Cranfield University

Sarocha Phumbua

Simulation modelling of service contracts within the context of Product-Service Systems (PSS)

School of Applied Science

PhD Thesis

Cranfield University

School of Applied Science

PhD Thesis

Academic Year 2011-2012

Sarocha Phumbua

Simulation modelling of service contracts within the context of Product-Service Systems (PSS)

Supervisor: **Dr Benny Tjahjono**

January 2012

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Abstract

This thesis deals with the decision support tools for service contracting within the context of Product-Service Systems (PSS). The research contributes to the modelling constructs that can support modellers in developing service contract simulation models in an effective and efficient manner. Overall, the models can assist manufacturers to understand implications of contracting decisions that may either lead to profitable solutions or loss of business opportunities.

PSS is recognised as a survival strategy for many manufacturers to sustain their market competitiveness. It is an emerging manufacturing paradigm that integrates services into products to ensure the required capability or availability of products. This concept is often delivered as long-term service contracts which can be made in separation or together with product acquisition. As the contracts can span over decades, the manufacturers need to absorb the future risks. For this reason, a decision support tool that allows the risks and rewards to be visualised and ultimately support contract design is in urgent need. However, PSS has various characteristics beyond the traditional product-selling businesses and involves potential dynamic behaviour. Existing tools are inadequate to effectively analyse the issues and also to be reused across cases or during the contract delivery phase. For this reason, this thesis intends to provide modelling constructs that enhance effective and efficient development of simulation models for PSS offerings

To accomplish this aim, various simulation modelling techniques have been first explored from the literature and through the practical model developments to identify the backbone of the constructs. The hybrid Discrete-Event Simulation and Agent-Based Simulation has subsequently been selected as the most suitable technique to represent the PSS cases. This technique was applied in four reported cases to generalise the modelling approach. All the developed models have been verified and validated using several methods. The approach was then analysed and refined to enhance efficiency in building models. The refined approach was used to form the modelling constructs. The constructs were validated using three other cases and tested by three other modellers with different simulation background. The results have demonstrated the applicability, practicality, feasibility, and efficiency of the constructs.

The outcomes of this research are the final modelling constructs which provide significant contributions academically and practically. Academically, this research provides a new way of capturing PSS characteristics and dynamic behaviour, and brings together PSS theoretical research, operational planning and decision support tools. Practically, manufacturers can effectively analyse the implication of service contracts and modellers can rapidly develop service contract simulation models.

Acknowledgement

Three years have passed since the day I made a significant decision in my life – the decision to have this PhD as my best friend. Since then, this friend brings me contentment, responsibility, challenges, and concerns. However, I'm grateful for the given opportunity. With this, I would like to express my deepest gratitude to the Thai government for the supports over eight years, and also to the Office of Educational Affairs for taking care of my welfare during this period.

I would like to thank my supervisor – Dr. Benny Tjahjono – for giving invaluable opportunity and experience throughout my PhD. Additionally, I am heartily thankful to Dr. John Ahmet Erkoyuncu, Evandro Leonardo Silva Teixeira, Prikshat Sharma, and Dmitry Borisoglebsky, whose collaborations and feedback enabled me to reach the end. In particular, it would not have been possible without the advice and the encouragements from Dr. Peter Thomas.

Undoubtedly, my years could be filled with smiles and sweet memories thanks to my friends: Joanna Zawadska, Marine Vienet, Agota Berkes, Catia Sousa, Robert Sawko, Grzegorz Lachowski, and Martino Tomizioli. Particularly, my first few days in Cranfield would have been a struggle without Katarzyna Panikowska - who also gave me a perception of responsibility and capability. It was more than a pleasure that I could share some moments with all of you.

And above all, I can have this life journey because of the motivation from the greatest woman in my life. There was no single day in my childhood that I did not see her passion in supporting her students and not a day was passed without her contribution to the unfortunates. It has always been my honour to be raised with this admirable mindset. Thanks to this, I could overcome difficulties and tiredness all these years. Therefore, I would like to take this chance to thank my mum – Sarai Phumbua. My appreciation also goes to my beloved friend – Jutharat Ahchawarattaworn – who made me feel at home outside my country. Your friendship is one of the most precious things I have ever had. And I hope our path will come across again in the future.

Last, my dedication should go to the man who has devoted his every day in the past sixty years in helping others to have a better life. This man turns a ‘country’ to become a ‘homeland’ for millions of people. This man has proven himself his capability, caring, and kindness far beyond his responsibility. It is my pride to follow your footsteps. My deepest appreciation is for our great ‘Father of the Country’.

สโรชา พุ่มบัว

List of publications

Phumbua, S. and Tjahjono, B., 2011. Understanding implication of service contracts in product-service business: A simulation approach. *In: Proceedings of the Operational Research Society Simulation Workshop 2012 (SW12)*, March 27th-28th, Birmingham, UK.

Phumbua, S. and Tjahjono, B., 2011a. Towards Product-Service Systems modelling: a quest for dynamic behaviour and model parameters. *International Journal of Production Research*, DOI: 10.1080/00207543.2010.539279.

Phumbua, S. and Tjahjono, B., 2011b. Simulation Modelling of Availability Contracts. *In: Proceedings of the 44th CIRP International Conference on Manufacturing Systems*, June 1st-3rd, Madison, WI, USA.

Phumbua, S. and Tjahjono, B., 2010. Simulation Modelling of Product-Service Systems: the Missing Link. *In: Proceedings of the 36th International MATADOR Conference*, July 14th-16th, Manchester, UK.

Phumbua, S. and Tjahjono, B., 2011. A hybrid simulation modelling approach for availability contract. *Computers in industry*, Revised version submitted, July 2011.

Phumbua, S. and Tjahjono, B., 2011. Simulation Modelling Constructs to Enable Evaluation of Service Offerings. *Simulation Modelling Practice and Theory,* Under review, October 2011.

Phumbua, S. and Tjahjono, B., 2011. Exploring simulation modelling approaches for service contracts of aero-engine overhaul. *International Journal of Advanced Manufacturing Technology*, Under review, November 2011.

Contents

List of figures

List of tables

Glossary of terms

ABS	Agent-Based Simulation
ATTAC	Availability Transformation: Tornado Aircraft Contract
BPMN	Business Process Modelling Notation
CAD	Computer-Aided Design
CfA	Contracting for Availability
CIRP	The International Academy for Production Engineering
CPN	Coloured Petri Net
CLS	Contractor Logistics Support
DEMATEL	Decision Making Trial and Evaluation Laborat
DES	Discrete-Event Simulation
DoD	Department of Defense
EGT	Exhaust Gas Temperature
EMSA	European Maritime Safety Agency
FBD	Functional Block Diagram
HiCPN	Hierarchical Coloured Petri Net
HiCS	Highly Customised Solution
IDEF0	Integration Definition for Function Modelling
IPPM	Integrated Production Process Model
IPSS	Industrial Product-Service System
IVHM	Integrated Vehicle Health Management
LCH	Lost Customer Hours
LCS	Life Cycle Simulator
LLP	Life-Limited Part

MEPSS	Methodology Development and Evaluation of PSS
MOD	Ministry of Defence
MRO	Maintenance Repair and Overhaul
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
OEM	Original Equipment Manufacturer
OML	Optimization Modelling Language
OO	Object-oriented modelling
PBC	Performance-Based Contract
PBL	Performance-Based Logistic
PDD	Property-Driven Development/Design
PE	Polyethylene
PP	Polypropylene
PSS	Product-Service System
PVC	Poly Vinyl Chloride
QFD	Quality Function Deployment
RAF	Royal Air Force
RSP	Receiver State Parameters
SC	Scenario
SD	System Dynamics
SLA	Service Level Agreement
SOPMF	Solution Oriented Partnership Methodology Framework
t-PSS	technical Product-Service System
TAT	Turnaround time
TRIZ	Theory of Inventive Problem Solving
UML	Unified Modelling Language

XML	Extensible Markup Language

1 Introduction

This chapter introduces the PhD research on "Simulation modelling of service contracts within the context of Product-Service Systems (PSS)". Described in this chapter are overviews of research context (Section 1.1), research aim and objectives (Section 1.2), scope of this research (Section 1.3), research methodology (Section 1.4), and the thesis outline (Section 1.5).

1.1 Overview of research context

This section summarises the key background knowledge and briefly describes the contents in Chapter 2. It highlights the need of this research which led to the development of the research programme in Chapter 3.

Embracing service offerings into products is commonly known as an alternative for Original Equipment Manufacturers (OEMs) to increase competitiveness. The new integrated offering is referred frequently in literature as "Product-Service System (PSS)", and commonly delivered in terms of service contracts. Numerous amount of research highlights the benefits of the transformation, as summarised by Baines et al. (2007). Among those, the driving forces towards the transition from business viewpoint are in terms of the advent of the global market (Section 2.1). Nowadays, competing on cost is no longer an option for western manufacturers. The OEMs need to seek a way to defend themselves from the lower cost economies. Similarly, sustaining leadership in innovation and technology becomes harder to achieve. On top of that, product sales are pretty much sensitive to market conditions, and customers have higher expectations than in the past. Faced with these threats, customising the offers through services is an attractive solution for the OEMs.

Although the benefits and the driving forces are paramount, implementing the concept requires a huge effort in changing the (established) cultural mindset, operation structure, and infrastructure (Baines et al., 2009a). Furthermore, service contracts often span over a long period of time. Estimating the condition in the far future encompasses uncertainties, for example, equipment usage rate, repair turnaround time, obsolescence rate, which entails a considerable risk to the OEMs (Erkoyuncu, 2011). Therefore, implementing PSS is an expensive decision and becomes the major challenge for the OEM. For these reasons, a decision support tool is in an urgent need to enable evaluation on the risks and rewards prior to making such an offer. Moreover, such a tool should effectively address the decision parameters specified in the contracts, for example, the availability performance required by customers, the resources the OEM should invest to sustain the contract, the payments and penalty structure (Section 2.2).

A number of modelling techniques have been proposed in the literature to enhance understanding of product-service offerings (Section 2.4), categorised based on the

focus of the model as user-related, system-related, and product-service-related (Phumbua and Tjahjono, 2011a). Considering the whole demand-supply network, the first group requires insight knowledge of customers, whereas the second group focuses on the OEM side. The last group is evolved from the traditional product design, thus, this group perceives the system as a combination of product-related elements and service-related elements. In comparison, the abstraction is increasing from the user-focus to the product-service focus, and the system focus, respectively. Regardless of the focus, the majority of these modelling techniques do not incorporate the capability to handle the dynamic behaviour in PSS and to be reused in the contract delivery phase (Phumbua and Tjahjono, 2011a). The exception is the simulation modelling techniques, which allows what-if analysis to be carried out easily. Simulation is defined as experimentation of a representation of dynamic operation system (Robinson, 2004). Generally, simulation aims to provide better understanding or to improve the system.

