Approaches to modeling the gas-turbine maintenance process.

Tai-Tuck Yu and James P Scanlan
School of Engineering Sciences
University of Southampton, UK

Richard M Crowder and Gary B Wills
School of Electronics and Computer Science
University of Southampton, UK

e-mail: ttyu@soton.ac.uk, j.p.scanlan@soton.ac.uk, rmc@ecs.soton.ac.uk, gbw@ecs.soton.ac.uk

Abstract

Discrete-event modeling has long been used for logistics and scheduling problems, while multi--agent modelling closely matches human decision-making process. In this paper a metric-based comparison between the traditional discrete-event and the emerging agent-based modeling approaches is reported. The case study involved the implementation of two functionally identical models based on a realistic, non-trivial, civil aircraft gas turbine global repair operation. The size, structural complexity, and coupling metrics from the two models were used to gauge the benefits and drawbacks of each modeling paradigm. The agent-based model was significantly better than the discrete-event model in terms of execution times, scalability, understandability, modifiability, and structural flexibility. In contrast, and importantly in an engineering context, the discrete-event model guaranteed predictable and repeatable results and was comparatively easy to test because of its single-threaded operation. However, neither modeling approach on its own possesses all these characteristics nor can each handle the wide range of resolutions and scales frequently encountered in problems exemplified by the case study scenario. It is recognized that agent-based modeling can emulate high-level human decision-making and communication closely while discrete-event modeling provides a good fit for low-level sequential processes such as those found in manufacturing and logistics.

1 Introduction

The modeling of problems in an engineering context, for example, supply chains, manufacturing processes, and logistics, is very frequently dealt with by the use of the discrete-event modeling technique. The existence of numerous mature, highly developed, and commercially available discrete-event modeling software packages (for instance, those described in Law and Kelton [1] indicates that this modeling paradigm is widely used. While it is evidently suited to process-oriented problems, it appears not to be as well matched for problems encountered in the social sciences or where interacting human decision making predominates. Instead, the concept of agent-based modeling is usually selected for modeling these human-oriented problems, such as found in the engineering design process [2, 3]. They pose the challenge as to when and where discrete-event and agent-based modeling may be applied for efficient decision support. To address this, it is reasonable to begin by carrying out an objective comparison of the two modeling techniques

Our work will be discussed in the context of aero-engine manufacturing where a fundamental shift is occurring, from selling products to providing services. For example, Rolls-Royce set the target of making half its engine fleet subject to profitable long-term service agreements by 2010 [4, 5]. Essential to success in this market shift is to ensure that new products are optimized for the aftermarket. In other words, new products must be designed to provide lower and more predictable maintenance costs. To minimize maintenance costs throughout the engine's life cycle, engineers must obtain and actively use knowledge gained from maintenance histories of similar products during the design phase of new products as discussed in our previous paper [6], as well as a better understanding of the maintenance process through its modeling.

As a technology, agent-based modeling is beginning to emerge from the software laboratory into the harsh, commercial world. It has been assessed as having attained this stage of maturity both according to the UK Ministry of Defence guidelines [7] and by its practitioners [8]. Nevertheless, it is a new technology which, in the main, still lacks the software tools and trained human resources needed to engage the marketplace. This assertion is evident in Belecheanu et al [9] which contains several case studies of agent systems developed by highly specialized software companies for production use in manufacturing, logistics, training, and energy production and distribution. Recently, Taylor et al [10] highlighted the gulf which exists between “the promising world of agents and the uncompromising world of the enterprise” and contemporaneously, Luck et al [8] admitted that mainstream deployment of agent technologies by 2010 would seem optimistic.

Various claims about the versatility of agent-based modeling and its obvious advantages over other more established modeling paradigms like System Dynamics and discrete-event modeling have been made by its proponents, for example Bonabeau [11] and Davidsson [12]. Despite that clamour, there is nevertheless a healthy awareness of the pitfalls [13] when agent technology is applied inappropriately and so advice is usually given to guide modelers as to when agent-based modeling is most beneficial [11, 13, 14]. In a system where high-level human decision making is dominant, agent--based modeling appears to provide the easier solution where other paradigms can struggle to implement a solution.

In recent years, comparative studies have also been carried out between agent-based modeling and System Dynamics [15], and between System Dynamics and discrete-event modeling [16, 17]. However, the previously mentioned assessments and brief case studies, and these more rigorous studies are qualitative investigations. As rightly acknowledged by Jennings [18], there is a yet unmet need to put agent technology on firmer footing by making objective, quantitative comparisons with existing software concepts. Therefore, this paper addresses the oversight by comparing an agent-based model (ABM) and a functionally equivalent traditional discrete-event model (DEM) in the context of a fleet maintenance operation over a typical engine production lifetime. The metrics like lines of executable code, branching, coupling, and execution times are standard measures for software complexity, maintainability, and performance [19]. They are elaborated in Section 3.

The remainder of this paper is set out as follows; Section 2 discusses the modeling paradigms which can be used for the simulation of a gas turbine repair program. In addition, the notion of agenthood is considered in detail. Sections 4 and 5 describes the case study which is used for comparing discrete-event and agent-based modeling, together with the results of the case study including the code metrics extracted from the functionally identical agent-based and the discrete-event models. Finally Section 6 and 7 presents a discussion of the results and the overall conclusions.

2 Modelling paradigms for gas turbine maintenance

Figure 1shows a general classification of paradigms usually encountered in modeling and simulation literature. Of the modelling paradigms, DEM meets the demands of the application most closely in terms of execution efficiency. The model considers a large number of items and stochastic events associated with each have to be tracked individually. Their operational lives are often much larger than the time needed to inspect, repair, or replace them. Hence, there will be long intervals of inactivity between events. It is more efficient to go immediately from one event to the next rather than to progress in a number of regular time steps between events. Although there are numerous sub-classes as well as combinations of them, they are too specialized and so descriptions of the modeling paradigms will be confined to the lowest tier depicted in this figure.

