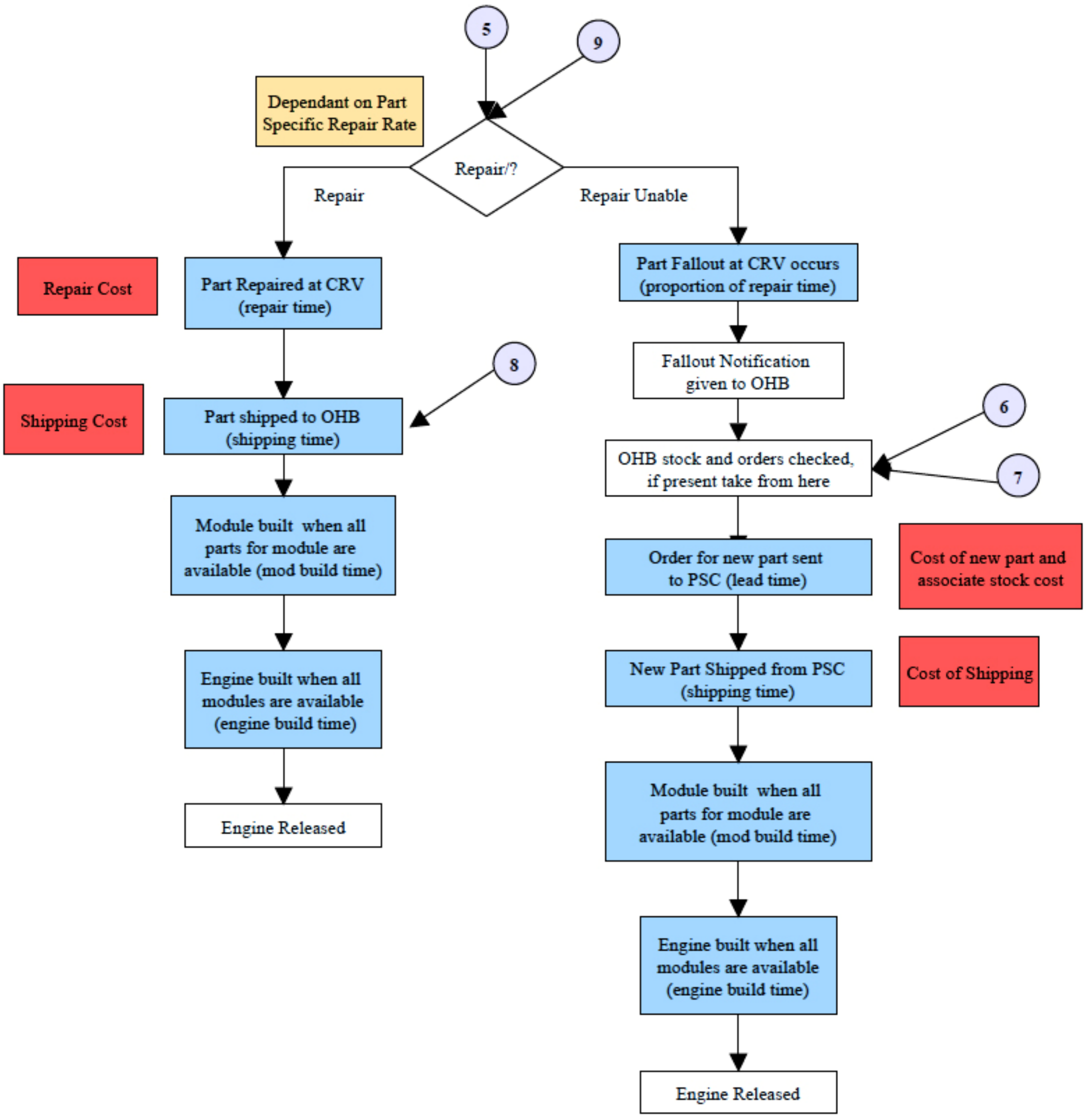

5
9
Dependant on Part Specific Repair Rate
Repair/?
Repair
Repair Unable
Repair Cost
Part Repaired at CRV (repair time)
Part Fallout at CRV occurs (proportion of repair time)
8
Shipping Cost
Part shipped to OHB (shipping time)
Fallout Notification given to OHB
6
OHB stock and orders checked, if present take from here
7
Module built when all parts for module are available (mod build time)
Order for new part sent to PSC (lead time)
Cost of new part and associate stock cost
Engine built when all modules are available (engine build time)
New Part Shipped from PSC (shipping time)
Cost of Shipping
Engine Released
Module built when all parts for module are available (mod build time)
Engine built when all modules are available (engine build time)
Engine Released

Figure C-5: Process map 5 – CRV processes (Rolls-Royce plc)

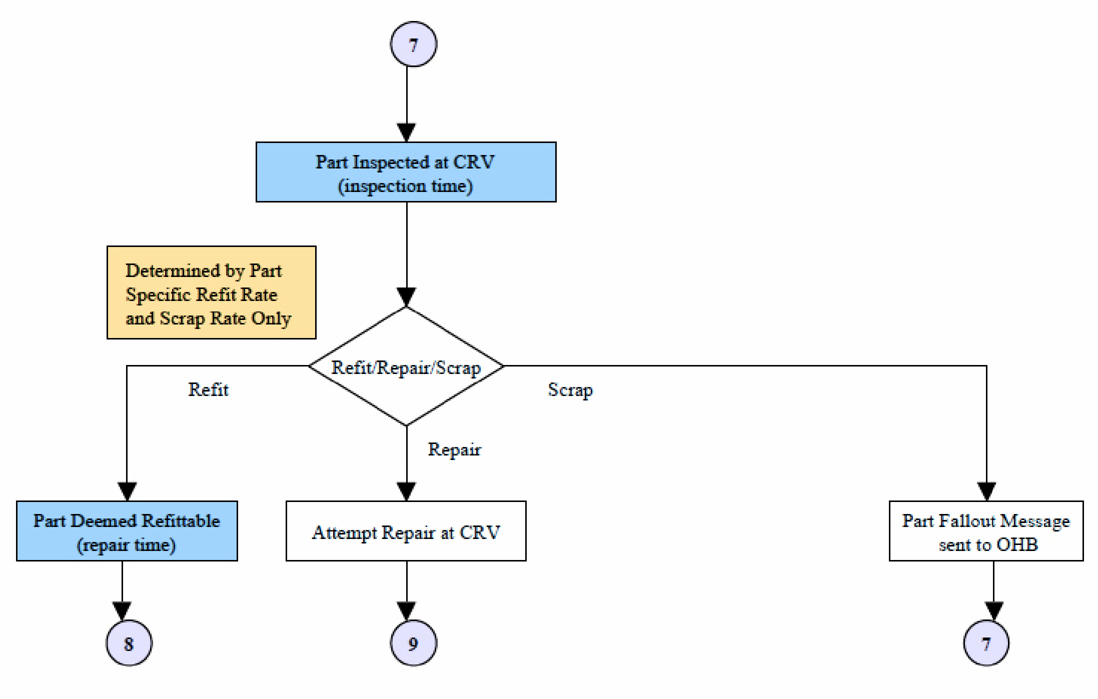


Figure C-6: Process map 6 – CRV processes (Rolls-Royce plc)

Appendix D

D.1 Data for the agent-based and traditional discrete-event models

The data for the agent-based model included a table of numerical values for the attributes of a 67-component engine model. Table D-1 lists the engine components and the modules they belong to while Table D-2 lists the attributes for each component.