Within the context of PSS, application of simulation is still limited (Phumbua and Tjahjono, 2010). In particular, the general approach that enables efficient developments of simulation model is the major shortcoming in the PSS research area (Section 2.5). This shortcoming indicated the direction for this research.

In summary, there exists driving forces for the OEMs towards service contracting. Nonetheless, the challenges to the transition are significant. To better understand the challenges and offer appropriate contracts, a number of models or modelling techniques have been proposed in the literature. However, very few techniques enable changes to contract negotiation to be effectively captured. An exception is simulation, yet, the general simulation approach that supports effective and efficient model developments has been missing in the literature. This led to the research aim and objectives described in the next section.

1.2 Overview of research aim and objectives

The previous section provided the background towards the development of this research. This section presents an overview of the research aim and objectives explained in detail in Chapter 3. The aim of this research is

to propose the modelling constructs that enhance effective and efficient development of service contract simulation models for PSS offering.

To realise this aim, a number of objectives were devised:

1. To identify a simulation technique that can potentially capture PSS characteristics and dynamic behaviour

2. To determine an appropriate modelling approach that enables effective and efficient development of the models

3. To form primary modelling constructs based on the approach

4. To evaluate and refine the primary constructs

5. To present the final constructs

In this thesis, the term *construct* covers modelling elements and modelling methods. Examples of modelling elements include 'source', 'sink', 'state', 'transition' etc., whilst modelling methods relate to technical programming commands used to describe, for example, object creation, condition-based triggers etc. A *technique* indicates the conceptual backbone of simulation models, for example, System Dynamics (SD), Discrete-Event Simulation (DES), Agent-Based Simulation (ABS). A modelling *approach* implies how a particular technique will be used to model a system in more details. To illustrate, a book is written about simulation methodology which explains a systematic process in conducting simulation studies. In a simulation study, a student builds a model using the ABS technique to represent an enterprise which comprises several business units. His approach is to capture this situation into three levels – **business unit** level which is embedded in the **enterprise** level underneath **market** level. To enable communication between units and enterprise, he sets up a message passing method using a 'transition' with a defined message receiving command.

1.3 Overview of scope of this research

'Product-Service System' is defined as an integrated product and service offering that delivers 'value in use' (Baines et al., 2007). Resulting from the concept is a wide range of business models. In order to set the context, an attempt was made through investigating various types and sub-categories of PSS, including related research areas. The search mainly covered product-oriented PSS, use-oriented PSS, result-oriented PSS, industrial PSS, technical PSS, product-centric servitisation, availability contract, performance-based logistic, Contractor Logistics Support (CLS), and performance-based contract. Eventually, this research adopts product-centric servitisation defined by Baines et al. (2009a) as the offering in which the product itself is central to the provision of services. Therefore, the context of this research was set as product-centric PSS.

1.4 Overview of research methodology

Five stages of research methodology were devised to achieve the five objectives of this research.

Stage I corresponds with the first objective to identify the technique that can potentially capture PSS characteristics and dynamic behaviour. In this stage, the three selected simulation techniques identified from the literature review from Chapter 2 were further evaluated in terms of capability and drawbacks in modelling service contracts. The evaluation was based on both literature and actual model developments. Therefore, research methods are associated with literature review, steps in model development, verification and validation methods. Besides, four business cases were collated from PSS literature and used as the data for developing the models. The outcome of this chapter is the technique that underlines the core structure of all the models developed from the constructs.

In Stage II, the technique obtained in Stage I was applied across cases to develop the modelling approach, which responds to the second objective of this. Thus, the major research methods also involve case studies, modelling steps, and verification and validation methods. Four cases were chosen under the product-centric PSS context. The resulting approach in this stage details the modelling requirements and methods inside the constructs.

Stage III relates to the third objective of this research, which aims to form the primary constructs based on the outcomes from Stage II. The methods applied in this stage are associated with technical programming and presentation of the constructs.

Stage IV deals with the evaluation and refinements of the constructs, which corresponds to the fourth objective of this research. The evaluation at this stage is in terms of efficiency and effectiveness in enabling service contract model developments. Efficiency refers to how quick a model can be developed using the constructs compared to without using them. Effectiveness was considered as the applicability to model existing cases, the practicality to aid decision making and the feasibility to develop a model from the constructs. Two types of validation were conducted; case study validation and user validation. Three case studies were conducted under the product-centric PSS context via interviews, using a case study protocol. This validation focussed on evaluating the applicability and the practicality, and enabled primary evaluation of the efficiency and the feasibility. The other validation was conducted with three users who have been involved in PSS research and have different levels of simulation background. This validation aimed to further evaluate the efficiency and the feasibility of the constructs. The analysis based on observations, feedback, and questionnaires, was used to refine the primary constructs.

Stage V deals with the final objective of this research, which aims to present the final constructs. Overarching, the purpose of the presentation is to instruct users how to use the constructs to build a model. Therefore, the presentation includes the overview of the constructs, a description of how to use them, the constructs, and an example of the implementation on a software package.

1.5 Thesis outline

Chapter 1 – Introduction – provides an overview of the research context, research aim and objectives, methodology, the scope, and the thesis structure.

Chapter 2 – Literature review – addresses challenges to manufacturers in the product-selling business which suggest the reasons why making product-service contracts is an attractive strategy. Following these drivers are existing business cases, including detailed decision parameters in the contracts. The findings from this part built up an understanding of offers and formulated the modelling scenarios in Chapter 4. The shift in business characteristics from being a pure manufacturer to a PSS provider and the additional dynamic behaviour are presented to build up the justifications as to why a modelling tool is needed. Last, a review of existing modelling approaches within a PSS context is provided along with the shortcomings in the field, which drives the direction of this research.

Chapter 3 – Research programme – clarifies the developed aim, objectives and scope of this research. Different ways to conduct the research are discussed in this chapter, which formulated the applied research methodology into stages to realise each objective.

Chapter 4 – Investigation of simulation modelling techniques – relates to the review of potential modelling techniques primarily evaluated from Chapter 2, from literature and actual model developments. Finally, analysis of the technique's capability against strengths and weaknesses of PSS modelling literature is provided. The detailed models are illustrated in the Appendices (B, C, D, and E).

Chapter 5 – Development of modelling approach – is concerned with application of the final chosen technique from Chapter 4 to various cases. The modelling approach is stated in this chapter, whilst the model code is documented in the Appendices (E, F, G, and H). This chapter summarises and discusses the adopted modelling approaches in all the cases. Finally, the selected approach that underlines the details inside the constructs is explained.

Chapter 6 – Formation of the modelling constructs – presents the modelling elements and the detailed modelling methods, along with a three-stage-roadmap which describes how to use these constructs to develop service contract simulation models.

The analysis of the constructs' capability against strengths and weaknesses of PSS modelling literature is also provided.

Chapter 7 – Evaluation of the constructs – describes the evaluation of the constructs though multiple case studies, and various modellers. The analysis of the results obtained from the cases and the modellers are presented in this chapter. The amendments to the constructs based on this analysis are described.

Chapter 8 – Presentation of the final constructs – presents the final constructs which contain modelling elements and methods, along with an instruction to use these constructs. Example of the constructs implementation in a software package is demonstrated in Appendix L.

Chapter 9 – Discussion and evaluation – discusses the research findings during the execution of this research, the strengths and limitations of this research, and the emergent literature in the area.

Chapter 10 – Conclusions – summarises achievements in correspondence with the research objectives, identifies direction for future research, and concludes this thesis.

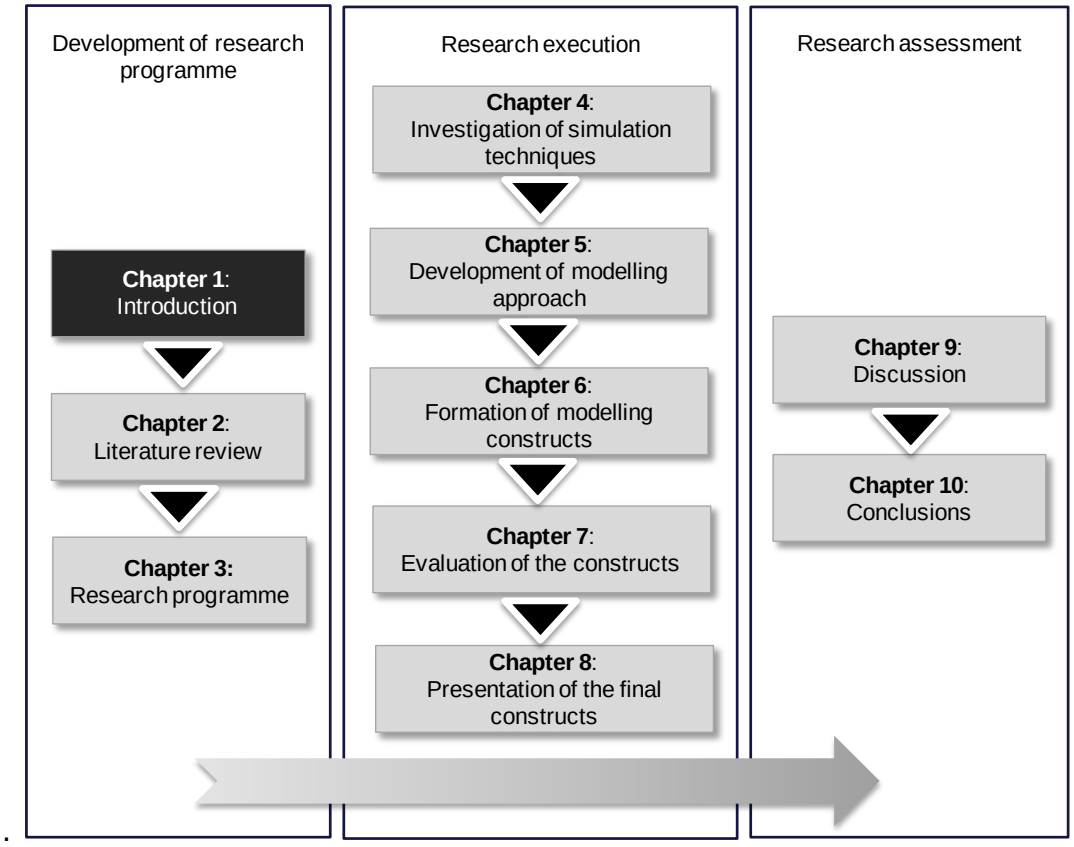


Figure 1-1: Thesis outline

2 Literature review

In Chapter 1, the significance of this research topic was introduced. This chapter describes in detail the development towards the research aim and objectives. Section 2.1 addresses the drivers towards PSS implementation for manufacturers. This section describes why traditional product-selling business may not be a sustainable solution. Towards PSS adoption, the decision on what to offer involves a number of issues. Section 2.2 highlights these issues through various PSS offering examples. Section 2.3 identifies PSS characteristics and dynamic behaviour during the service delivery phase, which extends beyond traditional business. These two sections depict the need for an approach that can capture the dynamic behaviour and enable alternative offerings to be investigated prior to offering a contract. Following this need, existing techniques and tools were investigated and are presented in Section 2.4. Finally, the research gap in existed techniques and tools is discussed in Section 2.5 as well as the summary of this chapter.