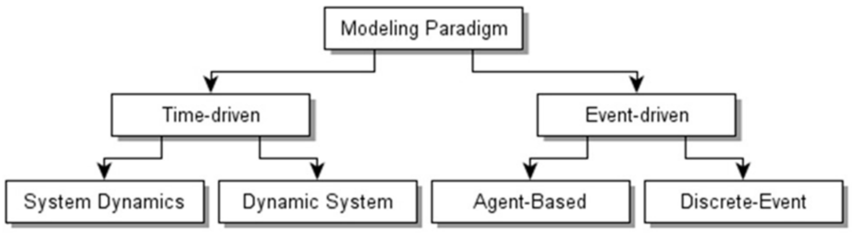


Figure 1. A high-level taxonomy of modeling paradigms.

A model built on the Dynamic System paradigm involves the iterative evaluation of a system of equations representing physical systems such as a pendulum or an electronic circuit. The System Dynamics [20] paradigm can be used to model the behaviour of large complex systems such as the economy and ecosystems. The technique achieves this through a network of stocks and flows which are affected by feedback loops and time delays. These two time-driven or continuous modelling paradigms are not suited to the problem investigated in this case study.

The paradigm most frequently used for modeling supply chain/logistics problems encountered in jet engine life cycle costing is event-driven rather than time-driven. The ABM and DEM may be implemented in either a procedural programming language like FORTRAN or C, or an object-oriented language like Java or C++. In the case of the ABM, although procedural languages can be used, the undoubted preponderance of implementations are object-oriented.

Although it is possible to represent a problem with any of the modeling paradigms, what influences the eventual choice of method will include the data inputs available, the outputs required, and the response times acceptable. For instance, while the System Dynamics paradigm can be used to model crowd behavior, the average input values it requires thus implying homogeneity, do not enable spatial, clustered or heterogeneous crowd behavior to result. If such behavior needs to be investigated, it is more appropriately and perhaps more easily, modeled as a multiagent system.

It is conceivable that within a problem to be modeled, the levels of description can range from fine-grained, micro-scale systems (e.g. cooling air flow in a turbine blade) to coarse-grained, macro-scale ones (e.g. the multi-fleet operation management role). Such a scenario can be encountered at some stage in the life of a gas turbine where a minor detail design change can sometimes cause a huge disruption to the planned maintenance activity of one or more engine fleets. As there is no universal paradigm which is able to handle all model resolutions and scales, it is important to minimize the mismatch between a system within the problem domain and the modeling concept as small as possible. Simulation modeling remains more art than science when it comes to balancing the level of model detail with the requirements of a model. Hence, to maintain fidelity and maximize resources, it does happen that different parts of a model are implemented in different modeling paradigms. It is not unusual to find time-driven subsystems integrated with event-driven subsystems in a large, complex model.

Since discrete-event modelling is well known and is dealt with the literature [1, 21, 22], it is not necessary to detail it here except to note that the scheduling and sequencing of events in a DEM are managed by a central event-list. Discrete-event modeling is centered on processes which are implemented using primitives such as queues, time delays, batching, and unbatching. Resources, like services and skills, are expended on the entities as they flow through the processes under the direction of the event-list.

2.1 Overview of agent based modeling

The common misperception of agents lies in the persistent absence of a generally agreed definition. Much has been written and there has been much debate about what essentially constitutes an agent. This is exemplified by Franklin and Graesser [23] who set out eleven attempts by various researchers at characterizing one. More recently, Jennings [18] offered this concise description which an increasing number of researchers in the agent community have found to be useful;

"*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."*

This definition sets out from a software engineering perspective five essential features, all of which must be present, for an object to be considered an agent.

The characteristic which has achieved consensus is that an agent must be capable of autonomous action. This is in contrast to conventional computer programs which are absolutely obedient to each other. The apparent ceding of absolute control over a piece of software by its designer endows a multiagent system with the ability to behave with various degrees of cooperation or otherwise. The distinction is not often made between an ABM of a real world system and an agent system used in a real world application, or simulated and performing agent respectively [24]. In the former, an event which causes a change in the agent environment can also be made to schedule an action in the agents. Hence, in an ABM, there is no need to monitor continuously for events. As these agents reside in an artificial world where events are predictable, it is valid to implement them as event-driven agents. In contrast, agents in a real world system cannot predict when changes to their environment will occur and therefore they need to monitor their environment frequently if they are to respond in a timely manner. Therefore, by necessity, such an agent system is time-driven.

Some notable features of an ABM are as follows;

1 Entities and agents are involved.
   a Entities are passive objects as found in a traditional DEM.
   b Agents are active objects. Depending on the requirements of the system to be modeled, their architecture can be reactive (or reflex), deliberative (or reasoning), or a hybrid of both [25, 26].
2 Agents do not communicate directly with each other. Messages are always passed between them by way of a common environment. In a time-driven model, agents respond to incoming messages and also assess how they should react to the dynamic conditions of their environment at each simulation clock tick in between scheduled events.
3 Agents run concurrently in their own independent threads of execution.
4 Agents can be created or destroyed during a simulation run thus endowing an ABM with the ability to change its own structure.
5 An ABM has decentralized or local control in contrast to a traditional DEM which requires centralized or global system control. This allows an ABM to be implemented generatively or bottom-up as compared with top-down modeling in a traditional DEM.
6 The high-level structure of an ABM frequently consists of agents loosely coupled in a network but sometimes a hierarchical structure may be needed. It is pertinent to use the analogy of an ABM being more akin to an informal music group in a jamming session rather than a formal orchestra under a conductor.