Table D-1: Components and modules for the Trent 800 engine (Rolls-Royce plc)

Number	Component name	Module number
1	Annulus-Fillers	01
2	Fan-Blades	01
3	Fan-Disc	01
4	Fan-Shaft	02
5	Front-Combustion-Liner	04
6	HPC-Blades-St1	04
7	HPC-Blades-St2	04
8	HPC-Blades-St3	04
9	HPC-Blades-St4	04
10	HPC-Blades-St5	04
11	HPC-Blades-St6	04
12	HPC-Drum-St1-4	04
13	HPC-Drum-St5-6	04
14	HPC-Vanes-St1	04
15	HPC-Vanes-St2	04
16	HPC-Vanes-St3	04
17	HPC-Vanes-St4	04
18	HPC-Vanes-St5	04
19	HPT-Blade	04
20	HPT-Disc	04

Number	Component name	Module number
21	HPT-NGV	04
22	HPT-Seal-Segment	04
23	IPC-Blades-St1	02
24	IPC-Blades-St2	02
25	IPC-Blades-St3	02
26	IPC-Blades-St4	02
27	IPC-Blades-St5	02
28	IPC-Blades-St6	02
29	IPC-Blades-St7	02
30	IPC-Blades-St8	02
31	IPC-Drum	02
32	IPC-OGV	02
33	IPC-Shaft	02
34	IPC-Vanes-St3	02
35	IPC-Vanes-St4	02
36	IPC-Vanes-St5	02
37	IPC-Vanes-St6	02
38	IPC-Vanes-St7	02
39	IPT-Blade	05
40	IPT-Disc	05
41	IPT-NGV	05
42	IPT-Seal-Segment	05
43	IPT-Shaft	05
44	LPT-Shaft	08
45	LPT-Stg-1-Blade	08
46	LPT-Stg-1-Disc	08
47	LPT-Stg-1-NGV	08
48	LPT-Stg-1-Seal-Segment	08
49	LPT-Stg-2-Blade	08
50	LPT-Stg-2-Disc	08
51	LPT-Stg-2-NGV	08
52	LPT-Stg-2-Seal-Segment	08
53	LPT-Stg-3-Blade	08

Number	Component name	Module number
54	LPT-Stg-3-Disc	08
55	LPT-Stg-3-NGV	08
56	LPT-Stg-3-Seal-Segment	08
57	LPT-Stg-4-Blade	08
58	LPT-Stg-4-Disc	08
59	LPT-Stg-4-NGV	08
60	LPT-Stg-4-Seal-Segment	08
61	LPT-Stg-5-Blade	08
62	LPT-Stg-5-Disc	08
63	LPT-Stg-5-NGV	08
64	LPT-Stg-5-Seal-Segment	08
65	VIGVs	02
66	VSV-St1	02
67	VSV-St2	02

Table D-2: Component attributes (Rolls-Royce plc)

Number	Attribute
1	Component name
2	Module number
3	Kit name
4	Quantity per engine
5	Maximum number of repairs
6	Weight in pounds
7	Inspection policy
8	CRV
9	PSC capacity per day (new parts)
10	CRV capacity per day (repaired parts)
11	PSC price (US$) for a new part
12	CRV price (US$) for a repaired part
13	PSC turnaround in days (new parts)
14	CRV turnaround in days (repaired parts)

Number	Attribute
15	OHB inspection price (US$)
16	OHB inspection time (days)
17	CRV inspection time (days)
18	CRV total repair time (days)
19	Airline A (max used %)
20	Airline B (max used %)
21	Refit rate (shop visit number 1)
22	Refit rate (shop visit number 2)
23	Refit rate (shop visit number 3)
24	Refit rate (shop visit number 4)
25	Refit rate (shop visit number 5)
26	Refit rate (shop visit number 6)
27	Scrap rate (shop visit number 1)
28	Scrap rate (shop visit number 2)
29	Scrap rate (shop visit number 3)
30	Scrap rate (shop visit number 4)
31	Scrap rate (shop visit number 5)
32	Scrap rate (shop visit number 6)
33	Repair rate (shop visit number 1)
34	Repair rate (shop visit number 2)
35	Repair rate (shop visit number 3)
36	Repair rate (shop visit number 4)
37	Repair rate (shop visit number 5)
38	Repair rate (shop visit number 6)

Appendix E

E.1 Applying the Delphi Method

In the case study carried out for this research, the Delphi process outlined earlier by Brown (1968) and Gordon (1994), and reiterated more recently by Stellman and Greene (2005), was implemented with the minor adaptation that all communication was carried out remotely by email. This was necessary because the heavy work load of the three participating experts meant that it was almost impossible to hold meetings in a common location for the Delphi sessions within a reasonable timeframe. As it transpired, asynchronous communication by email not only enabled the time taken by the whole process to be shortened because participation could take place at any time, it also ensured that the questionnaire responses remained anonymous since there was no direct communication.

The three experts who were invited to be participants were all highly experienced software engineers who had coded programmes and managed software projects of various sizes and complexities. However, each of them possessed different levels of knowledge of ABM and DEM. In addition to the three experts, a fourth person acted as the facilitator to initiate the Delphi process, to collect and analyse the responses, and to keep the process in motion until consensus was achieved.

The Delphi '*kick-off*' session was in the form of an email with two attachments – a questionnaire (the details are presented in Section E.2), and a document which contained a statement of the purpose of the session as well as descriptions of the Delphi process to be followed, the goal of the session, the two modelling paradigms, and the problem to be considered. This email was sent to to the three participants at the same time and it also contained the instruction that responses to the questionnaire were expected within one week. Further, the functional specification, the flowcharts, and packages of code for the agent-based model and the discrete-event model were not distributed but were made available for easy access by the participants for reference if the need arose.

Using the interquartile measure as the criterion, it was judged that some responses to the initial questionnaire ranged too widely for them to be described as consensual opinion. With the exception of one question to which all participants gave the same score, responses to the other questions were separated by up to two clear points on the six-point ordinal response scale. Those which differed by two clear points were therefore wider than the interquartile range and so did not represent consensus.

The results for all 14 questions, i.e. both the scores as well as the reasons given to justify the scores, were collated and sent out again. Those questions which did not achieve consensus were highlighted so that the participants could consider them further. All responses to the second iteration of the questionnaire were within the interquartile range and therefore did not require further consideration by the participants. The quantitative results for the questions presented in Section E.2 show the number of participants who estimated that a particular point in the ordinal scale corresponded most closely to his or her opinion.

To classify the responses as '*nominal*', '*moderate*', or '*very*', the results were processed as follows –

- Assign a score of 6 to the leftmost point on the response scale of each question. Decrease the score by 1 as the scale is traversed from left to right. The rightmost point should have a score of 1.
- The weighted mean for the response to a question is calculated and a class is assigned. For example, in Section E.2.2, the two weighted means for Question 1 are evaluated as (((5*1)+(4*2))/3)-3.50 = 0.83 and (((6*1)+(5*2))/3)-3.50 = 1.83.
- The weighted means for a response is classified using the scheme shown in Table E-1.

Table E-1: Result classification

Value	Class	Symbol
Between 2.99 and 2.00	Very	+++
Between 1.99 and 1.00	Moderate	++
Between 0.99 and 0.00	Nominal	+
Between -0.01 and -1.00	Nominal	-
Between -1.01 and -2.00	Moderate	--
Between -2.01 and -3.00	Very	---

E.2 The questionnaire

The following sub-sections present the questionnaire which was distributed to the participants. The questions did not require alteration as they appeared to be well understood and consensus was reached quickly after two iterations.

E.2.1 Introduction to the questionnaire

The goal of this questionnaire is to obtain some measure of the maintainability of the two models. In order to perform this part of the model lifecycle, i.e. to correct or enhance an existing model, a person coming fresh to it, must understand it first before modifying the code and subsequently testing the model to ensure new outputs are valid and the existing ones continue to be so. The questionnaire is divided into four parts, i.e. understandability, modifiability, testability, and problem/paradigm matching.

It will be useful to bear in mind the context of this case study and to remember the descriptions of the two modelling paradigms given earlier.

It is important that in the first 2 or 3 of iterations of this questionnaire, supplying the reasons for the responses will help greatly towards ensuring a rapid convergence of view.

E.2.2 Understandability

A single process may be made up of a sequence of several activities involving a number of people in different roles. Processes are usually represented as flowcharts. While traditional DEM is centred on processes, ABM tends to be centred on people and their activities. The role of an agent may include one or more activities. The functional specification, flowcharts, and code are available for reference.

1. How do you rate this statement? – *As* a model implementer, *it is easier to understand how a model works if it is set out as processes rather than activities*.