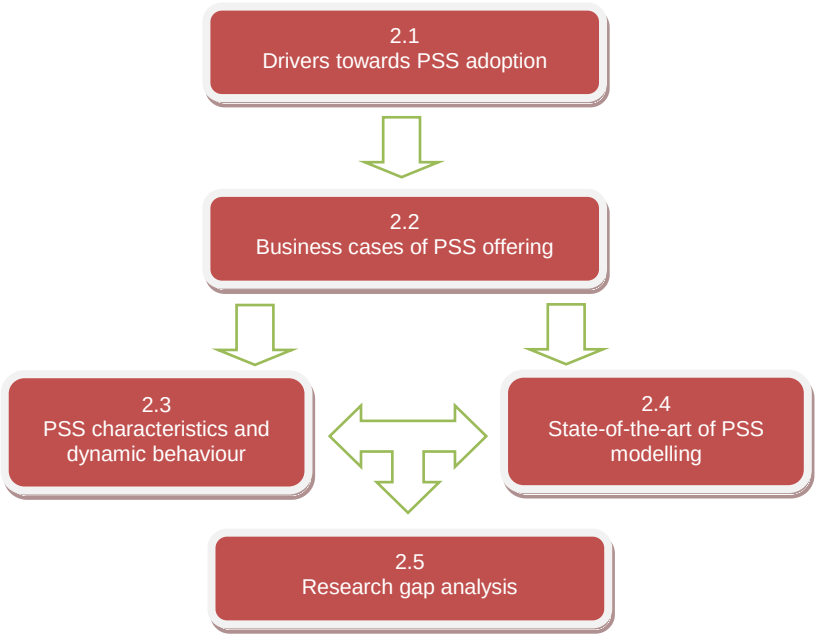


Figure 2-1: Chapter 2 outline

2.1 Drivers towards PSS adoption

The purpose of this section is to highlight the reasons why the OEMs may seek a PSS solution in spite of the possible big investment.

The benefits and opportunities which PSS and servitisation can create have been widely discussed in literature (e.g. Roy and Cheruvu, 2009; Neely, 2008; Baines et al., 2007; Bey and McAloone, 2006; Mont, 2002). Along this line, the major driving forces

arise from a business point of view. Therefore, this thesis mainly highlights those which impact on the survival of the business.

To defend from lower cost economies: An impact from global competition causes a threat to the OEM, as supported by Chandraprakaikul (2008). The study exposed that the drastic increase in pressure to the UK manufacturers come from an advent of low cost economies such as China and India in the global market. This means competing on cost is no longer an appropriate strategy for the OEMs. Whilst the manufacturers are seeking for a solution, PSS appears as a survival alternative for the companies (Roy and Cheruvu, 2009). This hypothesis is supported by Baines et al. (2009b) as "The integrated product-service offerings are distinctive, long-lived, and easier to defend from competition based in lower cost economies".

To sustain competitiveness: As well as cost, competing on product innovation and technological superiority is no longer sustainable, which is often not the case for competing on services (Baines et al., 2009b). As service is intangible, service delivery is heavily dependent on skill, more flexible, hence harder to imitate on one hand (Oliva and Kallenberg, 2003). On the other hand, servicing skill can be continuously improved throughout the contracts. Accordingly, competing on services seems to be more sustainable in relation to other types of competitiveness.

To survive in economic crisis: Traditionally, business survival is tied up with product sales, which heavily depend on market and economic conditions. On the contrary, PSS contracts ensure continuity of income to the OEM as customers are obliged to pay regularly. It can attract more customers since the payments are split into several instalments in which they can afford more than buying the whole product. Therefore, the impact from an economic crisis is less in PSS business than in the product selling-only business. The potential in coping with a poor market condition of servitisation was mentioned in Mallaret (2006), and Oliva and Kallenberg (2003). Also via simulation modelling, Buxton et al. (2006) demonstrated that product-service business models could generate a better robust revenue system to handle the demand crash than a traditional business could.

To handle demand from customer: Satisfying customer need is a critical success factor for a business. However, customers, especially industrial customers, become more demanding on services (Oliva and Kallenberg, 2003), and lower prices (Chandraprakaikul, 2008). The reasons may be caused by a wider range of choices available in the market, or by the pressure the customers received from downstream. For instance, the customers may also be faced with the budget constraints as in case of the UK Ministry of Defence (MOD) (Erkoyuncu, 2011), or with the increase in in-service costs as addressed by Berkowitz et al. (2005) and IFS (IFS, 2010). The PSS offerings can avoid unnecessary expenses, for example, $1.4 billion over 30 years in the F/A-18 contract (Gansler and Lucyshyn, 2006), $53.4 million in the case of the F404 engine PBL agreement (AIA, 2011), and £510 million over 10 years on an aircraft support contract

(IFS, 2010). In commercial contracts, £10.4 million was saved over four years on the office printing in British Telecom (Xerox, 2010a).

In all, this section highlights that PSS is not only an opportunity for the manufacturers, but also a survival strategy in today's market. The next section identifies issues and decision parameters prior to offering PSS to customers.

2.2 Business cases of PSS offering

Though PSS is an attractive solution, inexperience in pricing is a major barrier towards transition (Baines et al., 2007). The intention of this section is 1) to provide background on PSS offerings and 2) to present the major decision parameters regarding the offerings via examples of existing contracts. Therefore, an overview of successful cases reported in literature is presented in Section 2.2.1, and offering decision parameters from actual contracts are covered in Section 2.2.2. The findings described in this section were used as a basis for research gap analysis presented in Section 2.5 as well as research context and modelling scenario developments presented in Chapter 3 and 4 respectively.

2.2.1 Overview of the offers

A number of business cases have been reported in implementing PSS in industries. Examples include launderette, car sharing, carpet leasing, train leasing, document management solution, and aircraft engine leasing (Khumboon et al., 2009; Baines et al., 2009c; Mont, 2000; Harding and Watts, 2000), which were developed from contract arrangements such as rental contract, pay-per-use, lease and take back (Lindahl et al., 2009). Among these cases, five industries have been continuously offering PSS solutions.

Aircraft: In commercial aircraft, the OEM sells both asset and services to support the asset in use and the ownership is shifted to customers or a 3rd party such as financial organisation (Baines et al., 2009c). However, the OEM provides remote monitoring to ensure delivery of functionality, including efficient maintenance scheduling, effective management of repairs and spares, and/or part re-manufacturing. The asset data are recorded and used to trigger upgrades. Even so, low-level maintenance and consumable management, are mostly carried out by customers (sometimes also with their business partners). Another example of commercial service contracts in the aero-industry was made between Pratt & Whitney (P&W) and Jet2. In this case, P&W supplied Life-Limited Parts (LLPs) to support the CFM56-3 engine (Pratt & Whitney, 2009). In military context, PSS solutions are generally in the form of Performance-Based Logistic (PBL), frequently adopted by US Department of Defense (DoD). PBL was categorised by Aviation Week into four classes; components (which usually aim for consistency and timely delivery), major subsystems (whereby availability is crucial),

entire aircraft (whereby availability is the goal), and mission capability (in which readiness and continuous capability improvement become the focus). An example of PBL includes the C-17, a long-range cargo/transport aircraft. The aircraft is capable of performing airlift/drop missions and providing strategic delivery of troops and cargo to main operating bases. It can also be configured for aero medical evacuation (Mahon, 2007). Other examples of military service contracts have been made between P&W and Italian Air Force (Pratt & Whitney, 2011), Rolls-Royce and UK Royal Air Force (RAF)(Rolls-Royce, 2010), and BAE and UK RAF (BAE Systems, 2006).

Train: In the railway industry, the operator's risk and reward can be confined by the department of transport, as reported by Macbeth and De Opacua (2009) that the contract between the two parties is renewed every 7-10 years. Previously, the operators were often responsible for both light and heavy maintenance. However, the operators can have three options for heavy maintenance under PSS business model; 'Dry' in which ROSCO (i.e. a financial organisation such as HSBC) has no responsibility, 'Wet' which is the opposite, and 'Soggy' which is a combination of both. In the context of an underground train, a service contract was made by ALSTOM, which aimed to provide an agreed level of availability to London Underground Ltd (LUL). LUL took the trains every morning and returned to ALSTOM at night after operation for maintenance activities (Harding and Watts, 2000).

Photocopier: Typical example of service offers in integration with photocopiers can be illustrated by Xerox's document management solution. As highlighted by Anderson and Tukker (2005), the focus of Xerox has moved to the entire commercial documentation process in which the asset can be leased or sold under multi-year contract on functional guarantee and the payment is fixed per copy. Five offering packages are currently available from Xerox; Document Transaction Processing Services, Enterprise Marketing Services, Enterprise Print Services, and Product Lifecycle Communication Services (Xerox, 2010b). The service activities involve a mail handling system, transforming document format, DocuCare and print-on-demand, and managing document infrastructure (equipment, maintenances, suppliers, help desk support). In other words, the company provides flexible contract (in 6-sigma project format) which aims to reduce cost to clients. Xerox provides proactive asset management using a software suite to discover, track, control, configure, and report on the multi-vendor environment of clients in real time, which enables alerts. Indicated by one of the contracts, preventive strategy is valid only to high-use equipment in some product families (Xerox, 2010c). Consumable and parts can be stocked and supplied on-site. Under a rental/leasing contract, a Thai company was reported to buy second hand photocopiers from the second hand market after a three years in-service period, and recondition the assets (Khumboon et al., 2009). The reconditions were associated with rebuild, replacement, and cleaning of components; for example, disassembly and repaint the cover. The reconditioned photocopiers were placed for rent to 5000 customers, in which the maintenance activities were in the provider's responsibility. A similar leasing case was also reported by Kuo (2011).