2.2 Agent and human roles

When modeling high level processes involved in the lifecycle of a gas turbine, the supervisory control of lower level processes frequently take on identifiable and clearly defined human roles such as those of a manager or a coordinator. An advantage agent-based modeling holds over traditional discrete-event modeling is its ability to emulate human individual and group behavior. Although discrete-event simulations have included expert systems to model human behaviour [27], 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 modeling for investigating social phenomena not only because it matches social activities well but chiefly because of its ease of use [28].

Parunak et al [15] concur that in an agent-based simulation, agents `correspond one-to-one with the individuals being modeled, and their behaviors are analogs of the real behaviors'. This aspect of agent technology has been developed further by Kendall [29, 30] where human roles form the basis of multi-agent system analysis, design, and implementation. It is also a concept integral to the Gaia methodology [31] which starts from the premise of an organization which is made up of roles, their relationships to one another, and patterns of 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 [32] based on the belief-desire-intention technology has the role as its basic unit of abstraction. Examples of such role-centered agent systems can be found in the papers presented at the Winter Simulation Conference [33] in recent years.

3 Background to code metrics

As discussed in Section 1, an objective of this work is to apply metrics to both modelling approaches, this section provides an introduction to the metrics that can were applied. The work by Brooks and Tobias [34] proposed the use of McCabe's Cyclomatic (MCI) as a measure of a simulation model's level of implementation detail and complexity more than 10 years ago, it has not been adopted as a metric for model characterization despite the easy availability of software for compiling that metric. A reason for this may be due to the sheer inadequacy of describing a complex model with just a single number thus hiding useful information like the MCI range and distribution for the software modules making up a model. A model can yield the same MCI if it has a few highly complex modules or a high number of simple modules. Moreover, the MCI may not be appropriate as an absolute measure as proposed by Brooks and Tobias but may find its usefulness as a comparative metric.

Some software metrics which can give a broad view of the model software are therefore suggested here. To ensure that the set of metrics possesses a degree of generality, a criterion for metric selection is to exclude those which are highly dependent on the programming paradigm. For example, the depth-of-inheritance metric for object-oriented programming languages has no equivalence in the procedural languages. The suggested metrics are those which are widely-used, well-validated, and are equally valid for both procedural (Extend ModL, in this case) and object-oriented (Java) programming languages;

1 Lines of non-commented code. Such has been its useful longevity that Basili and Hutchens [35] proposed it should be used as the 'null hypothesis', or benchmark, against which all other metrics are to be compared. To allow for differences in the power of expression of the programming languages, this metric was adjusted using programming language data compiled by Jones [36] which rates a language by number of statements per function point.

2 Number of classes and methods for object-oriented languages, or blocks and procedures respectively for procedural languages. This is a more readily accessible alternative size metric to lines of non-commented code and less sensitive to programming language. In an object-oriented language like Java, classes are created or instantiated at runtime and therefore, other than the code to instantiate them, their class codes do not exist at compile time. In Extend, multiple instances of a block, and hence their code, have to be present at the time of model creation. To allow for this difference in programming languages, only unique Extend blocks in a model subsystem were counted.

3 The McCabe Cyclomatic Index [36] (MCI) may be used to calculate the control flow complexity in individual modules in the lower reaches of a program structure. A directed graph may be employed to display the flow of control in a program with the decision points represented by the nodes and the code chunks between the decision points by the edges. The MCI is a measure of the number of linearly independent paths through the flowgraph. For this reason, it may be used as a metric for the basis or minimum number of test cases to test a program fully. In practice, MCI is a count of decision points in a program module where branching occurs and hence provides an indication of how difficult the program module is to understand, modify, and test.

4 Weighted Methods per Class [37] (WMC) is the sum of the complexity of each method or module, measured by its MCI, within a class or a block. The larger the

number of methods and the higher their complexities, the more time and effort are likely to be required for their maintenance.

5 Coupling is the degree of interdependence between software modules. There is no recognized standard measure, and Fenton and Pfleeger [19] list six types of coupling on an ordinal scale ranging from no coupling (most desirable), through data coupling, stamp coupling, control coupling, common coupling, to content coupling (least desirable). The measure of coupling in a model is taken to be the category furthest towards the less desirable end of the scale.

It is also customary to measure model execution time to provide an indication of performance. In addition to that, the scalability of a modeling paradigm can be deduced by its response, in terms of execution time, to an increasing computational load.

4 Gas turbine maintenance - A case study

The scenario in this case study is representative of problems encountered in the maintenance of gas turbines. The case study considers the benefits and drawbacks of agent-based and discrete-event modeling within the context of this case study although the lessons learnt may find wider applicability.

4.1 Methodology

To ensure that the comparison was carried out on models which were not disparate, the initial activity was to build a DEM which would be functionally identical to the pre-existing ABM supplied by Rolls-Royce plc (RR). The DEM, implemented using the Extend modeling tool[1], was validated first by a static code walkthrough against the functional specification. Confirmation of its dynamic correctness was obtained by running it back-to-back with the ABM using a representative and common set of input data and then comparing their outputs. Subsequently, the same set of input data was used for measuring model execution times for a range of engine fleet sizes. Finally, code metrics were extracted from both models to compare their sizes and complexities.

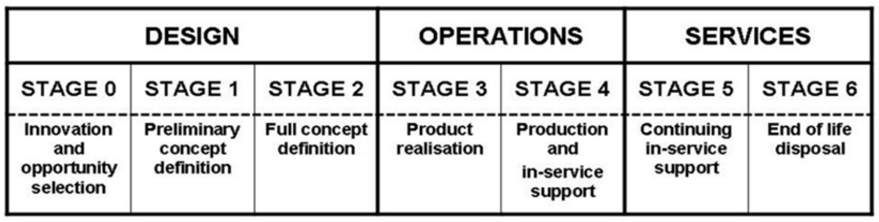
4.2 Model Scenario

The scenario may be described as a problem involving scheduling and logistics for the ongoing maintenance of a fleet of turbofan gas turbines within the lifecycle period extending from just after entry into fleet operation until just before retirement and disposal. This phase of engine life corresponds to Stages 4 and 5 of the RR Derwent Cycle as shown in Figure 2 and they are the stages most closely associated with in-service support. Further, the scenario is concerned specifically with off-wing engine repair in an overhaul base and not with routine on-wing inspection and servicing. It follows that the processes of engine stripping and reassembly, the repair of the individual item, and the movement of new and used engine parts need to be considered in some detail.