	Strongly agree	Moderately agree	Agree	Disagree	Moderately disagree	Strongly disagree
Activities		1	2			
Your reason						
Processes	1	2				
Your reason						

2. How do you rate this statement? – *As* a professional engineer (e.g. engine designer)*, it is easier to relate to an engineering problem if it was described as processes rather than activities*.

	Strongly agree	Moderately agree	Agree	Disagree	Moderately disagree	Strongly disagree
Activities				3		
Your reason						
Processes	2	1				
Your reason						

3. How do you rate this statement? – *As* a non-engineer (e.g. an employee in a commercial department) *it is easier to understand how a model works if it is set out as processes rather than activities*.

	Strongly agree	Moderately agree	Agree	Disagree	Moderately disagree	Strongly disagree
Activities		2	1			
Your reason						
Processes		1		2		
Your reason						

4. Based on the UK education model, what *level of formal education* do you think is required to understand a DEM and an ABM?

	GCSE	A-Levels	Batchelor	Master	Doctorate	Post-doctorate
ABM				2	1	
Your reason						
DEM		1	2			
Your reason						

E.2.3 Modifiability

When software is in production use within an established quality assurance framework, it is the normal practice for all modifications to be formally specified initially as textual descriptions. The software implementer then transforms the specification into program code. The functional specification, flowcharts, and code are available for reference.

5. If *additional activities* were to be made to one or more subsystems of the model, e.g. a Component Repair Vendor (CRV), how easy or difficult do you think it would be to code the required changes working from the textual specification?

	Very easy	Moderately easy	Easy	Difficult	Moderately difficult	Very difficult
ABM			1	2		
Your reason						
DEM			3			
Your reason						

6. If *two or more* CRVs were to be added to the model, how easy or difficult do you think it would be to modify the existing models?

	Very easy	Moderately easy	Easy	Difficult	Moderately difficult	Very difficult
ABM		1	2			
Your reason						
DEM	1		2			
Your reason						

E.2.4 Testability

Once the changes have been coded, they have to be tested to ensure that the model outputs are valid. The question here relate to the question in the previous section and it will be useful also to remember that the DEM is normally implemented as a single CPU process while the ABM is normally implemented as concurrent threads of execution. The functional specification, flowcharts, and code are available for reference.

7. If *additional activities* were to be made to one or more subsystems of the model, e.g. a Component Repair Vendor (CRV), how easy or difficult do you think it would be to test the changes?

	Very easy	Moderately easy	Easy	Difficult	Moderately difficult	Very difficult
ABM					3	
Your reason						
DEM					3	
Your reason						

E.2.5 Matching of problem to modelling paradigm

The case study scenario (Rolls-Royce Trent 800 engine maintenance program) involves logistics when estimating lifecycle costs. The models are at a fairly high level, e.g. the aggregated time for stripping an engine module for repair is specified or modelled but not the detailed activities.

8. How do you rate the suitability of ABM and DEM to modelling at a high level of abstraction?

	Very unsuitable	Moderately unsuitable	Unsuitable	Suitable	Moderately suitable	Very suitable
ABM				1	2	
Your reason						
DEM				3		
Your reason						

9. How do you rate the suitability of ABM and DEM when modelling to very fine details is required?

	Very unsuitable	Moderately unsuitable	Unsuitable	Suitable	Moderately suitable	Very suitable
ABM		1	2			
Your reason						
DEM					2	1
Your reason						

References

AL-KILIDAR, H., COX, K. & KITCHENHAM, B. (2005) The Use and Usefulness of the ISO/IEC 9126 Quality Standard. IN *2005 International Symposium on Empirical Software Engineering*, IEEE, 126-132.

AMDAHL, G. (1967) Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. IN *AFIPS Conference Proceedings,* Atlantic City, N.J., AFIPS Press, Reston, Va., 483-485.

ARENA (2007) *Arena Product Overview* [Online][Accessed on 01 September 2007] Available from: <http://www.arenasimulation.com/products/default.asp>

BALCI, O. (1998) Verification, validation, and testing. IN Banks, J. (Ed.) *The Handbook of Simulation*, New York, NY, John Wiley & Sons.

BANKER, R.D., DATAR, S.M., KEMERER, C.F. & ZWEIG, D. (1993) Software Complexity and Maintenance Costs. *Communications of the ACM,* 36, **11,** 81-94.

BANKS, J., CARSON, J.S. & NELSON, B.L. (1999) *Discrete-event System Simulation*, 2nd edition, Upper Saddle River, NJ, Prentice Hall, Inc.

BARNES, J.G.P. (1996) *Programming in Ada 95*, Wokingham, Addison Wesley.

BARTON, P.I. & LEE, C.K. (2002) Modeling, Simulation, Sensitivity Analysis, and Optimization of Hybrid Systems. *ACM Transactions on Modeling and Computer Simulation,* 12, **4,** 256-289.

BASILI, V.R. & HUTCHENS, D.H. (1983) An Empirical Study of a Syntactic Complexity Family. *IEEE Transactions on Software Engineering,* 9, **6,** 664-672.

BELADY, L.A. & LEHMAN, M.M. (1985) Model of Large Program Development. IN Belady, L.A. & Lehman, M.M. (Eds.) *Program Evolution: Processes of Software Change*, London, Academic Press, 165-200.

BELECHEANU, R.A., MUNROE, S., LUCK, M., PAYNE, T., MILLER, T., MCBURNEY, P. & PECHOUCEK, M. (2006) Commercial Applications of Agents: Lessons, Experiences and Challenges. IN Stone, P. & Weiss, G. (Eds.) *Proceedings of the Fifth International Conference on Autonomous Agents and Multiagent Systems,* Hakodate, Japan.

BOEHM, B.W. (1981) *Software Engineering Economics* Englewood Cliffs, NJ, Prentice-Hall.

BOEHM, B.W. (1988) A Spiral Model of Software Development and Enhancement. *IEEE Computer,* 21, **5,** 61-72.

BOEHM, B.W., BROWN, J.R., KASPAR, H., LIPOW, M., MACLEOD, G.J. & MERRITT, M.J. (1980) *Characteristics of Software Quality*, TRW Series of Software Technology, Amsterdam, North-Holland Publishing Company.

BOEHM, B.W., BROWN, J.R. & LIPOW, M. (1976) Quantitative Evaluation of Software Quality. IN *Proceedings of the 2nd International Conference on Software Engineering,* San Francisco, California, IEEE Computer Society Press, 592-605.

BOOCH, G. (1994) *Object-oriented Analysis and Design with Applications*, 2nd edition, Reading, MA, Addison-Wesley.

BORSHCHEV, A. & FILIPPOV, A. (2004) From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools. IN Kennedy, M., Winch, G.W., Langer, R.S., Rowe, J.I. & Yanni, J.M. (Eds.) *22nd International Conference of the System Dynamics Society,* Oxford, England, UK, Wiley.

BOWEN, J.B. (1978) Are Current Approaches Sufficient for Measuring Software Quality? IN *Proceedings Software Quality Assurance Workshop on Functional and Performance Issues*, 148-155.

BOWEN, T.P., POST, J.V., TSAI, J., PRESSON, P.E. & SCHMIDT, R.L. (1983) Software Quality Measurement for Distributed Systems, Guidebook for Software Quality Measurement. *RADC-TR-83-175, Volume II*, Rome Air Development Center, Air Force Systems Command, Griffiths Air Force Base, NY

BRAILSFORD, S.C. & HILTON, N.A. (2000) A Comparison of Discrete Event Simulation and System Dynamics for Modelling Healthcare Systems. IN Riley, J. (Ed.) *Proceedings of ORAHS 2000,* Glasglow Caledonian University, 18-39.

BROOKS, F.P. (1987) No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer,* 20, **4,** 10-19.

BROOKS, R.A. (1986) A Robust Layered Control System For a Mobile Robot. *IEEE Journal of Robotics and Automation,* RA-2, **1,** 14-23.

BROOKS, R.J. & TOBIAS, A.M. (1996) Choosing the Best Model: Level of Detail, Complexity, and Model Performance. *Mathematical and Computer Modelling,* 24, **4,** 1-14.