Carpet: A carpet's life cycle can be divided into four phases; manufacturing, transportation and installation, use phase, and disposal/recycling. A carpet is typically constituted of a layer of face fibre made of Nylon, PE, or PPL, and a backing made of a sandwich of PPL and latex (or PVC) (Lu, 2010). For this reason, carpet manufacture causes chemical emissions, depletion of petroleum and other natural resources, indoor air quality concerns, and disposal at landfills/recycling process. Therefore, the recycling process can become the focus of service contracts in this industry. Regarding maintenance, a carpet needs to be cleaned periodically and replaced within a specific usage period, independent from the wear condition. A typical carpet leasing programmed is offered by Interface Limited, named as Evergreen, and paid monthly. The company's current measures mostly involve environmental and social issues. For instance, QUEST focuses on waste reduction, EcoMetric incorporates environmental outcomes, and SocioMetric relates to social performance (Stubbs and Cocklin, 2008). To achieve these targets, Interface has been redesigning products from renewable resources to eliminate waste and emissions. For instance, the company is replacing bio-based fibre from oil-based fibres. By doing so, worn (or damaged) carpet tiles could be easily replaced and reused rather than being disposed in landfills. This can reduce waste during installation and maintenance processes. Besides this strategy, the company has been trying to recycle nylon and produce only fabrics made from recycled PE and wood. Interface currently works with Universal (their supplier) to utilise post-consumed nylon and PVC backing (Nelson, 2009). Besides Interface, Dupont provides three choices to customers; 1) customers simply buy the particular products, 2) the company selects the right products based on customer requirements, and 3) the company leases the products to customers on 2-5 years contracts (Mont, 2000). Other services include installation, cleaning, consultation and providing recommendation for cleaning products.

Washing machine: A washing machine life cycle can be divided into 6 stages; fabrication (mostly done by suppliers), system assembly, product delivery, in-service use, in-service maintenance, and recycle/disposal. The fabrication includes raw materials extraction (such as iron, copper, and oil) from natural reservoir. These raw materials are used to manufacture components of a washing machine which comprise electrical components (e.g. motor, transmission), mechanical components (e.g. tub and plumbing), electronic assembly (e.g. control panel, connector) and housing. After being used, a washing machine and obsolete parts are disposed by landfill, incineration, and recycling. Metals from equipment frame, plastic part, and packaging boxes can be recycled, in fact, sixty percent of weight material is designed to be recycled or reusable with no design changes required (Graedel, 1997). In general, 54% of 5-year-old machines are repaired at least once, and motors are removed from the end-of-life machines to be re-manufactured as spares (Simon et al., 2000). An existing PSS offer from Electrolux (ELS) has provided system solutions, driven by customer preferences, in which the necessary equipment for a launderette are supplied (Mont, 2000). Such solutions include installation, training, consultation (suggesting layout of equipment location and advising on ironing and delivery), maintenance and repair,

financial schemes, and upgrades with the latest machine. Other than this PSS solution, customers can opt to own equipment (with 1 year guarantee) or make Service Level Agreement (SLA) in which full spares are included or lease the products, as reported by Kowalkowski (2006) that there are three levels of SLA agreements; visit one or few times a year, service plus spares, and full service. Kowalkowski also revealed that ELS has technology to obtain usage data, called CMIS. The company can monitor processes, error code and the time it starts to be out of operation promptly without site visit.

To sum up, this section provides a background understanding on PSS offerings in the five selected domains which expose different offering formats. In the commercial aircraft context, contracts can be offered at fleet level (e.g. P&W fleet management contracts), and individual level (e.g. power-by-the-hour from Rolls-Royce). In terms of military aircrafts, contracts are often made on fleet basis (e.g. BAE Systems' ATTAC contracts). Both contexts offer services around maintenance activities. This is similar in the context of train contracts. Differently, the strategic service in printing and washing sectors is not on maintenance activity, but re-manufacturing. The launderette option and document management solution share a common principle in providing capability on the fleet-basis and leasing on an individual basis. With regards to the carpet sector, the choice of raw materials and production process naturally encourages recycling as a central strategy. Understanding this basic, the next part looks into decision parameters.

2.2.2 Contract parameters

The literature search focused on the actual contracts or a case study report. However, as the information is in a very detailed level, very few results have been found.

Aircraft: ATTAC (Availability Transformation: Tornado Aircraft Contract) is a contractual agreement between BAE Systems and the UK's Ministry of Defence, reported to be worth in the region of £1.5bn (BAE Systems, 2010). ATTAC guarantees availability of Tornado aircrafts for the Royal Air Force (RAF) by providing advanced maintenance of the aircraft fleet throughout their service life. As with other contracting for availability, ATTAC is an outcome-based contract in which payments are linked to the performance of the availability. Key features of the ATTAC were reported to include up to 140 flight hours per month and over 1000 demands per day of asset and inventory, with two key contracting modes; availability or turnaround time. Following these key features, the performance indicators include available flying hours, spare parts and technical support. The service provides the customer with flexibility in flying hours between 70% and 110%.

Train: In the case of LUL and Alstom (Harding and Watts, 2000), LUL requires a complete package covering design, manufacture, maintenance, and cleaning of trains and associated trackside equipment. Key performance parameters involve a guaranteed number of trains in peak service, reliability in service and the management

of depot stocks, which are linked to the on-train run time. In case of not achieving the targets, LUL can terminate the contract before the actual agreed term of 20 year. The contract encompasses four elements; manufacturing, depot supports, communication, and train supports, in which Alstom disseminates responsibility to different partners. Trains are built by Alstom Transport Limited, having GPT to take care of communication system such as train-to-train CCTV, and AMEC to look after the depot supported work. Alstom Transport Service Limited carries out maintenance services daily after the trains are returned from LUL at the end of operations.

Photocopier: Based on the Xerox DocuCare contract (Xerox, 2010c), issues stated in the contract involves product family, contracted machines, maximum distance between machines, location, contract hours (e.g. between 9 am - 5 pm), and monthly availability hours. The measures are in the form of achievable percentages of service responses within 1 hour period, equipment uptime, and technical response time. All time dimensions are calculated from the point of receiving notifications from the customers. Uptime is a comparison between the monthly contracted period and downtime. The downtime is considered from the point Xerox receives notification until the machine is ready for operation. Xerox keeps reporting the uptime statistic on a monthly basis.

Oil Vessel: A vessel service contract was formulated by the European Maritime Safety Agency (EMSA) for the standby oil spill recovery service (EMSA, 2006). The contractor was obligated to provide vessels which had a stored capacity defined by EMSA, on the point of request. The service also covered trained crews and equipment. However, the vessels could be used by the contractor for other businesses when EMSA did not call for services. Otherwise, EMSA could request the contractor for mobilisation if an oil spill is foreseen, and for demonstrating up to 10 days or 3 days per vessel. The contract was designed for 3 years, in which the payment was made quarterly. The contractor was required to schedule annual repairs in the beginning of the year, for the operation in summer. Once this happens, the process must finish in 10 days or the payment would be decremented in the proportion of additional days. In case a period of 20 days was exceeded, a replacement of vessel was needed. Contract termination by EMSA was possible, also at the cost of the contractor.

To summarise, this section described the major decision parameters that have been reported in PSS contracts. The requirements from customers can cover fleet availability, maximum unavailable duration, availability within an agreed period, reliability, time to respond, and capability. The penalty is ranged from contract termination at the cost of the PSS provider to no serious penalty required. Therefore, a vast range of PSS contracts exists under a similar format of decision parameters. This suggests a need for decision support tools capable of evaluating the alternatives. The next section addresses characteristics and dynamic behaviour which distinguish PSS from traditional businesses. The outcomes of these two sections underlie major capability requirements from potential decision support tools.

2.3 PSS characteristics and dynamic behaviour

This section addresses distinctive PSS characteristics from traditional business (Section 2.3.1), and the resulting additional dynamic behaviour (Section 2.3.2).

2.3.1 PSS characteristics

PSS implementation necessitates a number of investments and changes from traditional business, as stated by Baines et al. (2009a), Manzini and Vezzoli (2003), and Mont (2002). These can be associated to business and strategy, operation, and network (Phumbua and Tjahjono, 2011a). This thesis captures only those which reflect on the modelling implications.

On the business and strategy level, PSS customers generally care for asset availability and performance rather than the features of the asset. It is possible to reuse the same product for different contracts, thus, products in PSS are encouraged and often designed to be reusable or recyclable. In replacement of production cost, life cycle cost is the major issue in PSS business, which is, in many cases, influenced by product life. Service efficiency is the area of significant contribution in system improvement (Mont, 2002), unlike production efficiency as in traditional business, which is enabled by human capital.

From an operational perspective, it is crucial in PSS that staff are knowledgeable in the products, and skilful in delivering services and managing ongoing relationships with customers, as highlighted by Baines et al. (2009a). This leads to a more decentralised decision making. Capacity utilisation is hard to achieve, as staff responsiveness to unexpected events is more crucial to obtain the availability/performance defined in the contract. Therefore, resources are mostly kept at maximum level. Several companies manage existing technology to obtain efficiency and effectiveness, as reported by Manzini and Vezzoli (2003). Especially, as information flow becomes much more critical than in the traditional business (Mont, 2002), monitoring technology is often used as a key enabling strategy for responsiveness optimisation, as evident in aircraft, photocopy, and washing machine industries.

From a network viewpoint, relationships between stakeholders during the in-service phase are mostly governed by asset availability and performances (Baines et al., 2009a). Frequent interactions and strong cooperation in the network are commonly known as a major characteristic of PSS. These interactions induce a number of dynamic behaviours that are stated in the next section.

2.3.2 Dynamic behaviour

Dynamic behaviour is referred to as disturbances that may affect system performance, both manageable and unmanageable (Phumbua and Tjahjono, 2011a).

In the broad sense, the dynamic behaviour involves external events such as **natural disaster**, **recession**, **regulation changes**, and **market** and **price changes**. The impact from these events can be varied. A natural disaster (such as the volcano ash) can delay part/equipment supplied which may prolong the unavailability of contracted assets. A recession can impact the main business of stakeholders and can cause the parties to request for contract modification. A regulation change (such as an advent of new standards) may require product modification. Finally, market and price changes affect directly on the profits. As PSS expands an OEM's responsibility to cover the in-service and after-use phases, they are likely to be faced with more risks caused by this dynamic behaviour than before. As a result, the payback period may take longer than expected.

As each contract is designed specifically for each customer (Roy and Cheruvu, 2009), a number of dynamic behaviours is brought in by customer involvement. The **level of asset usage** (i.e. how heavy the asset is used by the customer) affects the in-service time (or remaining life) of that particular asset (as evident in Aircraft Commerce, 2006). Customers may provide **feedback** on product functionality and service performance, and ask for product modification and service quality improvement (Mont, 2002). The asset availability/performance can influence **customer loyalty** on one end (De Coster, 2008), and **contract termination** on another end as it appeared in the vessel service contract (EMSA, 2006). **Contract modification** can take place during the in-service phase, named as the adaptive phase by Roy and Cheruvu (2009).

On top of customer involvements, other factors originate from the OEM policy. **Short-term service contracts** such as fleet management provided by MAN truck (Cantos, 2009) directly impact on total demands and resource availability. One subsystem development in an asset may require an upgrade of other subsystems, which leads to **obsolescence.** From an asset perspective, **asset failure** is one of the most significant dynamic behaviours, which influences asset unavailability and triggers the OEM service activities (Baines et al., 2009c).