The engine consists of approximately 3300 items each of which has to be uniquely identified so that they can be assembled into an engine by grouping them first into components and then into modules. Attributes such as item life, replacement cost, and number of repairs are also attached to each item. When an engine is stripped for overhaul, it is ungrouped into

[1] Extend from Image That Inc, San Jose, Ca

individual items and tracked throughout the overhaul process. This allows the history of each item to be accumulated over a simulation run and the generated data to be processed subsequently.



DESIGN			OPERATIONS		SERVICES	
STAGE 0	STAGE 1	STAGE 2	STAGE 3	STAGE 4	STAGE 5	STAGE 6
Innovation and opportunity selection	Preliminary concept definition	Full concept definition	Product realisation	Production and in-service support	Continuing in-service support	End of life disposal

Figure 2. The Rolls-Royce Derwent Cycle (Diagram reproduced with permission of Rolls-Royce plc)

During this operational phase, in addition to scheduled overhauls or shop visits, an engine can also be subjected to accidental damage and so unscheduled repairs have to be carried out at major maintenance bases which are spread strategically around the world. The ongoing cost of maintenance is influenced by policies related to engine inspection, transportation, repair, scrapping, as well as the stocking of spare parts, the availability of equipment, and the employment of skilled human resources.

In a shop visit, the engine modules which are to undergo maintenance are stripped down to their individual items. They are inspected and assessed according to existing criteria whether they should be repaired, scrapped, or else refitted without further work. Where an item has been assessed as repairable, remedial work is carried out at its designated location while one which has been sentenced to be scrapped may trigger the process to restock if the stock level is considered to be too low for anticipated needs. Once they are available, the original, new, and repaired items are reassembled with the rest of the engine in the overhaul base. Should it be required, a repaired item will be refitted to its engine of origin. Also, depending on overhaul base constraints in force at a specific time, a complete engine module may be swapped.

4.3 The agent based model

This is a time-driven multi-agent system which has been coded fully in Java by one programmer using the open-source Eclipse software development package. The open-source Java Agent Development Framework, JADE [38], provides both the environment in which the agents can reside and communicate as well as a user interface for monitoring and managing the running of the agents. The agents are executed concurrently, each to a thread in a multi-threaded computer process, and their behaviors run without interruption until completion once they have been invoked. All input data necessary for initializing the model is specified in an XML file.

In the developed model, the top-level functions fulfilling all the requirements were set out in a number of process maps that were coded as individual agents while the functions within them have been implemented as agent behaviors. The program code of the model, whose agents are summarized in Table 1, 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 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.

Table 1. The agents used in the agent based model, together with 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.

As is typical of multi-agent systems in the logistics problem domain, there is almost a one-to-one mapping of agents to the main roles or functions in the real world. The agents are loosely coupled and interact with each other by passing messages via the JADE platform. Some of the agent behaviors in this application include the management of an engine fleet, the coordination of the overhaul process, and the shipping and management of parts for repair as well as spare parts. This model implements the operation of a subset of RR's global network of repair and maintenance bases but to mimic the complete network, multiple instances of the agents will have to be created and configured for them to interact as defined by an appropriate network topology.

4.4 Discrete-event modelling of gas turbine maintenance.

The traditional DEM (Figure 3) was implemented using the Extend general purpose modeling tool, a commercial off-the-shelf modeling package which provides a powerful and intuitive visual programming environment. The model is functionally identical to the ABM and was constructed with only the Extend standard library blocks. A Microsoft Excel file supplied the input data and also served as the tool for the post-processing of simulation results. Also, the model was written such that it did not require any further interaction with external software applications after its initialization.

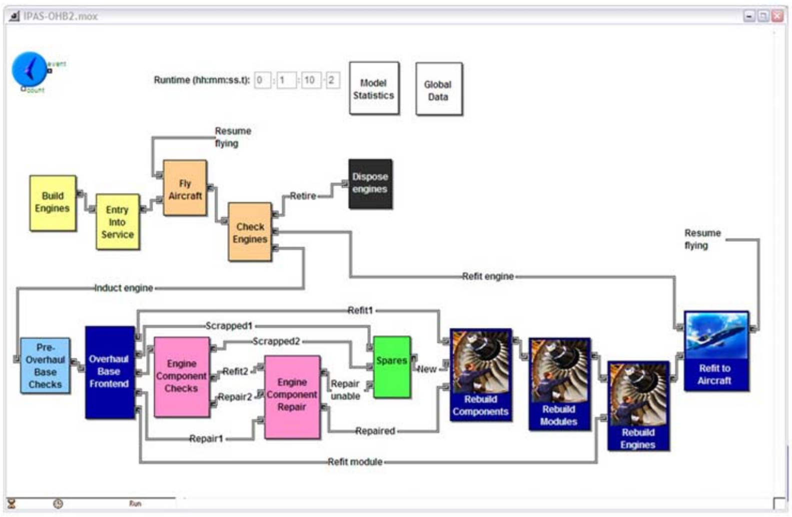


Figure 3. The topmost level of the discrete-event model showing lifecycle activities for an engine fleet from entry into airline service until final disposal.