BROWN, B.B. (1968) Delphi Process: A Methodology Used for the Elicitation of Opinions of Experts. The RAND Corporation. [Online][Accessed on 21 May 2006] Available from: <http://www.rand.org/pubs/papers/2006/P3925.pdf>

BROWN, R. (1988) Calendar Queues: A Fast O(1) Priority Queue Implementation of the Simulation Event Set Problem. *Communications of the ACM,* 31, **10,** 1220-1227.

BUCHANAN, B.G. & SHORTLIFFE, E.H. (1984) *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Reading, MA, Addison-Wesley.

BUSSMANN, S., JENNINGS, N.R. & WOOLDRIDGE, M. (2004) *Multiagent Systems for Manufacturing Control: A Design Methodology*, Springer Series on Agent Technology, Springer Series on Agent Technology, Springer-Verlag.

BUTLER, M.H. & SANDÉN, B.I. (2001) Replacing Processes with Threads in Parallel Discrete Event Simulation. *Technology Review Journal,* Fall/Winter 2001**,** 19-27.

CARSON, J.S., II (2004) Introduction to Modeling and Simulation. IN Ingalls, R.G., Rossetti, M.D., Smith, J.S. & Peters, B.A. (Eds.) *Proceedings of the 2004 Winter Simulation Conference*, 9-16.

CHANDY, K.M. & MISRA, J. (1979) Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering,* 5**,** 440-452.

CHAPMAN, D. (1987) Planning for Conjunctive Goals. *Artificial Intelligence,* 32, **3,** 333-377.

CHIDAMBER, S.R. & KEMERER, C.F. (1991) Towards a Metrics Suite for Object Oriented Design. IN Paepcke, A. (Ed.) *Object-oriented Programming Systems, Languages, and Applications OOPSLA'91,* Phoenix, Arizona, USA, ACM Press New York, NY, USA, 197 - 211

CHOI, T.Y., DOOLEY, K.J. & RUNGTUSANATHAN, M. (2001) Supply Networks and Complex Adaptive Systems: Control Versus Emergence. *Journal of Operations Management,* 19, **3,** 351-366.

CHOY, M.C., SRINIVASAN, D. & CHEU, R.L. (2003) Cooperative, hybrid agent architecture for real-time traffic signal control. *IEEE Transactions on Systems, Man, and Cybernetics,* 33, **5,** 597-607.

CHWIF, L., BARRETTO, M.R.P. & PAUL, R.J. (2000) On Simulation Model Complexity. IN Joines, J.A., Barton, R.R., Kang, K. & Fishwick, P.A. (Eds.) *Proceedings of the 2000 Winter Simulation Conference*, 449-455.

CIOPPA, T.M., LUCAS, T.W. & SANCHEZ, S.M. (2004) Military Applications of Agent-based Simulations. IN Ingalls, R.G., Rossetti, M.D., Smith, J.S. & Peters, B.A. (Eds.) *Proceedings of the 2004 Winter Simulation Conference*, 171-180.

CONNELL, J.H. (1992) SSS: A Hybrid Architecture Applied to Robot Navigation. IN *Proceedings of the 1992 IEEE Conference on Robotics and Automation (ICRA-92)*, 2719-2724.

CONTE, S.D., DUNSMORE, H.E. & SHEN, V.Y. (1986) *Software Engineering Metrics and Models*, Menlo Park, California 94025, The Benjamin/Cummings Publishing Company, Inc.

CONWAY, R.W. (1963) Some Tactical Problems in Digital Simulation. *Management Science,* 10, **1,** 47-61.

COSTANZA, R. & SKLAR, F.H. (1985) Articulation, Accuracy and Effectiveness of Mathematical Models: A Review of Freshwater Wetland Applications. *Ecological Modelling,* 27, **1/2,** 45-68.

COULTER, N.S. (1983) Software Science and Cognitive Psychology. *IEEE Transactions on Software Engineering,* SE-9, **2,** 166-171.

DAVIDSSON, P. (2002) Agent Based Social Simulation: A Computer Science View. *Journal of Artificial Societies and Social Simulation,* 5, **1**.

DE SWAAN ARONS, H. (1999) Knowledge-based Modeling of Ddiscrete-event Simulation Systems. IN Farrington, P.A., Nembhard, H.B., Sturrock, D.T. & Evans, G.W. (Eds.) *Proceedings of the 1999 Winter Simulation Conference*, 591-597.

DE SWAAN ARONS, H. & VAN ASPEREN, E. (2000) Computer Assistance for Model Definition. IN Joines, J.A., Barton, R.R., Kang, K. & Fishwick, P.A. (Eds.) *Proceedings of the 2000 Winter Simulation Conference*, 399-408.

DEMARCO, T. (1979) *Structured Analysis and System Specification*, Upper Saddle River, NJ, USA, Yourdon Press.

DEMARCO, T. (1982) *Controlling Software Projects*, New York, Yourdon Press.

DOJ (2003) *The Department of Justice Systems Development Life Cycle Guidance Document* [Online][Accessed on 12 May 2006] Available from: <http://www.usdoj.gov/jmd/irm/lifecycle/table.htm>

DOOLEY, K.J. (1996) A Nominal Definition of Complex Adaptive Systems. *The Chaos Network,* 8, **1,** 2-3.

DOOLEY, K.J., JOHNSON, T.L. & BUSH, D.H. (1995) TQM, Chaos, and Complexity. *Human Systems Management,* 14, **4,** 1-16.

DUDENHOEFFER, D.D. & JONES, M.P. (2000) A Formation Behavior for Large-scale Micro-robot Force Deployment. IN Joines, J.A., Barton, R.R., Kang, K. & Fishwick, P.A. (Eds.) *The 2000 Winter Simulation Conference*, 972-982.

ECLIPSE (2006) An Open Development Platform. 3.1.4 ed., Eclipse Foundation Inc., Portland, OR 97204 USA.

ETZIONI, O. (1996) Moving Up the Information Food Chain: Deploying Softbots on the Worldwide Web. IN *The Thirteenth National Conference on Artificial Intelligence (AAAI-96),* Portland, Oregon.

EXTEND (2005) Extend Industry, Simulation Package. 6.0.7 ed., Imagine That Inc., San Jose, CA 95119 USA.

FENTON, N.E. & PFLEEGER, S.L. (1997) *Software Metrics: A Rigorous and Practical Approach*, 2nd edition, London, International Thomson Computer Press.

FIPA (2002) *FIPA Abstract Architecture Specification* [Online][Accessed on 05 May 2007] Available from: <http://www.fipa.org/specs/fipa00001/SC00001L.pdf>

FORRESTER, J.W. (1961) *Industrial Dynamics*, Cambridge, Massachusetts, The MIT Press.

FRANKLIN, S. & GRAESSER, A. (1996) Is it an Agent, or Just a Program? A Taxonomy for Autonomous Agents. IN *Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 21-35.

FUJIMOTO, R.M. (1989) The Virtual Time Machine. IN Leighton, F.T. (Ed.) *Proceedings of the First Annual ACM Symposium on Parallel Algorithms and Architectures,* Santa Fe, New Mexico, USA, 199-208.

FUJIMOTO, R.M. (1990) Parallel Discrete Event Simulation. *Communications of the ACM,* 33, **10,** 30-53.

FULTON, E.A., SMITH, A.D.M. & JOHNSON, C.R. (2003) Effect of Complexity on Marine Ecosystem Models. *Marine Ecology Progress Series,* 253**,** 1-6.

GAT, E. (1992) Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. IN *Proceedings of the National Conference on Artificial Intelligence (AAAI-92),* San Jose, CA, 809-815.

GAT, E. (1998) Three-layer architectures. IN Kortenkamp, D., Bonasso, R.P. & Murphy, R. (Eds.) *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, Cambridge, MA, MIT Press, 195-210.

GELL-MANN, M. (1995) What is Complexity? *Complexity,* 1, **1,** 16-19.

GILB, T. (1988) *The Principles of Software Engineering Management*, Wokingham, Addison-Wesley.

GOLDSTEIN, J. (1999) Emergence as a Construct: History and Issues. *Emergence: Complexity and Organization,* 1, **1,** 49-72.