2.4 State-of-the-art in PSS modelling

This section presents the state-of-the-art of PSS modelling techniques and tools. Together with the results from Section 2.3, the outcome of this section identifies the research gaps (Section 2.4) and the development of the research programme (Chapter 3).

A systematic process was formulated to conduct the study in the state-of-the-art in PSS modelling. First, the databases that include journal papers, conference proceedings, books and theses were selected. Due to the potential in covering the wide range of databases and the ease of access, SCOPUS, Google Scholar, Ingenta

Connect and Emerald were initially chosen amongst other electronic databases. The second stage involved identification of relevant keywords, which was guided by a number of general PSS and servitisation review papers such as Baines et al. (2009b), Almeida et al. (2008), and Baines et al. (2007). These review papers cover 95 publications related to product-service systems. As a result, the keyword included product-service model, after-sale model, servitisation model, functional sale model, and functional product model. In the third step, a manual search was conducted on the Winter Simulation Conference online proceedings (years 1999 to 2009) and the CIRP Industrial Product-Service Systems (IPS2) conference proceedings (year 2009). To do so, the abstracts of those papers were first reviewed to filter from over 500 down to some 90 articles that were relevant to the work. The selected papers encompass service elements in production environment, and vice versa. The fourth step involved with grouping and removing redundancy through skim-read. Finally, cross-references of these papers were examined. The whole process resulted in 22 most relevant PSS models, as shown in Table 2-1.

2.4.1 Existing modelling techniques and tools within PSS context

Commonly known methodologies are developed from Methodology Development and Evaluation of PSS (MEPSS), and Highly Customised Solution (HiCS) projects. MEPSS was a project funded by EU, which provides a toolkit for successful implementation of new PSS. The methodology was devised into phases; strategic analysis, opportunity assessment, PSS idea development, PSS development, and preparation for implementation (Tukker and Tischner, 2004). Due to the generality, users can enter from any phase, driven by their current status, and can make adjustments to suit their cases. Similarly, the HiCS project also aimed at providing high-level methodology but focused specifically on enhancing partnerships in network. As a result of HiCS project, Solution Oriented Partnership Methodology Framework (SOPMF) was proposed. SOPMF is a method developed and shared by a network to obtain a highly customised solution (Evans et al., 2007). However, these methodologies will not be further discussed as the scope of this thesis relates to modelling techniques.

A model is defined as a simplification of the system that is of interest, and only incorporates the aspects that affect the problem of the study (Banks et al., 2009). Table 2-1 illustrates 22 models formulated from a number of techniques, mainly representing a PSS for the design purpose. It can be seen that many modelling techniques were combined with other analytical techniques such as model for value and cost, analytical hierarchy process, activity-based costing, etc. This thesis will focus only on modelling techniques.

Table 2-1: Classification of techniques and tools (Phumbua and Tjahjono, 2011a)

Reference	Technique	Tool
Low et al. (2000)	TRIZ	n/a
Morelli (2002)	Functional and Use case analysis Blueprint	n/a
Fujimoto et al. (2003)	Discrete Event Simulation	Life Cycle Simulator
Alonso-Rasgado et al. (2004)	Molecular Modelling Service blueprinting Discrete Event Simulation	General programming language
Weber et al. (2004)	Property-Driven Design/Development	n/a
Komoto et al. (2005)	Discrete Event Simulation and genetic algorithm	Life Cycle Simulator
Sakao and Shimomura (2005)	Service Engineering Quality Function Deployment Analytical Hierarchy Process	Service Explorer JAVA2 SDK, Std Edition 1.4.1 XML 1.0
Aurich et al. (2006)	Life cycle oriented method Process modularization	UML 2.0 Integrated Production Process Model (IPPM)
Buxton et al. (2006)	Agent-based Simulation	AnyLogic
Hara et al. (2006)	Service Engineering Petri net Simulation Quality Function Deployment Analytical Hierarchy Process DEMATEL Model for Value & Cost	Service Explorer CPN Tools
Morelli (2006)	Scenarios and use case analysis Service blueprinting IDEF0	n/a
Evans et al. (2007)	Solution map Life Cycle Costing	n/a
Maussang et al. (2007)	Functional Analysis Agent-Based Value Design Use case	n/a
Muller and Blessing (2007)	Process entities/ V-Model	n/a
Komoto and Tomiyama (2008)	Service modelling	Integrated Service CAD and Life Cycle simulator
Shen and Wang (2008)	Fuzzy Extended Quality Function Deployment Analytical Hierarchy Process Optimisation model	n/a
Abramovici et al. (2009)	UML	n/a
Bianchi et al. (2009)	System Dynamics	n/a

Reference	Technique	Tool
Hara et al. (2009)	Service blueprinting Business Process Modelling Notation (BPMN)	Service CAD
Kim et al. (2009)	Ontological representation Activity Modelling Cycle UML OML	Protégé, with conversion to Jess
Schuh et al. (2009)	Life Cycle Cost-Oriented Models Activity-based costing	n/a
Schuh and Gudergan (2009)	Service blueprinting Advance sequential incident Qualitative interdependence analysis Pair Wise comparison Progressive abstraction	n/a

Service blueprinting

Service blueprinting, introduced by Shostack (1982), models the flow of services, time dimension, and time tolerances which can eventually be added up to the total deviation tolerance and the total acceptable execution with a given execution time. In a blueprint, a line of visibility can be used to separate the activities which are not directly associated with the customer but affect service performance from other functions. Within PSS, the technique modelled a list of events generated by a use case (Morelli, 2002). The blueprint demonstrated a service cycle which encompasses technical services, maintenance services, related tangible elements, and intangible elements. The technique was also applied to develop a framework for analytical methods and tools (e.g. pair wise comparison, progressive abstraction, advance sequential incident) for the purpose of designing new PSS solutions (Schuh and Gudergan, 2009). Blueprints can be constructed by a general graphical software tool or BPMN 2.0 package.

Service engineering technique

Sakao and Shimomura (2007) define service engineering as a paradigm which seeks a methodology for designing service. Service engineering technique represents the modelling technique in the paradigm. In service engineering, service providers and receivers are modelled. On the top level, a ‘flow model’ presents the stakeholders in the network. Each pair of these stakeholders is further considered one by one in a ‘scope model’. Their states (e.g. being happy, being unhappy) can be changed as a result of some parameters called receiver state parameters (RSPs) such as cost of service and availability. These RSPs vary across customers, driven by the customer's attributes (e.g. age, social status) and captured in a ‘view model’. Based on these RSPs, associated products and services can be designed and customised to specific customer need. Often, this modelling technique is integrated with other modelling techniques

for detailed analysis, for example, Business Process Modelling Notation (BPMN), Hierarchical Coloured Petri Net (HiCPN), Functional Block Diagram (FBD), Discrete-Event Simulation (DES). To support this service engineering technique, Service Explorer (or Service CAD) was developed based on Java language in Eclipse environment. The tool provides a design workspace to build a model using drag and drop operations, and comprises four elements; scenario editor, flow editor, scope editor and view editor, corresponding to the conceptual models described earlier.

Business Process Modelling Notation (BPMN)

BPMN was used to model a flow of service activities after being designed by the service engineering technique (Hara et al., 2009a). Generally, BPMN models business processes as a flow chart. BPMN offers several fundamental notations such as activity, data, connector, pool, and lane. Within PSS, the authors defined interactions using connecters which link between customers, products, and providers, represented as pools. The staff pools were separated into two lanes which contained customer's visible services and back office services. Regarding BPMN tools, BPMN 2.0 is currently available on platform such as ARIS and Visio which enable simple drag on drop operations.

Hierarchical Coloured Petri Net (HiCPN)

HiCPN was integrated with the service engineering technique to study effects of time and variation in service delivery process (Hara et al., 2009). HiCPN is an extension of Petri Net that breaks down complex systems into hierarchies for simplification. Petri Net models a system using place, arc, transition, and token. An arc, connected between a place and a transition, directs a path to a token. A token is moved once a transition is fired. The simulation was implemented in the software called CPN tools which offers drag and drop operations of the Petri Net conventions.

Functional Block Diagram (FBD)

FBD was used to capture servicing functions such as assisting customers in PSS. In FBD, a block represents a function and has its own input and output. Based on the service engineering technique, Maussang et al. (2007) determined value associated with agents in the network and designed service functions (e.g. assisting customers) associated with each pair of the agents. Products or service departments were then derived from the functions. The author named this technique Agent-Based Value Design.

Discrete Event Simulation (DES)

DES has been tried within the PSS context for two main purposes; to test functionality of a new service support system (Alonso-Rasgado et al., 2004), and to evaluate total life cycle effects on environment and finance. In general, DES models a system in which

the state variables change only at discrete points in time (Banks et al., 2009) using queue and delay as a basis. The technique has been widely adopted in production-based applications. In PSS, the technique was used to develop a life cycle model and its three sub-models: process, product and user (Fujimoto et al., 2003). The first sub-model entailed a network of processes such as manufacturing, operation, recycling and remanufacturing. State changes between these processes were governed by a set of rules. The product model was composed of modules having sets of attributes. The user model managed customers according to types of product packages and their behaviour. The life cycle model was developed to investigate sustainability and economic impacts of a PSS strategy. Another application of PSS was proposed by Komoto et al. (2005) who defined service, service provider, service receiver and products and modelled their subsequent relationships. The author simulated occurrences of PSS events (e.g. repair, reuse) in a number of traditional and PSS business models. The results allowed the impacts on environment and company finance to be assessed. Later, the authors combined DES with Service CAD to enhance primary offering design (Komoto and Tomiyama, 2008). In their study, DES was employed using general language programming and Life Cycle Simulator (LCS). LCS simulates the stochastic behaviour of a product lifetime throughout product life cycle. Actions, performed by actors, were triggered by product behaviour and denoted as events (e.g. repair, dispose). Occurrence of these events was used as a basis for the calculations.

System Dynamics

System dynamics (SD) was used to model the transition between product-oriented manufacturers and PSS providers in a market (Bianchi et al., 2009). SD is generally used for long-term decisions in a broad scope with high level of abstraction where 'structure determines behaviour' (Brailsford, 2008). A quantitative SD allows the rate of change from one state (stock) to another state to be investigated, using stock and flow diagrams. The rate, represented by a flow variable, is governed by mathematical formulations of variables and/or parameters. In PSS context, key success and failure factors of PSS were captured qualitatively using a cause-effect diagram and modelled quantitatively as parameters in a stock and flow diagram. The stocks illustrated product-oriented manufacturers and PSS providers. Connecting between the two stocks were two flows representing *PSS adoption* and *unsuccessful implementation*. The authors demonstrated the use of the model in 1) investigating the impact of different incentive policies on the ratio of traditional manufacturers to the PSS providers, and 2) seeking for the condition where the market constituted equal numbers of traditional and PSS manufacturers (Bianchi et al., 2009). In general, SD software packages such as iThink, VenSim, and AnyLogic can provide drag and drop operations on the stock and flow elements.