5 Results

5.1 Execution time

After the DEM had been validated satisfactorily against the ABM, a stand-alone computer was prepared as the common test environment for the two models. A series of performance runs were carried out for each model by varying the size of engine fleet and each fleet size data point was executed five times. The results showed good repeatability as the maximum spread of any data point was less than 0.7%. The results of model execution times are shown in Figure 4.

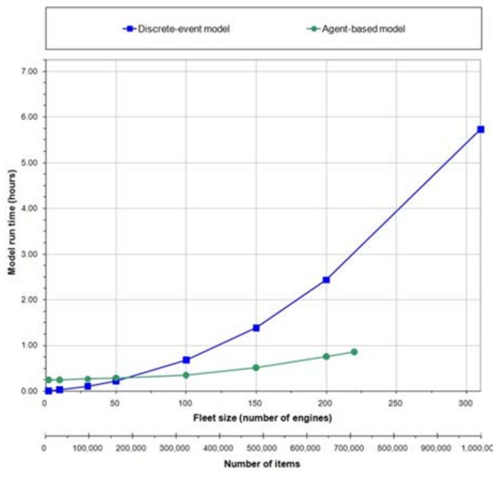


Figure 4 Variation of ABM and DEM execution times with size of simulated engine fleet or number of engine items.

Within this 32-bit Windows test environment, the largest ABM engine fleet size achieved was 220 before the Java runtime environment terminated abnormally with an 'out of memory' error. The maximum DEM engine fleet of 310 was selected because it contained just more than 106 individual engine items and such a quantity was representative of real operational engine fleets. Larger engine fleets could be accommodated by both models if the test environment had more memory and ran a 64-bit Windows operating system.

5.2 Number of lines of non-comment code, methods, and classes, and weighted methods per code

Five subsystems with diverse functions were identified in the models so that their code metrics could be compared against each other. When estimating program size the lines-of-code (LOC) metric has been considered as the benchmark or null hypothesis [35] against which other competing metrics are to be judged. Despite its simplicity, or perhaps because of it, LOC gives a reasonably good estimate of software size.

In compiling the LOC metric for each Java class or Extend block, only executable statements were included. It was refined to exclude code used specifically in the Extend tool for constructing the DEM. For example, code for handling parameter setting, block creation, and block report generation were left out in a block's statement count. Further, the number of statements were factored using data from Jones [36] to allow for the difference in expressive power of Java and ModL. The latter has been described by the developer of the Extend tool as closely resembling the C programming language. Lastly, only unique Extend blocks were included in the count for each of the subsystems. The adjusted results are tabulated in Table 2. The weighted methods per class metric for a subsystem are the quotient of the sum of the complexity (measured by the MCI) of all methods within a subsystem and the number of classes within it. The results are shown in the last column of Table 2. In every instance, the quantities for the DEM are larger than those for the ABM by a significant margin.

Table 2 Code metrics showing multiples of DEM to ABM quantities

Subsystem	Lines of code	Number of methods	Number of classes	Weighted methods per class
Fleet manager	6.01	12.29	9.50	2.22
Overhaul base	3.63	4.34	3.20	3.86
Component repair vendor	9.83	14.07	5.50	6.79
Aftermarket service centre	20.94	24.63	3.67	25.60
Part service centre	11.66	14.75	5.00	9.83

5.2.1 The McCabe Cyclomatic Index

McCabe [39] suggested from empirical evidence that a value greater than ten indicated likely problems with program testability and hence maintainability. Similarly, Grady [40] concluded from a large-scale study of FORTRAN code that an MCI of 15 for a module would be a suitable upper limit.

Although this metric is usually used for procedural languages, it is nevertheless a valid measure for object-oriented languages at the method level. Figure 5 shows that the control flow complexity for both models are well managed, and is markedly so in the ABM. The proportion of modules with a MCI greater than 15 was 0.72% for the ABM and 8.19% for the DEM. Note that the ABM contained tailored code newly developed from scratch while the DEM consisted wholly of mature, commercially available, general purpose library blocks. Typically, software tends to grow more complex with each revision as the bug fixes, special cases, and enhancements need be accommodated.

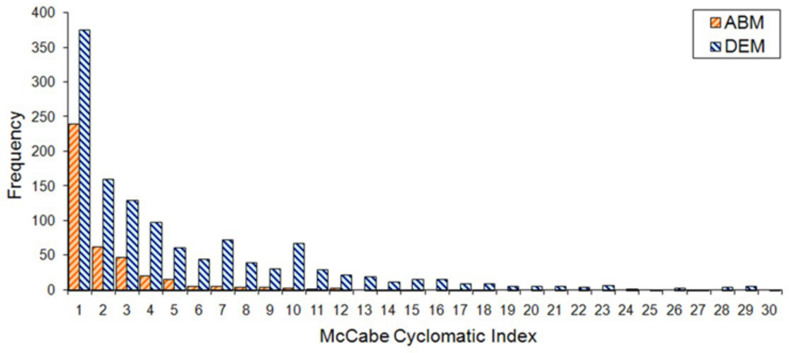


Figure 5 Distribution of McCabe Cyclomatic Index at method and procedure level in the agent-based model and the discrete-event model respectively.

5.2.2 Coupling

Agents in the ABM interact not by invoking each other directly but by message passing via the JADE agent environment. For the additional reason that the message passing is asynchronous, coupling between agents can be judged to fall in between no coupling and data coupling, i.e. the loosest classification short of no coupling at all.

A visual inspection of the DEM code revealed that blocks communicated through message passing and most procedures within a block displayed data coupling. However, the undesirable feature of coupling through global data (coupling Type 5 according to the scale provided by Fenton and Pfleeger [19]) was commonly used for communicating system-wide variables. Therefore, the DEM is biased towards tight coupling and is less desirable than the ABM where modifiability is concerned. It is mitigated by the fact that the DEM was developed within the fixed Extend environment and therefore coupling through global data did not bear directly on the DEM.