GORDON, T.J. (1994) The Delphi Method. IN *Futures Research Methodology, AC/UNU Project*. [Online][Accessed on 22 May 2007] Available from: <http://www.gerenciamento.ufba.br/Downloads/delphi%20(1).pdf>

GRADY, R.B. (1994) Successfully Applying Software Metrics. *IEEE Computer,* 27, **9,** 18-25.

GRAETTINGER, C.P., GARCIA, S., SIVIY, J., SCHENK, R.J. & SYCKLE, P.J.V. (2002) Using the Technology Readiness Levels Scale to Support Technology

Management in the DoD's ATD/STO Environments (A Findings and Recommendations Report Conducted for Army CECOM). *CMU/SEI-2002-SR-027*, Carnegie-Mellon University, Software Engineering Institute,

GUIMARAES, T. (1983) Managing Application Program Maintenance Expenditures. *Communications of the ACM,* 26, **10,** 739-746.

HALFON, E. (1983a) Is There a Best Model Structure? 1. Modelling the Fate of a Toxic Substance in a Lake. *Ecological Modelling,* 20**,** 135-152.

HALFON, E. (1983b) Is There a Best Model Structure? 2. Comparing the Model Structures of Different Fate Models. *Ecological Modelling,* 20**,** 153-163.

HALSTEAD, M.H. (1977) *Elements of Software Science*, Operating and Programming Systems Series, vol. 7, New York, NY, Elsevier.

HELMER-HIRSCHBERG, O. & RESCHER, N.H. (1958) On the Epistemology of the Inexact Sciences. The RAND Corporation. [Online][Accessed on 21 May 2007] Available from: <http://www.rand.org/pubs/papers/2005/P1513.pdf>

HENDERSON-SELLERS, B. (1996) *Object-oriented metrics: Measures of coplexity*, The Object-Oriented Series, Upper Saddle River, NJ, Prentice Hall.

HENRY, S. & KAFURA, D. (1981) Software Structure Metrics Based on Information Flow. *IEEE Transactions on Software Engineering,* SE-7, **5,** 510-518.

HOFFMANN, A.O.I., JAGER, W. & VON EIJE, J.H. (2007) Social Simulation of Stock Markets: Taking It to the Next Level. *Journal of Artificial Societies and Social Simulation,* [Online][Accessed on 05 April 2007] Available from: <http://jasss.soc.surrey.ac.uk/10/2/7.html>

IEEE (2000) *IEEE 1516-2000 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*, New York, NY, Institute of Electrical and Electronics Engineers, Inc.

IMAGINETHAT (2006) *Extend Overview* [Online][Accessed on 11 September 2006] Available from: <http://www.imaginethatinc.com/prods_overview.html>

IPAS (2007) *IPAS Three Worlds* [Online][Accessed on 21 December 2007] Available from: <http://www.3worlds.org/>

ISO (1991) *Information Technology - Software Product Evaluation - Quality Characteristics and Guide Lines for Their Use*, ISO/TEC IS 9126, Geneva, Switzerland, International Standards Organization.

IZQUIERDO, S.S., IZQUIERDO, L.R. & GOTTS, N.M. (2008) Reinforcement Learning Dynamics in Social Dilemmas. *Journal of Artificial Societies and Social Simulation,* [Online][Accessed on 02 April 2008] Available from: <http://jasss.soc.surrey.ac.uk/11/2/1.html>

JACK (2007) *What is JACK?* [Online][Accessed on 02 May 2007] Available from: <http://www.agent-software.com/shared/products/index.html>

JADE (2006) Java Agent DEvelopment Framework: an Open Source Platform for Peer-to-Peer Agent Based Applications. 3.4.1 ed., TILab, S.p.A., Turin, Italy.

JEFFERSON, D.R. (1985) Virtual Time. *ACM Transactions on Programming Languages and Systems,* 7, **3,** 404-425.

JENNINGS, N.R. (2000) On Agent-based Software Engineering. *Artificial Intelligence,* 117**,** 277-296.

JENNINGS, N.R., SYCARA, K. & WOOLDRIDGE, M. (1998) A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems,* 1, **1,** 7-38.

JENNINGS, N.R. & WOOLDRIDGE, M.J. (1995) Applying Agent Technology. *Applied Artificial Intelligence,* 9, **4,** 351-361.

JONES, C. (1994) Software Metrics: Good, Bad, and Missing. *IEEE Computer,* 27, **9,** 98-100.

JONES, C. (1996) *Programming Languages Table, Version 8.2* [Online][Accessed on 21 December 2007] Available from: <http://www.cs.bsu.edu/homepages/dmz/cs697/langtbl.htm>

JONES, D.W. (1986) An Empirical Comparison of Priority-Queue and Event-Set Implementations. *Communications of the ACM,* 29, **4,** 300-311.

JUNG, H.-W., KIM, S.-G. & CHUNG, C.-S. (2004) Measuring Software Product Quality: A Survey of ISO/IEC 9126. *IEEE Software,* 21, **5,** 88-92.

KENDALL, E.A. (2000) Role Modelling for Agent System Analysis, Design, and Implementation. *IEEE Concurrency,* 8, **2,** 34-41.

KENDALL, E.A. (2001) Agent Software Engineering with Role Modelling. IN Ciancarini, P. & Wooldridge, M.J. (Eds.) *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*, Springer-Verlag Berlin Heidelberg, 163-169.

KINNY, D., GEORGEFF, M. & RAO, A. (1996) A methodology and modelling technique for systems of BDI agents. IN Van De Velde, W. & Perram, J.W. (Eds.) *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Springer-Verlag: Berlin, Germany, 56-71.

KITCHENHAM, B., PICKARD, L. & PFLEEGER, S.L. (1995) Case Studies for Method and Tool Evaluation. *IEEE Software,* 12, **4,** 52-62.

KITCHENHAM, B.A. (1989) Software Quality Assurance. *Microprocessors and Microsystems,* 13**,** 373-381.

KITCHENHAM, B.A. & WALKER, J.G. (1989) A Quantitative Approach to Monitoring Software Development. *Software Engineering Journal,* 4, **1,** 2-13.

KLÜGL, F., HERRLER, R. & OECHSLEIN, C. (2003) From Simulated to Real Environments: How to Use SeSAm for Software Development. IN Schillo, M., Klusch, M., Müller, J. & Tianfield, H. (Eds.) *Proceedings of the First German Conference on Multiagent System Technologies, MATES 2003,* Erfurt, Germany, Springer-Verlag GmbH, 13-24.

KNUTH, D.E. (1973) *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd edition, Reading, MA, Addison-Wesley

KOKOL, P., PODGORELEC, V., ZORMAN, M. & PIGHIN, M. (1999) Alpha - a Generic Software Complexity Metric. IN Kusters, R.J., Cowderoy, A., Heemstra, F.J. & Van Veenendal, E.P.W.M. (Eds.) *Proceedings of ESCOM-SCOPE 99,* Herstmonceux, England, 397-405.

KRAHL, D. (2001) The Extend Simulation Environment. IN Peters, B.A., Smith, J.S., Medeiros, D.J. & Rohrer, M.W. (Eds.) *Proceedings of the 2001 Winter Simulation Conference*, 217-225.

KROGSTIE, J., JAHR, A. & SJØBERG, D.I.K. (2006) A Longitudinal Study of Development and Maintenance in Norway: Report from the 2003 Investigation. *Information and Software Technology,* 48, **11,** 993-1005.

KUHL, F., WEATHERLY, R. & DAHMANN, J. (1999) *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Upper Saddle River, NJ 07458, Prentice Hall PTR.

LAW, A.M. & KELTON, W.D. (1999) *Simulation Modeling & Analysis*, 3rd edition, McGraw-Hill Series in Industrial Engineering and Management Science, New York, McGraw-Hill, Inc.

LAW, A.M. & MCCOMAS, M.G. (2003) How the ExpertFit Distribution-fitting Software Can Make Your Simulation Models More Valid. IN Chick, S., Sánchez, P.J., Ferrin, D. & Morris, D.J. (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, 169-174.