Agent-Based Simulation

Agent-based simulation (ABS) was used to simulate the dynamic behaviour of the market to examine the agent's performance in various PSS business models (Buxton et al., 2006). In principle, ABS models interactions between agents having their own decision rules and autonomy. Within PSS, the OEM, the marketplace and the aero engines were all modelled as agents. The model encompassed activities throughout the entire engine life cycle of 50 years. The simulation experiment generated a numerical evaluation of system behaviour with subject to a set of inputs, using AnyLogic. AnyLogic software package is capable of modelling SD, DES and ABS by simple drag and drop operations, and also enables manual creation of object and class using the Java language.

Ontology representation

Ontology representation was proposed by Kim et al. (2009) to model a network of value elements, and relation elements among values, products and services in a PSS context. The value elements consisted of nature, constraints, category and realisation. The relations were classified by type (enable, enhance, and proxy) and subclass (product-value, product-service, service, value). The approach was implemented in Protege, a software package providing ontology graphical editor for the purpose of knowledge management.

Object-oriented modelling (OO)

Based on the object principle of OO, Aurich et al. (2006) developed a process modularisation technique and captured it in a reference life cycle service model. The model had four fundamental objects defined by attributes. The description object contained attributes such as objective, the reference object encompassed attributes such as product components, the function object incorporated information such as support, and the resource object identified physical and non-physical resources. In general, OO is a modelling paradigm that models an entire system as several interacted objects. Each object has its own attributes and methods shared with others. The life cycle oriented concept was also adapted in combination with activity-based costing to generate services throughout product lifetime and enable life cycle cost to be evaluated (Schuh et al., 2009). In correspondence with the objects in the model, Aurich et al. (2006) developed a tool named Integrated Production Process Model (IPPM) for the stakeholders to compile and share processes.

Unified Modelling Language (UML)

Within PSS, UML was implemented to visualise a metadata reference model (Abramovici et al., 2009), to develop an ontology (Kim et al., 2009), and to enable use case analysis (Maussang et al., 2007; Morelli, 2006). UML can be constructed in seven different ways depending on the feature that the model aims to expose (Table 2-2).

Kim et al. (2009) adopted object and class diagram to capture value and relation elements as described before. Morelli (2002, 2006) applied use case to analyse customer behaviour in using a system to design PSS. The same type of UML was implemented in Abramovici's model (2009) to realise information objects entailed in IPS2. The information objects were in accordance with, for example, service, product, resource, usage, and function. Several software packages are available for UML, such as Eclipse UML2 Tools.

Besides the aforementioned techniques, Evans et al. (2007) developed a **solution map** which presented actors in service networks and their associated relationships on the basis of material movement and information flow. An **optimisation model**, in conjunction with a mathematical model and quality function deployment (QFD), was applied by Shen and Wang (2008) to design services that maximised customer's value under implementation cost and technical feasible range constraints. **V-model** was suggested by Muller and Blessing (2007) to be used in the development of product and service modules as well as in synchronisation of processes among stakeholders. The **property-driven development/design (PDD)** was adopted by Weber et al. (2004) in capturing PSS characteristics and properties driven by customers. They developed a shell model which encapsulated three layers; product model on the core, process model, and PSS respectively. Inside each layer were elements such as characteristics, properties, resource, requirements, which were connected and positioned in association with layers. Low et al. (2000) employed **Theory of Inventive Problem Solving (TRIZ)** for systematic designing and clustering products to achieve eco-services. These methodologies are at high level thus employed no specific tools in the development process.

Table 2-2: Seven methods of UML

Method	Focused feature
Use case diagram	Actions of a user in a network
Object and class diagrams	Objects/classes and their attributes.
Sequence and Collaboration diagrams	Interactions between actors
Activity diagram	A flow of activities
State diagram	State changes
Component diagram	Product structure
Deployment diagram	Execution of a product

Overall, it can be seen that the majority of these methodologies are appropriate for the primary offering design stage in which companies have little idea/knowledge what product-service bundles should be offered to customers (e.g. Aurich et al., 2006; Weber et al., 2004; Morelli, 2002; Low et al., 2000). In the later stage, a company will have alternative options and requires a means to evaluate these; simulation techniques were typically used (e.g. Fujimoto et al., 2003; Komoto et al., 2005; Komoto and Tomiyama, 2008).

2.4.2 Model parameters

This section summarises how the models can be used for decision making by investigating the outputs and the inputs (Table 2-3). To illustrate, if a company wants to estimate how much it will cost for a particular offering throughout the contract, the company may consider employing LCS (Komoto et al., 2005; Komoto and Tomiyama, 2008), Service Explorer (Sakao and Shimomura, 2007), or metadata (Abramovici et al., 2009), by checking the outputs from the models. The company may then compare the information required from the inputs across these tools. LCS requires a lot of product property data, Service Explorer needs detailed knowledge of the customer, and the metadata model asks for accurate product usage information. Based on the available data and the nature of the business, the company can decide which tool is the most appropriate.

Table 2-3 initially categorises input data into product, service, system and user, so that the focus of each model can be realised quickly, as demonstrated from the example. By considering a system of product, service and actor elements, the input parameters can be recognised as properties attached to those elements. Some factors that cannot be dictated to any of these elements are classified as system-related. On top of cost, other accounting parameters involve cash flow, net present value and profit. On the lower level, operation measures such as resource level and product specification were included repeatedly in several models, e.g. Abramovici et al. (2009), Sakao and Shimomura (2007), Aurich et al. (2006), Maussang et al. (2007), and Fujimoto et al. (2003). There appears to be also environmental measures, for example, waste amount and resource consumption.

To summarise, Section 2.4 presented state-of-the-art of PSS modelling in terms of existing modelling techniques, supported tools, and model parameters. The findings were used to identify the research gaps in Section 2.5 and develop the research programme (Chapter 3).

Table 2-3: Summary of parameters of PSS models (Phumbua and Tjahjono, 2011a)

Reference	Parameters			
	Product	Service	System	User
Fujomoto et al. (2003)	Price Product development cycle Failure change rate Weight Material Lifetime Recyclability Process energy Process cost	Monthly fee Collection rate Product change fee	n/a	n/a
Alonso-Rasgado et al. (2004)	n/a	Time taken to perform the service	The quality and flow of information	n/a
Komoto et al. (2005)	Module cost Process cost Lifetime Capacity Wear out time Failure rate Price Reparability Reusability	Activity costs Service fee	Number of operations Usage rate	n/a
Aurich et al. (2006)	Specifications	Service process	Resource Information exchange	n/a

Reference	Parameters			
	Product	**Service**	**System**	**User**
Buxton et al. (2006)	n/a	n/a	Sales frequency and volume Engine Flight Hours Engine Flight Cycles Usage characteristics	Customer characteristics
Hara et al. (2006)	n/a	n/a	n/a	Demographic data Psychological data Sense of fulfilment Being well respected Self-respect Fun and enjoyment in life
Morelli (2006)	n/a	n/a	Time dimension Interaction between people Cultural mind frames and social habit	n/a
Maussang et al. (2007)	Quality/condition Cost Time for making design Specifications Time for delivery new product	Available time Cost of activities	Cost of infrastructure Security	Cost of ownership Convenience
Sakao and Shimomura (2007)	Choices	n/a	n/a	Name/Age/Gender Family/Career Excitement Security Being well respect Self-fulfilment Sense of accomplishment Warm relationship Fun and enjoyment Self-respect

Reference	Parameters			
	Product	Service	System	User
				Sense of belonging Personality
Komoto and Tomiyama (2008)	Lifetime Rate of failure occurrence Market size Interval of function release Newness Functionality	n/a	Duration	Preference of user to service type
Abramovici et al. (2009)	n/a	n/a	Product use information (sensor data, environment parameters, maintenance events, failure)	n/a
Bianchi et al. (2009)	n/a	n/a	Initial members Aptitude to PSS transition Dissatisfaction to PSS Barriers to PSS Intensity and duration of incentives First time of activation	n/a
Kim et al. (2009)	Product design Availability	Service roles	Element relation Relation type	User characteristics
Hara et al. (2009)	Visibility to receiver Interactivity with receiver	Visibility to receiver Interactivity with receiver	n/a	Degree of receiver participation

2.5 Research gaps and summary

This section first summarises the current state of PSS modelling on high level, then maps it with the PSS characteristics and dynamic behaviour to identify the research gaps. The gaps that were tackled later by this research are clarified in Chapter 3 based on this gap analysis.

The primary observations from Section 2.4 can be summarised as follow.

- The majority of techniques in the literature produced results in the forms of guidelines, configurations, or specifications, which are appropriate for the high level design stage. Once designed, relationships in the system were mostly fixed.

- There is no modelling framework proposed for different types of PSS, instead, existing frameworks are either abstract or specific to a particular case.

- Simulation models are often valid for one problem/case, mostly for strategy evaluation. The applicability of the model is mostly narrow.

- SD, DES, and ABS were all used in PSS, however, there is no existence of hybrid simulation techniques proposed in PSS literature. This implies that an evaluation could only be made separately between the hierarchies, thus the effects of unexpected behaviours from across levels may be discarded in the models.

- There appears to be a hybrid modelling approach between service engineering technique and DES, in which design alternatives were generated by Service Explorer while the lifecycle costs subsequent to each selection were simulated using LCS.

- There is no guideline or feedback information on the usability and suitability of the simulation techniques for a particular PSS problem.

- The means to assess value in Service Explorer is subjective.

- LCS, which supports DES, is capable of evaluating expenses throughout contracts, yet, visual interactive features are limited.

- Existing measures cover an economical viewpoint (cost, profit), environmental viewpoint (resource consumption, waste amount), financial viewpoint (net present value), and operational viewpoint (performance level, resource level).

2.5.1 Relationships between model parameters and PSS characteristics

The first analysis is the mapping of the PSS model parameters in the literature (Section 2.4.2) with actual contract parameters (Section 2.2.2) and the PSS characteristics (Section 2.3.1). The intention is to analyse the effectiveness of existing models in modelling PSS. This effectiveness is referred to the extent to which the models can explain the changing roles of a PSS provider from being a product manufacturer, in other words, the ability to represent PSS environment.

With regards to the business and strategy characteristics, existing model parameters, such as lifetime, reusability, asset usage and cost of ownership, indicate that the extended responsibility of the manufacturer from design and manufacture to services during the life cycle was considered in several papers. Value parameters, qualitatively driven by customer needs were often entailed in many models. Yet, indicators of service efficiency, such as asset availability, asset operating times, and functional reliability, were not commonly found. The changes of states of assets during the in-service phase, asset's relationships with manufacturer and customer, and the associated risks and penalty, are still limitedly exposed.