6 Discussion of Results

6.1 Model execution times

The model execution times presented in Figure 4 show that the speed advantage of the ABM over the DEM becomes more pronounced as the engine fleet increases. A sizeable contributing factor can be attributed to the fact that the ABM is executed as a process with concurrent threads while the DEM is executed as a single-threaded process. Although there are overheads associated with switching between threads of execution, they are insignificant when compared with the beneficial effects of multithreaded operation.

The DEM is limited by the speed of processing of its central event calendar, the bottleneck through which all events in the model are sequenced and scheduled. While this may be considered a drawback because it is slower in execution, the management of the model through the single central event list guarantees repeatability and this is an important and desirable characteristic of a modeling paradigm. On the other hand, the threads of a multithreaded ABM cannot be guaranteed to complete in a consistently repeatable sequence because they can be interrupted at any time in their execution by other essential processes in a computer. To ensure repeatability, ABMs are sometimes executed as single-thread processes thus obviating the considerable advantage offered by multithreaded execution. In describing the pitfalls of agent-based development, Wooldridge and Jennings [13] cautioned that the agent paradigm should not be used where single-thread execution is necessary.

A 310-engine fleet compares reasonably well with the current RR Trent 800 fleet of more than 500 engines in worldwide operation. Assuming the computing equipment is able to run the DEM normally to completion, a run time of about 12 hours is projected for the current engine fleet. While this may be acceptable in a research environment, it is unlikely to be acceptable to engine designers who may want to estimate the lifecycle costs as they will expect an answer within a few minutes at the most. This problem may be alleviated in part by the seemingly inexorable improvement in hardware processor speed and the introduction of new technologies like multi-core processors into desktop computers. As it is unlikely that a speed increase of more than 100 times can be brought about in the short term from desktop computer hardware alone, a software approach like reducing the DEM's level of details or distributed modeling may bring an acceptable model response time within reach today.

As indicated by the initially constant and shallow gradient of the run time plot, the ABM is relatively insensitive to fleet size. The small increase in execution time in response to a large increase in computational load shows that the ABM is highly scalable. Even though its scalability worsened when the engine fleet size exceeded 100, it took only 0.26 of the time required by the DEM for a fleet size of 220.

During a simulation run for a small engine fleet, the long interval of inactivity between scheduled visits to the overhaul base is punctuated by a brief but very active period experienced during a scheduled overhaul. This behavior may be observed in a time history of processor loading as the agents process the messages they pass to each other. As the size of the engine fleet increased, the periods of low activity shortened while the processor worked to clear the backlog of agent messages. An engine fleet size of 100 marks the point where the periods of low activity are reduced to zero. Since the engine fleet was introduced into the overhaul system en bloc, and there was an upper limit on overhaul base throughput, a long queue of engines waiting to be overhauled ensued and it was not emptied before the engines which were repaired first rejoined the queue for their next scheduled shop visit. Under such conditions, the effect of computer system limitation became more evident. Removing the overhaul base's upper limit resulted in even longer execution times as the ABM was observed to be completely compute bound during the scheduled shop visits thus supporting the notion that the change in gradient had a large element of computer system limitation rather than it being a design artefact of the of the simulation experiment.

6.2 Model code metrics

The advantages offered by an ABM over a DEM are evident in the code metrics for size and complexity. Adjusting those DEM metrics to make them more directly comparable with the corresponding ABM metrics gave results which were invariably lower than their starting values. They are presented in Table 2. Adjusting those DEM metrics to make them more directly comparable with the corresponding ABM metrics gave results which were invariably lower than their starting values. Despite that, the differences which had been measured for the subsystems remained large and therefore allow the metrics to be accepted with some confidence. The inference to be drawn from the metrics is that a model of smaller size and lower complexity would tend towards easier maintenance. As it is the nature of software to undergo modification after initial implementation, this desirable software quality is of considerable economic benefit. Software maintenance costs typically varies between 50% to 65% of overall lifetime costs and enhancement and adaptation activities can make up more than 80% of the maintenance effort [41]. Focusing on software maintenance is therefore justified as it is a highly significant activity within the lifecycle of a model.

The software external property of maintainability may be considered to consist of three sub-characteristics, and they are understandability, modifiability, and testability [42]. A small software program of low complexity will require relatively little mental effort by a programmer to understand it first before amending its code for the desire effect. However, it does not necessarily follow that it will be easy to test especially when the program forms part of a system which has other processes running at the same time. While a DEM with a single thread of execution will produce predictable and repeatable results, an ABM with independent, multiple threads will be more difficult to test and also may not guarantee repeatable results. For the reason that an ABM runs as a process with multiple concurrent threads, their interaction with each other and the possibility of interruption during execution are added complications which make predictability in testing difficult. In this ABM, although JADE makes testing a little easier by enabling an agent behavior to run to completion without interruption, it is nevertheless difficult to test rigorously to achieve a level of confidence comparable with the DEM.

The DEM is larger and more complex than the ABM primarily because of the general purpose nature of the Extend modeling tool used to implement it. First, it is reasonable to infer that the DEM is sub-optimal and this is shown up frequently in that the construction of a single model process function required two or more of the supplied library blocks to implement. Inheritance, in an object-oriented language like Java, is a powerful property which encourages software reuse and enables compact code to be written. This is not available in the procedural ModL language. Furthermore, each library block needs to be self-contained and so some code procedures are repeated in other blocks. Finally, the product has been amended and enhanced over a number of years and additional code and branching logic have increased code complexity. This has been exacerbated by different programmers all with their personal coding idiosyncrasies.