LI, W. & HENRY, S. (1993) Object Oriented Metrics Which Predict Maintainability. *Journal of Systems Software,* 23**,** 111-122.

LINSTONE, H.A. & TUROFF, M. (2002) *The Delphi Method: Techniques and Applications* [Online][Accessed on 08 September 2006] Available from: <http://www.is.njit.edu/pubs/delphibook/delphibook.pdf>

LORENZ, M. (1993) *Object-oriented software development: a practical guide*, Englewood Cliffs, NJ, Prentice Hall.

LOSTWAX (2005) *Lost Wax Agent Framework* [Online][Accessed on 24 October 2005] Available from: <http://www.lostwax.com/agents/framework/>

LOW, Y.H., LIM, C.C., CAI, W., HUANG, S.Y., HSU, W.J., JAIN, S. & TURNER, S.J. (1999) Survey of languages and runtime libraries for parallel discrete-event simulation. *Simulation,* 72, **3,** 170-186.

LUCK, M., MCBURNEY, P., SHEHORY, O. & WILLMOTT, S. (2005) *Agent Technology Roadmap: A Roadmap for Agent Based Computing* [Online][Accessed on 22 December 2007] Available from: <http://www.agentlink.org/roadmap/al3rm.pdf>

MACAL, C.M. & NORTH, M.J. (2006) Tutorial on agent-based modelling and simulation Part 2: how to model with agents. IN Perrone, L.F., Wieland, F.P., Liu, J., Lawson, B.G., Nicol, D.M. & Fujimoto, R.M. (Eds.) *Proceedings of the 2006 Winter Simulation Conference*, 73-83.

MARIA, A. (1997) Introduction to modeling and simulation. IN Andradóttir, S., Healy, K.J., Withers, D.H. & Nelson, B.L. (Eds.) *Proceedings of the 1997 Winter Simulation Conference*, 7-13.

MARÍN, M. (1997) An empirical comparison of priority queue algorithms. *Technical Report PRG-TR-10-97.* Oxford University.

MARTIN, J. (1991) *Rapid Application Development*, Indianapolis, IN, USA, Macmillan Publishing Co., Inc.

MARTIN, P. (1999) The modelling of tactics and procedures using a component based system. IN Farrington, P.A., Nembhard, H.B., Sturrock, D.T. & Evans, G.W. (Eds.) *The 1999 Winter Simulator Conference*, 1131-1136.

MATARIC, M.J. (1992) Behaviour-based control: Main properties and implications. IN *Proceedings of the IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems,* Nice, France, 46-54.

MATARIĆ, M.J. (1997) Behaviour-based control: examples from navigation, learning, and group behaviour. *Journal of Experimental & Theoretical Artificial Intelligence,* 9, **2,** 323-336.

MATHWORKS (2007) *Simulink - Simulation and Model-Based Design* [Online][Accessed on 01 September 2007] Available from: <http://www.mathworks.com/products/simulink/>

MCCABE, T.J. (1976) A complexity measure. *IEEE Transactions on Software Engineering,* SE-2, **4,** 308-320.

MCCABE, T.J. & BUTLER, C.W. (1989) Design complexity measurement and testing. *Communications of the ACM,* 32, **12,** 1415-1425.

MCCALL, J.A., RICHARDS, P.K. & WALTERS, G.F. (1977) Factors in software quality. *Vols I, II, III, US Rome Air Development Center Reports NTIS AD/A-049 014, 015, 055*,

MILLER, D.P., FIRBY, R.J., FISHWICK, P.A. & ROTHENBERG, J. (1992) AI: what simulationists really need to know. *ACM Transactions on Modeling and Computer Simulation,* 2, **4,** 269-284.

MISRA, J. (1986) Distributed discrete-event simulation. *ACM Computing Surveys,* 18, **1,** 39-65.

MOD (2007) *Acquisition Management System: Technology Management Guidance for the UK MOD Defence Acquisition Community (Annexe A)* [Online][Accessed on 21 December 2007] Available from: <http://www.ams.mod.uk/content/docs/techman/content/trlann/trlanna.pdf>

MORECROFT, J. & ROBINSON, S. (2006) Comparing discrete-event simulation and system dynamics: modelling a fishery. IN Robinson, S., Taylor, S., Brailsford, S. & Garnett, J. (Eds.) *Proceedings of the 2006 OR Society Simulation Workshop*, The OR Society.

MOSTERMAN, P.J. (1999) An Overview of Hybrid Simulation Phenomena and Their Support by Simulation Packages. IN Vaandrager, F.W. & Schuppen, J.H.V. (Eds.) *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science, vol. 1569, Berlin, Springer-Verlag, 165-177.

NANCE, R.E. & SARGENT, R.G. (2002) Perspectives on the evolution of simulation. *Operations Research,* 50, **1,** 161-172.

NEILL, C.J. & LAPLANTE, P.A. (2003) Requirements Engineering: the State of the Practice. *IEEE Software,* 20, **6,** 40-45.

NEUMANN, P.G. (1993) Modeling and Simulation. *Communications of the ACM,* 36, **4,** 124.

NEWELL, A. (1982) The Knowledge Level. *Artificial Intelligence,* 18**,** 87-127.

NOSEK, J.T. & PALVIA, P. (1990) Software Maintenance Management: Changes in the Last Decade. *Journal of Software Maintenance: Research and Practice,* 2, **3,** 157-174.

ODELL, J., PARUNAK, H.V.D. & BAUER, B. (2000) Representing agent interaction protocols in UML. IN Ciancarini, P. & Wooldridge, M. (Eds.) *Proceedings of the First International Workshop (AOSE-2000)*, Springer-Verlag: Berlin, Germany.

OMAN, P.W. & HAGEMEISTER, J. (1992) Metrics for Assessing a Software System's Maintainability. IN *Proceedings of the Conference on Software Maintenance*, IEEE Computer Society Press, 337-344.

ÖREN, T.I. (1977) Simulation - as it has been, and should be. *Simulation,* 29, **5,** 182-183.

ÖREN, T.I. (1979) Concepts for advanced computer assisted modelling. IN Zeigler, B.P., Elzas, M.S., Kir, G.J. & Ören, T.I. (Eds.) *Methodology in Systems Modelling and Simulation*, 29-55.

ÖREN, T.I. (1986) Knowledge bases for an advanced simulation environment. IN Luker, P.A. & Adelsberger, H.H. (Eds.) *Intelligent Simulation Environments*, 16-22.

PARK, R. (1992) Software size measurement: a framework for counting source statements. *Technical Report CMU/SEI-92-TR-20*, Software Engineering Institute, Carnegie Mellon, Pittsburgh

PARUNAK, H.V.D., SAVIT, R. & RIOLO, R.L. (1998) Agent-based modeling vs. equation-based modeling: a case study and users' guide. IN Sichman, J.S., Conte, R. & Gilbert, N. (Eds.) *Multi-Agent Systems and Agent-Based Simulation: First International Workshop, MABS '98,* Paris, France, Springer-Verlag GmbH, 10-25.

PATHAK, S.D., DILTS, D.M. & BISWAS, G. (2003) A multi-paradigm simulator for simulating complex adaptive supply chain networks. IN Chick, S., Sánchez, P.J., Ferrin, D. & Morrice, D.J. (Eds.) *Proceedings of the 2003 Winter Simulation Conference*, 808-816.

PRECHELT, L. (2000) An Empirical Comparison of Seven Programming Languages. *Computer,* 33, **10,** 23-29.

REFSGAARD, J.C. & HENRIKSEN, H.J. (2004) Modelling guidelines - terminology and guiding principles. *Advances in Water Resources,* 27, **1,** 71-82.

REYNOLDS, P.F., JR. (1988) A spectrum of options for parallel simulation. IN Abrams, M., Haigh, P. & Comfort, J. (Eds.) *Proceedings of the 1988 Winter Simulation Conference*, 325-332.

ROBINSON, S. (2003) *Simulation: The practice of model development and use*, Wiley, Chichester, UK.