In terms of operations and technology, designing service operations to maximise product availability was considered in few papers. The majority of existing techniques use a top-down approach, in other words, the models were built from the system perspective rather than the individual's viewpoint. No model incorporated autonomy of service staff and their variation in skills, behaviour, and decision making. As a result, decentralised decision making was not properly taken into account in any model. Also, it was observed that there was no proposition for new technologies, which in fact supports the study by Manzini and Verzolli (2003). Besides, service response time was not examined as an output. Similarly, service levels were not explicitly incorporated.

In contrast, the interactions with the customers in the network were clearly illustrated in the majority of the PSS models. Nonetheless, the relationships triggered by the product availability and performance, were not emphasised. Similarly, redesigns caused by customer feedback were not commonly found. Government influence which can be a dominant player in public sectors such as the water sector was hardly considered. The degree of interactions between suppliers and manufacturers, market responsiveness and the influences between customers were also not explicitly included in any model. Although the cultural mind sets and social habits were mentioned in some models, none took into account the fact that profit can be improved by efficiency provision and the increase in staff skills throughout the contract.

To summarise, even though the theoretical PSS research has continuously proposed new PSS business models, several PSS characteristics have not yet been covered and realised in the models. The next section looks into the development of modelling techniques in details.

2.5.2 Capability of modelling tools and techniques

The second analysis is the mapping between the capability of existing PSS modelling techniques and tools (Section 2.4.1) with the dynamic behaviour of PSS (2.3.2). The purpose of this section is to investigate the potential of existing techniques and tools in capturing the dynamic behaviour.

A wealth of research incorporates analytical techniques with modelling techniques which in turn allows better numerical evaluation of the PSS offerings. Nevertheless, supporting tools have not yet been fully matured (Hara et al., 2006; Komoto et al., 2005). The majority of the existing tools do not incorporate the time dependent variables and so they did not allow for the dynamic behaviour to be investigated over time. The exceptions are those found in the applied simulation techniques and lifecycle concepts. Nonetheless, the tools and techniques that allow variations caused by customer involvements to be captured are yet to be explored. Although the need for a time dimension was suggested by Morelli (2006), the tool implementation was not presented. Moreover, there was only one paper that mentioned the capability of the tool in encapsulating customer's autonomy in asset operations and in assessing risks from external events beyond the boundary of each agent (Buxton et al., 2006).

These strengths and weaknesses, which lead to opportunities, are summarised below.

Strengths:	**Weaknesses:**
Variety of technique stemmed from product perspective and service perspective. Rich combination between analytical methods and simulation techniques. Lifecycle perspective in an extension to product selling. Various value proposition from the use Explicit interactions between parties in supply chain and customer. Wealth economic and environmental measures evaluation. Clear link between asset transformation and service supports.	Lack of service efficiency measure (availability, service response time) Weak link between product performance and customer-manufacturer relationship, as well as customer involvements and redesigns. Insufficient representation of decentralised decision making process, cultural mind frame, and social habits. Absence of influences between customer, effect of technology on the company's capability, impacts from government.
Opportunities: New definition and customisation of performance measures in PSS New techniques/approaches that can support the design of product-service offering more efficiently Development of operational level, computer-based simulation tools that incorporate the dynamic behaviour of PSS Better illustration of PSS modelling techniques and tools through case studies and industry implementation	

Figure 2-2: Strengths, weaknesses and opportunities of PSS modelling research (Phumbua and Tjahjono, 2011a)

Figure 2-2 reveals a number of weaknesses in the PSS modelling literature. This research attempted to bridge these gaps, detailed in the next chapter.

To conclude, the importance of PSS as a survival strategy for today's market was highlighted in this chapter (Section 2.). It also introduced successful cases and decision parameters that appeared in aircraft, train, printing, washing, and carpet sectors (Section 2.2). The extended characteristics of PSS beyond traditional business as well as dynamic behaviour were addressed (Section 2.3). The applied modelling techniques and tools in the PSS offering domain were presented and reviewed in Section 2.4 in terms of their applicability and potential to evaluate alternative PSS offerings (Section 2.5). The review reveals that the research on PSS decision supports is still lagging far behind the theoretical PSS developments. Techniques and tools that can effectively describe PSS and enable dynamic behaviour to be captured are therefore needed. Without a tool, a company may be reluctant to adopt PSS or may fail to sustain service contracts or may be at great risk of losing business. Based on this outcome, the research programme presented in Chapter 3 was developed.

3 Research programme

The previous chapter provided the detailed justification towards the development of this research programme. In this chapter, the development of the research aim and subsequent objectives are presented in Section 3.1. The scope of this research is covered in Section 3.2, and the development of the research methodology is explained in Section 3.2. Finally, Section 3.4 summarises this chapter.

3.1 Development of research aim and objectives

The driving force towards PSS for manufacturers (Section 2.1) highlighted the need for a customised decision making tool prior to signing a contract. The scope of OEM's responsibility that goes beyond manufacturing activities (Section 2.2 and 2.3) suggested the inadequacy of the modelling approach typically adopted in the manufacturing paradigm. The review of the existing approaches (Section 2.4) identified simulation modelling as potential in enabling detail evaluation of alternative offerings. Nevertheless, the research gap analysis (Section 2.5) revealed that the simulation modelling research within PSS context has been in the early stage of development.

Based on these findings, there were two alternatives for this research; developing an accurate and detailed approach for a specific PSS case, or seeking for general modelling constructs with sufficient details. The first option has less applicability but provides in-depth knowledge. The outcome from this option can be made as a use case. On the contrary, the second alternative can be applied to several cases. As PSS research is still in an early stage, the second approach which enhances applicability is more appropriate. Therefore, the aim of this research has been developed

to propose the modelling constructs that enhance effective and efficient development of service contract simulation models for PSS offerings.

The following research objectives were defined to set up the stages towards the aim.

- To identify a simulation technique that can potentially capture PSS characteristics and dynamic behaviour
- To determine an appropriate modelling approach that enables effective and efficient development of PSS simulation models
- To form primary modelling constructs based on the approach
- To evaluate and refine the primary constructs
- To present the final constructs

3.2 Development of research scope

This section defines the research context according to the aim, in other words, the scope for "PSS offerings". To do so, there are two major issues to be addressed: various closely related terminologies to PSS and the selected research boundary.

First, 'Product-Service Systems' is defined as an integrated product and service offering that delivers value in use (Baines et al., 2007). Besides PSS, the less-often-used terminologies which also refer to the integrated offering concept are 'functional sales' and 'functional products' (Lindahl et al., 2006). The process of creating value by adding services into a product is denoted as 'Servitisation' (Baines et al., 2009b). 'Product-Centric Servitisation' is the case of servitisation in which the product itself is central to the provision of services (Baines et al., 2009a). Special cases under the product-service concept are 'availability contract', 'Performance-based Logistic (PBL)', 'Performance-Based Contract (PBC)' and 'Contractor Logistics Support (CLS)'. All these four cases share the same goal in providing system readiness, but used in different countries. Within a PSS context, 'technical Product-Service System (t-PSS)', also referred to as 'Industrial Product-Service System (IPS^2)' (Erkoyuncu, 2011), focuses on business-to-business, high value assets (Meier et al., 2010; Aurich et al., 2006). IPS^2 was used interchangeably with 'Contracting for availability (CfA)' in Erkoyuncu (2011). CfA is a commercial process which aims to sustain system readiness at an agreed level over a period of time (Ng, 2008).

It can be seen that all the presented terminologies are closely linked and sometimes used interchangeably. To avoid confusion, this thesis considered all product-service offers as PSS and the research scope is addressed below.

Prior to the selection of the research scope, the first option was considered from a well-known classification developed by Tukker (2004). The classification was ranged on the significance of product to the total offering, categorised as product-oriented, use-oriented, and result-oriented. At one end, services in product-oriented PSS are mainly for enhancing product sales. At the other end, products are not pre-determined in result-oriented PSS but driven by the required capability. Therefore, ownership of the asset is often shifted to customers in product-oriented PSS, but remained with the OEM in case of result-oriented PSS. The scope of this research was not based on this classification due to two main ambiguities. To illustrate, consider the following examples: 1) Pratt & Whitney was awarded a service contract to support F100-PW-220E engines for the US and Italian Air Force (Pratt & Whitney, 2011), 2) Rolls-Royce signed a service contract on RB199 engines with the UK Royal Air Force (RAF) (Rolls-Royce, 2010), 3) Boeing provides support to C-17 for the US Air Force (Mahon, 2007). These cases aim to provide capability but the assets are specified, already sold from the OEMs and owned by the customers. This means the business models can fall under result-oriented PSS by considering the aim, as well as product-oriented PSS based on the importance of assets and the ownership perspectives. Additionally, this

classification includes some pure service businesses in PSS, for example, car sharing and a taxi model. Therefore, the boundary of this research was not based on this classification.

The alternatives were considered from the above mentioned terminologies. However, availability contract (or PBL/PBC) is too specific, and IPS^2 and t-PSS require several criteria to identify the cases. On the contrary, "product-centric servitisation" focuses on one key characteristic (i.e. the inseparable services to a specific asset) and leaves flexibility in terms of asset characteristics, ownership, and performance requirements. The definition also clearly indicates that the service contracts which are not tied up with specific assets are excluded (such as an activity management type contract and a product pooling contract), whilst all aforementioned examples of Pratt & Whitney, Rolls-Royce, and Boeing are clearly included. Thereby, the scope of this research adopted the product-centric terminology to PSS offerings.

Service contracts within a product-centric PSS context are often made in **long term** and indicate services that will be applied to the particular **tangible assets**. At a detail level, the contracts in the product-centric PSS context have the following characteristics:

- The contracted assets will be used by the same customer and will be replaced primarily on a malfunction basis. However, these assets can be shared with other customers if they are not used by the contracted customer.

- In case of asset replacements, the replaced asset must have identical structure to the original asset.

- The contracted assets are not necessary to be manufactured by the OEM.

- The contracts can be made on the used assets as well as new assets.

In conclusion, the aim, objectives, and the scope of this research enabled the gaps of knowledge that were handled by this research to be identified. Based on the strengths and weaknesses in the PSS modelling literature analysed in the previous chapter, there are six areas excluded in this study. First, this research focuses on product-centric context, thus, the modelling constructs may not be applicable in the service-centric context. Secondly, embedding an analytical technique inside the constructs was not necessary as this research aimed to provide modelling capability rather than the accuracy of a solution. Thirdly, service contracts are made between the OEMs and their customers therefore the interactions between parties in supply chain limit to the OEM-customers. Fourthly, environmental measures were excluded as it is unlikely that OEMs will sign a contract based on environment impact. Fifthly, cultural mind frame, social habits, and influence between customers are generally not in the OEMs' concern in deciding whether to sign a contract. Last, governments rarely influence the markets in the product-centric PSS context, therefore, its effect was not considered in this research.

The next section deals with the formation of the research methodology based on this direction.