6.3 Modelling large engine fleets

The measured model execution times can provide an indication of the level of detail a model can have by fixing the acceptable time limit for a model to complete a run. If the limit of acceptability for an engine designer is an execution time of no more than 15 minutes, then only a fleet of approximately 50 engines, consisting of a total of about 165,000 individual items, can be accommodated. By trading off between fleet size and level of engine detail, it is possible to simulate the repair operation of a 1000-engine fleet and yet obtain a result within an acceptable time. It is important that models need not be any more detailed than is necessary because excessive accuracy may not justify the additional effort.

The demands of modeling at large scale and fine granularity are not likely to be satisfied by improving computing hardware in isolation but joining them into local or wide area networks is likely to bring about the computing power needed. However, to make full use of networked computing, a model will need to be distributed and executed as a coherent entity among the computing nodes. Clearly identifiable agents in an ABM are inherently loosely coupled and this is a desirable property when implementing distributed models. Even so, the challenges of simulation time synchronization and data distribution across a network will need to be addressed and they can be aided by the framework provided by High Level Architecture [43]. It must be remembered that even though an ABM is easier to implement and maintain than a DEM, “intelligent agents are ninety-nine per cent computer science and one per cent Artificial Intelligence” [44] and therefore a disciplined approach needs to be adhered to.

While a DEM may pose more difficulties for distributed modeling when compared with an ABM, it is nevertheless possible to analyze and partition it appropriately by adhering to a mature, established structured methodology like that propounded by DeMarco [44] and Yourdon and Constantine [45]. While this methodology was conceived in the era of the procedural languages, the underpinning principles it embraces for managing the complexity of a large system, i.e. decomposition, abstraction, and hierarchy or organization, also form part of the foundation of the present day world of object oriented design [46]. By following this software engineering best practice, the resulting subsystems will each have high internal coherence and a low degree of coupling between them. It follows that such a system will have a relatively low volume of data flow between its subsystems thus making them suitable for distributed execution in a network of computing nodes where network latency can become a notable impediment.

7 Conclusions

In designing a gas turbine component using value driven design and concurrent engineering principles, computer modeling is the tool frequently employed to explore a solution space for a reasonably optimized solution within an acceptably short timeframe. Discrete-event modeling has long been used for logistics and scheduling problems but more recently, agent-based modeling has emerged as a credible alternative in that domain.

A quantitative comparison of agent-based and discrete-event modeling has been made and the results show some benefits of agent-based modeling over discrete-event modeling in terms of model software size, complexity, scalability, execution times, understandability, and modifiability. Further, the agent-based paradigm matches human decision-making more closely than the discrete-event approach and therefore enables a problem to be modeled quickly at a high level. They also indicate that it will be worthwhile to extend this line of investigation to other scenarios so as to establish finer objective measures and identify with greater certainty when and where ABM and DEM are to be used. This can be exploited in models of problems where human supervisory roles control the flow of low level processes.

The drawbacks of an ABM are the possibility of unpredictable behavior in testing a multithreaded process, and consequently the possibility of non-repeatable results in simulation runs. Moreover, because of the current dearth of commercial grade modeling tools, constructing an ABM still requires a high level of computer programming skill.

8 Acknowledgments

The research presented in this paper was undertaken as part of the IPAS project (DTI Grant TP/2/IC/6/I/10292). The authors would like to thank both Rolls-Royce plc and the Technology Strategy Board's Collaborative Research and Development programme for their support. Data for the gas turbine, the fleet maintenance variables, and the agent-based model had been kindly made available by the Strategic Research Centre, Rolls-Royce plc, Derby, UK.

References

1. Law, A.M. and Kelton W.D., 2000, Simulation Modeling & Analysis. 2000, McGraw-Hill, Inc, New York.

2. Crowder, R., Bracewell, R., Hughes G., Kerr, M., Knott, D., Moss, M., Clegg, C., Hall, W., Wallace, K., Waterson, P., 2003 “A future vision for the engineering design environment: A future sociotechnical scenario”, Proc International Conference on Engineering Design ICED03.

3. Crowder, R., Hughes, G., Hall, W., 2002, “An agent based approach to finding expertise”, in Fourth International Conference on Practical Aspects of Knowledge Management, D. Karagiannis, Reimer, U., Editor, Springer-Verlang. pp. 179-88.

4. Harrison, A., 2006 “Design for service harmonising product design with a services strategy”, Proc of GT2006, ASME Turbo Expo 2006: Power for Land, Sea and Air2006: Barcelona, Spain.

5. Xu, X. and Wang, Z., 2009, “State of the art: Buisness service and its impact on manufacturing”. Journal of Intelligent Manufacturing. DOI 10.1007/s10845-009-0325-3

6. Wong, S., Crowder, R., Wills, G,. and Shadbolt, N., 2008, Knowledge Transfer: From Maintenance to Engine Design. Journal of Computing and Information Science in Engineering, **8**(1).

7. Acquisition Management System:Technology Management Guidance for the UK MOD Defence Acquisition Community (Annexe A). 2007, Available from http://www.ams.mod.uk/content/docs/techman/ content/trlann/trlanna.pdf.

8. Luck, M., McBurney, P., Shehory, O,. and Willmott, S. “Agent technology roadmap: A roadmap for agent based computing”. Available from: http://www.agentlink.org/roadmap/al3rm.pdf.

9. Belecheanu, R., Munroe, S., Luck, M., Payne, T., Miller, T., McBurney, P., and Pechoucek, M., 2006 “Commercial applications of agents: Lessons, experiences and challenges”, Proc Fifth International Conference on Autonomous Agents and Multiagent Systems,Hakodate, Japan.

10. Taylor, P., and Evans-Greenwood, P., J. Odell, 2005, Agents in the Enterprise, Proc ASWEC2005, Brisbane, Australia.

11. Bonabeau, E., Agent-Based Modelling: Methods and Techniques for Simulating Human Systems. Proceedings of the National Academy of Sciences of the United States of America, 2002. **99**(3): p. 7280-7287.