ROLLS-ROYCE (2005a) *Aero repair and overhaul* [Online][Accessed on 04 July 2006] Available from: <http://ir.rolls-royce.com/rr/investors/analyst/investorday/paterson.pdf>

ROLLS-ROYCE (2005b) *Focused investment in technology* [Online][Accessed on 04 July 2006] Available from: <http://www.investis.com/rr/downloads/smith.pdf>

ROLLS-ROYCE (2008) *Trent 800 Operational Summary* [Online][Accessed on 01 May 2008] Available from: <http://www.rolls-royce.com/civil_aerospace/products/airlines/trent800/operational.jsp>

ROMBACH, H.D. (1987) A Controlled Experiment on the Impact of Software Structure on Maintainability. *IEEE Transactions on Software Engineering,* SE-13, **3,** 89-94.

ROMBACH, H.D. (1990) Design Measurement: Some Lessons Learnt. *IEEE Software***,** 17-25.

RÖNNGREN, R. & AYANI, R. (1997) A Comparative Study of Parallel and Sequential Priority Queue Algorithms. *ACM Transactions on Modeling and Computer Simulation,* 7, **2,** 157-209.

RÖNNGREN, R., RIBOE, J. & AYANI, R. (1991) Lazy Queue: An Efficient Implementation of the Pending-event Set. IN *Proceedings of the 24th Annual Simulation Symposium,* New Orleans, USA, IEEE, 194-204.

ROTHENBERG, J. (1991) Tutorial: artificial intelligence and simulation. IN Nelson, B.L., Kelton, W.D. & Clark, G.M. (Eds.) *Proceedings of the 1991 Winter Simulation Conference*, 218-222.

ROYCE, W.W. (1987) Managing the development of large software systems: concepts and techniques. IN *Proceedings of the 9th International Conference on Software Engineering,* Monterey, CA, USA, IEEE Computer Society Press, 328-338.

RUSSELL, S. & NORVIG, P. (2003) *Artificial Intelligence: A Modern Approach*, 2nd edition, Upper Saddle River, NJ, Prentice Hall.

SARGENT, R.G. (2007) Verification and Validation of Simulation Models. IN Henderson, S.G., Biller, B., Hsieh, M.-H., Shortle, J., Tew, J.D. & Barton, R.R. (Eds.) *Proceedings of the 2007 Winter Simulation Conference*, 124-137.

SCHLESINGER, S., CROSBIE, R.E., GAGNE, R.E., INNIS, G.S., LALWANI, C.S., LOCH, J., SYLVESTER, J., WRIGHT, R.D., KHEIR, N. & BARTOS, D. (1979) Terminology for Model Credibility. SCS Technical Committee on Model Credibility. *Simulation,* 32, **3,** 103-104.

SDX (2005) *SDX High Performance Computing* [Online][Accessed on 01 September 2007] Available from: <http://www.sdynamix.com/index.html>

SEI (2000) *Cyclomatic Complexity* [Online][Accessed on 15 December 2007] Available from: <http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html>

SHANNON, C.E. (1951) Prediction and entropy of printed English. *Bell Systems Technical Journal,* 30**,** 50-64.

SHANNON, R.E. (1977) Introduction to simulation languages. IN Highland, H.J., Sargent, R.G. & Schmidt, J.W. (Eds.) *Proceedings of the 1977 Winter Simulation Conference*, 14-20.

SHANNON, R.E. (1987) Models and artificial intelligence. IN Thesen, A., Grant, W. & Kelton, W.D. (Eds.) *Proceedings of the 1987 Winter Simulation Conference*, 16-24.

SHEPPERD, M.J. & INCE, D.C. (1989) An Empirical and Theoretical Analysis of an Information Flow-based System Design Metric. IN *Proceedings of the European Software Engineering Conference '89*, Springer Berlin / Heidelberg, 86-99.

SIMON, H.A. (1998) *The sciences of the artificial*, Third edition, Cambridge, Massachusetts, The MIT Press.

SLEATOR, D.D. & TARJAN, R.E. (1985) Self-adjusting binary search trees. *Journal of the Association for Computing Machinery,* 32, **3,** 652-686.

SLEATOR, D.D. & TARJAN, R.E. (1986) Self-adjusting Heaps. *SIAM Journal on Computing,* 15, **1,** 52-59.

SOMERVILLE, I. (2001) *Software Engineering*, 6th edition, London, Addison-Wesley.

STEINMAN, J.S. (1994) Discrete-event simulation and the event horizon. IN *Workshop on Parallel and Distributed Simulation*, ACM Press, New York, NY, USA, 39-49.

STEINMAN, J.S. (1996) Discrete-Event Simulation and the Event Horizon: Part 2: Event List Management. IN *Proceedings of the Tenth Workshop on Parallel and Distributed Simulation,* Philadelphia, Pennsylvania, USA, IEEE Computer Society, 170-178.

STELLMAN, A. & GREENE, J. (2005) *Applied Software Project Management*, Theory in Practice, O'Reilly Media, Inc.

STEVENS, W.P., MYERS, G.J. & CONSTANTINE, L.L. (1974) Structured design. *IBM Systems Journal,* 13, **2,** 115-139.

STOCKLE, C.O. (1992) Canopy photosynthesis and transpiration estimates using radiation interception models with different levels of detail. *Ecological Modelling,* 60, **1,** 31-44.

TAYLOR, P., EVANS-GREENWOOD, P. & ODELL, J. (2005) Agents in the Enterprise. *Australian Software Engineering Conference, ASWEC 2005.* Brisbane, Australia.

TEWOLDEBERHAN, T.W., VERBRAECK, A., VALENTIN, E. & BARDONNET, G. (2002) An evaluation and selection methodology for discrete-event simulation software. IN Yücesan, E., Chen, C.-H., Snowdon, J.L. & Charnes, J.M. (Eds.) *2002 Winter Simulation Conference*, 67-75.

THOYER, S., MORARDET, S., RIO, P., SIMON, L., GOODHUE, R. & RAUSSER, G. (2001) A Bargaining Model to Simulate Negotiations between Water Users. *Journal of Artificial Societies and Social Simulation,* [Online][Accessed on 12 September 2006] Available from: <http://jasss.soc.surrey.ac.uk/4/2/6.html>

TOBIAS, R. & HOFMANN, C. (2004) Evaluation of free Java-libraries for social-scientific agent based simulation. *Journal of Artificial Societies and Social Simulation,* 7, **1**.

TSATSOULIS, C. (1990) A review of artificial intelligence in simulation. *ACM SIGART Bulletin,* 2, **1,** 115-121.

VAKAS, D., PRINCE, J., BLACKSTEN, H.R. & BURDICK, C. (2001) Commander behavior and course of action selection in JWARS. IN Peters, B.A., Smith, J.S., Medeiros, D.J. & Rohrer, M.W. (Eds.) *Proceedings of the 2001 Winter Simulation Conference*, 697-705.

VAUCHER, J.G. (1985) Views of modelling: comparing the simulation and AI approaches. IN *Proceedings of the SCS Multiconference in Artificial Intelligence, Graphics, and Simulation*, 3-7.

WAKE, S. & HENRY, S. (1988) A model based on software quality factors which predicts maintainability. IN *Conference on Software Maintenance,* Scottsdale, Arizona, USA, IEEE, 382-387.

WELKER, K.D. (2001) The Software Maintainability Index Revisited. *Crosstalk: The Journal of Defense Software Engineering,* 14, **8,** 18-21.

WILLIAMS, J.D. (1994) Metrics for object-oriented projects. IN *Proceedings of OOP'94 and C++ World,* Lisbon, Portugal, 253-258.

WOOLDRIDGE, M. (1997) Agent-based software engineering. *IEE Proceedings Software Engineering,* 144, **1,** 26-37.

WOOLDRIDGE, M. & CIANCARINI, P. (2001) Agent-Oriented Software Engineering: The State of the Art. IN*Agent-Oriented Software Engineering*, Berlin/Heidelberg, Springer, 52-82.

WOOLDRIDGE, M. & JENNINGS, N.R. (1995) Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review,* 10, **2,** 115-152.

WOOLDRIDGE, M., JENNINGS, N.R. & KINNY, D. (1999) A Methodology for Agent-Oriented Analysis and Design. IN Etzioni, O., Muller, J. & Bradshaw, J.M. (Eds.) *Proceedings of the Third International Conference on Autonomous Agents (Agents 99),* Seattle, Washington, ACM, 69-76.

WOOLDRIDGE, M.J. & JENNINGS, N.R. (1999) Software engineering with agents: pitfalls and pratfalls. *IEEE Internet Computing,* 3, **3,** 20-27.

WSC (2007) *Winter Simulation Conference Programs with Full Papers* [Online][Accessed on 01 December 2007] Available from: <http://www.informs-cs.org/wscpapers.html>

XJTECHNOLOGIES (2006) *AnyLogic Overview* [Online][Accessed on 21 August 2006] Available from: <http://www.xjtek.com/anylogic/>

YIN, R.K. (2003) *Case Study Research: Design and Methods*, Third edition, London, Sage Publications, Inc.

YOURDON, E. & CONSTANTINE, L.L. (1979) *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*, Englewood Cliffs, NJ, Prentice-Hall.

ZHANG, W. & LUCK, S.J. (2008) Discrete fixed-resolution representations in visual working memory. *Nature,* [Online][Accessed on 04 April 2008] Available from: <http://www.nature.com/nature/journal/vaop/ncurrent/pdf/nature06860.pdf>

Bibliography

AGENTLINK (2007) *Agentlink: European Co-ordination Action for Agent-based Computing* [Online][Accessed on 21 May 2007] Available from: <http://www.agentlink.org/index.php>

AGENTWEB (2007) *Bookmarks for agent information* [Online][Accessed on 04 September 2007] Available from: <http://agents.umbc.edu/awbookmarks.html>

AITOPICS (2007) *AI Topics: A dynamic library of introductory information about Artificial Intelligence* [Online][Accessed on 10 December 2007] Available from: <http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/HomePage>

AMOUZEGAR, M.A. & GALWAY, L.A. (2003) Supporting Expeditionary Aerospace Forces: Engine Maintenance Systems Evaluation (EnMasse): A User's Guide. *MR-1614*, The RAND Corporation,

APRIL, J., BETTER, M., GLOVER, F. & KELLY, J. (2004) New Advances and Applications for Marrying Simulation and Optimization. IN Ingalls, R.G., Rossetti, M.D., Smith, J.S. & Peters, B.A. (Eds.) *Proceedings of the 2004 Winter Simulation Conference*, 80-86.

BANDECCHI, M. (2007) Concurrent Engineering at ESA: from the Concurrent Design Facility (CDF) to a Distributed Virtual Facility. IN *The 14th ISPE International Conference on Concurrent Engineering,* Sao Jose' dos Campos, SP, Brazil.

BANSLER, J.P. & BØDKER, K. (1993) A reappraisal of structured analysis: design in an organizational context. *ACM Transactions on Information Systems,* 11, **2,** 165-193.

BASILI, V.R., BRIAND, L.C. & MELO, W.L. (1996) A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering,* 22, **10,** 751-761.

BASILI, V.R. & ROMBACH, H.D. (1988) The TAME Project: Towards Improvement-oriented Software Environments. *IEEE Transactions on Software Engineering,* 14, **6,** 758-773.

DAVIDSSON, P. (2000) Multi Agent Based Simulation: Beyond Social Simulation. IN Moss, S. & Davidsson, P. (Eds.) *Multi-Agent-Based Simulation: Second International Workshop, MABS 2000,* Boston, MA, USA, Springer Berlin / Heidelberg, 97-107.

JASSS (2008) *The Journal of Artificial Societies and Social Simulation: An inter-disciplinary journal for the exploration and understanding of social* *processes by means of computer simulation* [Online][Accessed on 21 May 2008] Available from: <http://jasss.soc.surrey.ac.uk/JASSS.html>

JENNINGS, N.R. (2001) An Agent-based Approach for Building Complex Software Systems. *Communications of the ACM,* 44, **4,** 35-41.

JENNINGS, N.R. (2008) *Professor Nicholas Jennings: Publications* [Online][Accessed on 29 October 2007] Available from: <http://www.ecs.soton.ac.uk/people/nrj/publications>

LEE, L.H., HUANG, H.C., LEE, C., CHEW, E.P., JARUPHONGSA, W., YONG, Y.Y., LIANG, Z., LEONG, C.H., TAN, Y.P., NAMBURI, K., JOHNSON, E. & BANKS, J. (2003) Proceedings of the Discrete Event Simulation Model for Airline Operations: SIMAIR. IN Chick, S., Sánchez, P.J., Ferrin, D. & Morrice, D.J. (Eds.) *2003 Winter Simulation Conference*, 1656-1662.

LEGATO, P. & MAZZA, R.M. (2001) Berth Planning and Resources Optimisation at a Container Terminal via Discrete Event Simulation. *European Journal of Operational Research,* 133, **3,** 537-547.

LINSTONE, H.A. & TURROFF, M. (2002) *The Delphi Method: Techniques and Applications* [Online][Accessed on 08 September 2006] Available from: <http://www.is.njit.edu/pubs/delphibook/delphibook.pdf>

LUCK, M. (2007) *Michael Luck Publications* [Online][Accessed on 21 June 2007] Available from: <http://www.dcs.kcl.ac.uk/staff/mml/publications/publications.html>

MAES, P. (1994) Agents that reduce work and information overload. *Communications of the ACM,* 37, **7,** 30-40.

MÖLLER, B. & LÖF, S. (2006) A Management Overview of HLA Evolved Web Service API. IN *Proceedings of 2006 Fall Simulation Interoperability Workshop,* Orlando, Florida, USA, Simulation Interoperability Standards Organization.

NICOL, D.M. (1988) High performance parallelized discrete event simulation of stochastic queueing networks. IN Abrams, M., Haigh, P. & Comfort, J. (Eds.) *Proceedings of the 1988 Winter Simulation Conference*, 306-314.

NICOL, D.M. (1996) Principles of conservative parallel simulation. IN Charnes, J.M., Morrice, D.J., Brunner, D.T. & Swain, J.J. (Eds.) *Proceedings of the 1996 Winter Simulation Conference*, 128-135.

PAPAMICHAEL, K. & PROTZEN, J.-P. (1993) The Limits of Intelligence in Design. *4th International Symposium on System Research, Informatics and Cybernatics.* Baden-Baden, Germany.

RIZZOLI, A.E. (2007) *A collection of modelling and simulation resources on the Internet* [Online][Accessed on 08 November 2007] Available from: <http://www.idsia.ch/~andrea/simtools.html#vissim>

ROCHA, L.M. (2004) *From Artificial Life to Semiotic Agent Models: Review and Research Directions* [Online][Accessed on 21 May 2006] Available from: <http://informatics.indiana.edu/rocha/sim/review.html>

SHANTHIKUMAR, J.G. & SARGENT, R.G. (1983) A unifying view of hybrid simulation/analytic models and modeling. *Operations Research,* 31, **6,** 1030-1052.

SHEN, W. & NORRIE, D.H. (1998) A hybrid agent-oriented infrastructure for modeling manufacturing enterprises. IN *Proceedings of the Knowledge Acquisition Workshop (KAW-98),* Banff, Canada.

SIMON, H.A. *Herbert A. Simon 1916-2001* [Online][Accessed on 01 Feb 2008] Available from: <http://www.psy.cmu.edu/psy/faculty/hsimon/hsimon.html>

TESFATSION, L. (2008) *General Software and Toolkits: Agent-Based Computational Economics and Complex Adaptive Systes* [Online][Accessed on 21 May 2008] Available from: <http://www.econ.iastate.edu/tesfatsi/acecode.htm>

WOOLDRIDGE, M. (2002) *An Introduction to MultiAgent Systems* [Online][Accessed on 24 August 2005] Available from: <http://www.csc.liv.ac.uk/~mjw/pubs/imas/>

WOOLDRIDGE, M. (2008) *Mike Wooldridge - Publications* [Online][Accessed on 21 May 2008] Available from: <http://www.csc.liv.ac.uk/~mjw/pubs/>