12. Davidsson, P., 2000, “Multi Agent Based Simulation: Beyond Social Simulation” Proc. Multi-Agent-Based Simulation: Second International Workshop, MABS 2000, pp. 97-107.

13. Wooldridge, M.J. and Jennings N.R., 1999 “Software Engineering with Agents: Pitfalls and Pratfalls”, IEEE Internet Computing, **3**(3), pp. 20-27.

14. Macal, C.M. and North M.J., 2006 “Tutorial on Agent-Based Modelling and Simulation. Part 2: How to Model with Agents”, Proc of the 2006 Winter Simulation Conference, pp. 73-83.

15. Parunak, H.V., Savit, R. and Riolo, R.L., 1998 “Agent-Based Modeling Vs. Equation-Based Modeling: A Case Study and Users' Guide”, Proc. Multi-Agent Systems and Agent-Based Simulation: First International Workshop, MABS '98. pp. 10-25.

16. Brailsford, S.C. and Hilton, N.A., 2000, “A Comparison of Discrete Event Simulation and System Dynamics for Modelling Healthcare Systems”, in Proc of ORAHS 2000, pp. 18-39.

17. Morecroft, J. and Robinson, S., 2006 “Comparing Discrete-Event Simulation and System Dynamics: Modelling a Fishery”, Proc of the 2006 OR Society Simulation Workshop.

18. Jennings, N., 2000, “On agent-based software engineering” Artifcial Intelligence, **117**(2), pp. 277-297.

19. Fenton, N., and Pfleeger, S., 1997, Software Metrics: A Rigorous and Practical Approach, Thomson Computer Press.

20. Forrester, J., 1961, Industrial Dynamics, The MIT Press.

21. Banks, J., Carson, J., and Nelson, B., 1999 Discrete-event System Simulatiom, Prentice Hall, Inc.

22. Robinson, S., 2004, Simulation: The practice of model development and use, Wiley, Chichester, UK.

23. Franklin, S. and Graesser, A., 1996, “Is It an Agent, or Just a Program? A Taxonomy for Autonomous Agents”, Third International Workshop on Agent Theories, Architectures, and Languages, pp. 21-35.

24. Sarjoughian, H.S., B.P. Zeigler, and Hall, S.B., 2001, “A Layered Modeling and Simulation Architecture for Agent-Based System Development”. Proc of the IEEE, **89**(2), pp. 201-213.

25. Wooldridge, M., 2002, An Introduction to Multiagent Systems. John Wiley.

26. Russell, S. and Norvig, P., 2003, Artificial Intelligence: A Modern Approach, Prentice Hall.

27. Vakas, D., Prince, J., Blacksten. H., and Burdick, C., 2001 “Commander Behavior and Course of Action Selection in JWARS”, Proc of the 2001 Winter Simulation Conference, pp. 697-705.

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

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

30. Kendall, E.A., 2000, Agent-Oriented Software Engineering, in First International Workshop, AOSE 2000, Limerick.

31. Wooldridge, M., N.R. Jennings, and D. Kinny, 1999, “A Methodology for Agent-Oriented Analysis and Design”, Proc of the Third International Conference on Autonomous Agents, pp. 69-76.

32. Kinny, D., M. Georgeff, and Rao, A., 1996, “A Methodology and Modelling Technique for Systems of BDI Agents”, Proc of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World. pp. 56-781.

33. WSC, 2011, Winter Simulation Conference, http://wintersim.org/.

34. Brooks, R.J. and A.M. Tobias, 1996, “Choosing the Best Model: Level of Detail, Complexity, and Model Performance”, Mathematical and Computer Modelling, **24**(4): pp. 1-14.

35. Basili, V.R. and D.H. Hutchens, 1983, “An Empirical Study of a Syntactic Complexity Family”, IEEE Transactions on Software Engineering, 1983. **9**(6): pp. 664-672.

36. Jones, C., 2007, Programming Languages Table, Version 8.2.

37. Chidamber, S.R. and C.F. Kemerer, 1991, “Towards a Metrics Suite for Object Oriented Design,” Proc. Object-oriented Programming Systems, Languages, and Applications OOPSLA'91, Phoenix, Arizona, pp. 197 - 211.

38. Bellifemine, F., Caire, G., Greenwood D., 2007, Developing Multi-Agent Systems with JADE. Wiley Series in Agent Technology, John Wiley and Sons, Chichester, UK.

39. McCabe, T.J., 1976, “A Complexity Measure”, IEEE Transactions on Software Engineering, **SE-2**(4), pp. 308-320.

40. Grady, R.B., 1994, “Successfully Applying Software Metrics”, IEEE Computer, **27**(9): pp. 18-25.

41. Krogstie, J., A. Jahr, and D.I. Sjøberg, 2006. “A Longitudinal Study of Development and Maintenance in Norway: Report from the 2003 Investigation”, Information and Software Technology, 2006. **48**(11): pp. 993-1005.

42. Boehm, B., Brown, J., Kaspar, H., Lipow, M.,MacLeod, G., and Merritt, M., 1980, Characteristics of Software Quality, North-Holland Publishing Company, Amsterdam.

43. Kuhl, F., R. Weatherly, and J. Dahmann, 1999, Creating Computer Simulation Systems: An Introduction to the High Level Architecture, Prentice Hall Upper Saddle River.

44. DeMarco, T., 1979, Structured Analysis and System Specification, Yourdon Press Upper Saddle River, NJ.

45. Yourdon, E. and Constantine, L., 1979, Structured Design: Fundamentals of a Discipline of Computer Program and System Design, Prentice-Hall, Englewood Cliffs, NJ.

46. Booch, G., 1994, Object-Oriented Analysis and Design with Applications, Addison-Wesley Reading, MA.