3.3 Development of research methodology

There are some differences between research methods and research methodology as pointed out by Rajasekar et al. (2006) that research methods are procedures used in the research whereas research methodology is a systematic way to solve a problem. Therefore, a research methodology is formulated from a systematic management of research methods.

In this research, research methodology has been formulated into five stages corresponding to each objective (Figure 3-1). In Chapter two, simulation was addressed as a potential approach for PSS offering design. Stage I to Stage III deals with development of the modelling constructs. Therefore, it was important to realise the research methods for developing a simulation model at these stages. On top of that, the research methods for the data collection were necessary to form a model. In the fourth stage, the focus was shifted to evaluation of the primary constructs, thus, the research methods are linked with collection of data that comes from implementation of the constructs. The final stage focuses on presentation of the constructs. For these reasons, prior to the formation of the adopted research methodology, relevant research methods from literature were explored. Section 3.1 focuses on research methods for simulation studies and Section 3.2 addresses data collection methods and the rationale towards method selection. Finally, the selected methods were formulated into the research methodology, described in Section 3.3.

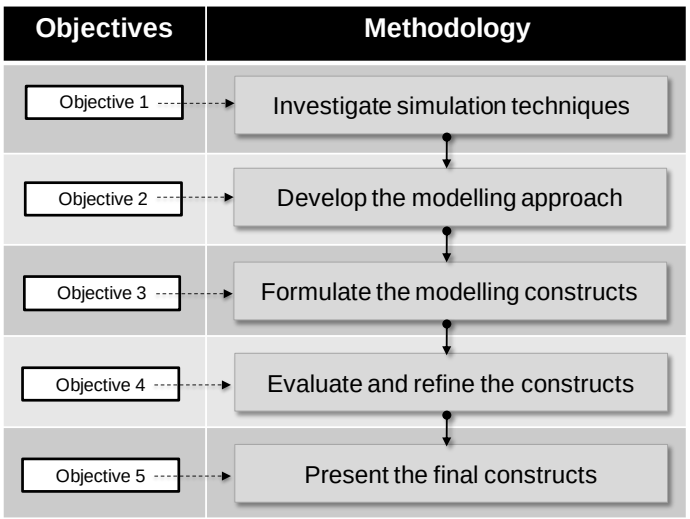


Figure 3-1: The five-stage methodology

3.3.1 Methods for conducting a simulation study

The general methods for conducting a simulation study are linked with the key modelling processes, and have been discussed by several authors, for example: Banks et al. (2009), Robinson (2004), and Bennett (1995). However, Robinson (2004) stated that the key modelling processes are very similar across different authors and the only difference is in terms of terminology. The author summarises these processes as shown in Figure 3-2.

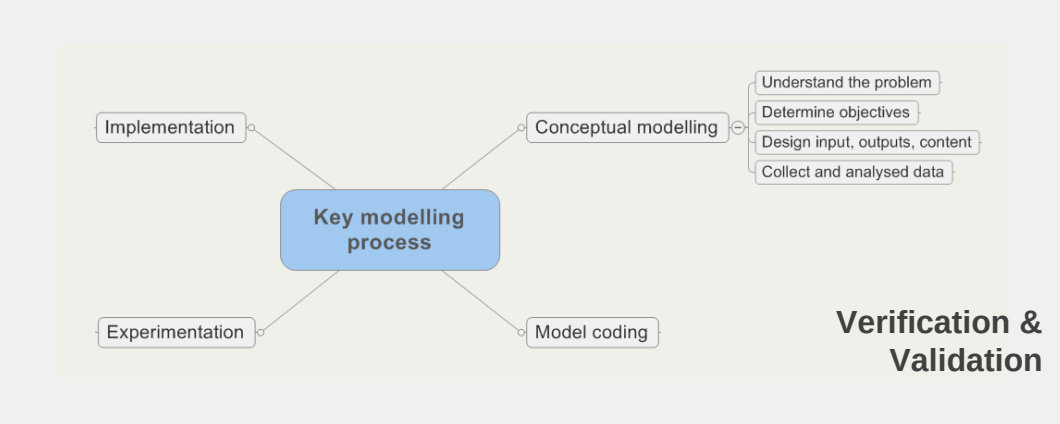


Figure 3-2: Key modelling processes (Adopted from Robinson, 2004)

It was pointed out that these processes in Figure 3-2, including sub-process, are not linear and can be cross-linked. Within the conceptual modelling phase, Robinson (2004) highlighted that conceptual models should have four components; validity, credibility, utility, and feasibility. Validity is defined as a modeller's perception that the conceptual model will produce a computer model with sufficient accuracy for the purpose at hand. This definition is also applied to credibility but from the model user's viewpoint. Utility relates to the usefulness of the model in assisting decision-making within the context. Finally, feasibility is concerned with the perception that a development of computer model is achievable based on the conceptual model. The implementation process can be achieved in three ways; applying the solution, using the model, and learning from the model.

In addition, two other processes are suggested to be performed throughout model development; model verification and validation. The purpose of verification is to check if the model performs what it supposes to do, whilst validation aims to match the model to the real world (North and Macal, 2007). Banks (2007) provides a full list of verification and validation techniques, which covers over fifty methods. However, this thesis presents those that are frequently suggested in literature.

In terms of verification, the following methods can be summarised from Banks (2007), Law (2007), North and Macal (2007), and Robinson (2004):

- Iteratively compare the model with the design document.
- Breakdown the whole model into several loops of events, and run the model after completing each loop.
- Assign a fixed number instead of a distribution, and force extreme conditions to occur.
- Manually run the models in single step. Timescale of occurrences may be reduced to make the step walk-through possible.
- Document all the code and choices of methods, and walk through the code.
- Record the model results and check if they are realistic.
- Check the model log.

However, keeping the model logged can be costly and referring to the design document may not be possible in exploratory research, especially in ABS context, as most substantial designs undertake major revisions prior to completion (North and Macal, 2007).

Validation can be made in terms of requirements, data, result, process, solution, and theory. Additionally, agent validation (in case of ABS model) can be performed by comparing agent behaviour and interactions with the real world behaviour (North and Macal, 2007). The choices of validation are recommended to be considered against the purpose of modelling (Robinson, 2004). The methods frequently covered in the literature (Banks, 2007; Law, 2007; North and Macal, 2007; Robinson, 2004) are as follows:

- Select real world cases with valid scenarios (or known results from other approaches), and compare the results with real world (or the results from other approaches). This includes checking whether the outputs realistically correspond to the change of inputs.
- Compare across multiple cases.
- Refine the model with subject experts.
- Externally assess the usefulness of the model via third party.

In case of SD and ABS paradigms, these general guidelines for simulation studies can be further detailed within conceptual modelling and model coding phases. In terms of ABS, Macal and North (2010) stated that similar ways can be conceptually done as in other models. These processes include: 1) identify the purpose of the model 2) analyse

the system under study 3) identify entities and their interactions, and 4) address sources of data. The common components to be identified include agent, agent's activity, relationship configuration, agent's goal, environment sensing (or message receiving) method and action method (Cavrak et al., 2009; Macal and North, 2010; Jennings, 2001; Guessoum and Briot, 1999). As for SD, key factors still need to be addressed. Nonetheless, they are not explicitly separated as inputs and outputs. The hypothesis should be formulated on the basis of internal effects which come from the feedback structure (Sterman, 2000). This causal structure should be represented using tools such as an influencing diagram or a stock-and-flow diagram. These processes are recommended by Sterman to be completed prior to making simulation models.

Overall, the methods in simulation study are well defined and generalised to any case. Therefore, these methods were adopted in this research, explained in Section 3.3.3.

3.3.2 Methods for data collection

This section provides general research methods in literature that relate to data collection, and the rationale as to why the particular methods were selected in this research.

The first step towards method selection was to decide whether this research is quantitative or qualitative. Quantitative research usually makes use of numerical analysis, whilst qualitative research is non-numerical and the data are collected in the form of words and observations (Rajasekar et al., 2006; Johnson and Harris, 2002). Creswell and Plano Clark (2007) highlighted that qualitative research is usually intended to learn participant's views about something whereas quantitative research is often aimed to support or disprove existing theory.

In this research, the data required in the first three stages are linked with the first key modelling process; understanding the problem. These data refer to decision parameters and modelling scenarios in the models. Although numerical input values are necessary, they are not critical considering that the aim of this research is not to provide the contracting solution for a particular case. Therefore, the required data are in descriptive format, which identifies this research as qualitative.

A number of research methods exist in the literature for qualitative studies. Creswell (1998) compared key characteristics of five methods; biography, phenomenology, grounded theory, ethnography, and case study. Biography is the study of individual life, Phenomenology deals with understanding of experiences from a phenomenon, grounded theory relates to development of theory from data, ethnography is associated with social interactions and culture, and finally, case study is an exploration of a system.

In the context of this research, using biography implies that contracting processes would be explored from personal viewpoints of the interviewees rather than the actual

processes. This is similar to phenomenology, as described by Goulding (2004) that interviewee's view is taken as fact. Applying ethnographic and grounded theory in this research would require a considerable amount of time for data collection, as Creswell (1998) described that grounded theory requires interviews with 20-30 people and ethnographic rely on observations and interviews after long time spent in the field. On the contrary, case study has none of these limitations. For these reasons, case study was selected as the research method for data collection during the first three stages in this research.

During the final stage of this research, the data relate to feedback from constructs implementation. Here again, grounded theory and ethnography would take a long period of time. Biography was not valid at this stage as the collected data were not in terms of an individual's life. Phenomenology could have been used by interviewing participant's experiences after implementing the constructs. This would practically be the same as conducting case studies by considering one user as one case (i.e. pilot case study). Accordingly, case studies were also selected to obtain the constructs implementation data.

The major sources of evidence for a case study were suggested by Yin (2009) as documentation, archival records, interviews, direct observations, participant observation, and physical artefacts. Participant observation differs from direct observations from the point that the researcher also takes a role in the system being studied. Yin provides a list of strengths and weaknesses of each source. Overall, bias is a major issue in case study research.

To cope with bias, Robson (2002) suggested some techniques to enhance trustworthiness in qualitative research. Trustworthiness is considered in terms of *validity, reliability,* and *generality* perspectives. Validity is linked with correctness of results, reliability deals with the use of standardised instruments in collecting data, and generality relates to the applicability of conclusions. To enhance the trustworthiness, several techniques can be adopted, as follows:

1. Prolong involvement of the researcher in the field.
2. Combine qualitative with quantitative method (methodological triangulation).
3. Use multiple sources for data collection (data triangulation).
4. Cover various theories (theory triangulation).
5. Include more than one observer (observer triangulation).
6. Discuss with a group of researchers (peer debriefing).
7. Re-check with the participants (member checking).

8. Search for instances that would disprove the researcher's theory (Negative case analysis)

9. Keep records of activities during the study (audit trail).

10. Standardise instruments for data collection.

The next section describes how these methods were adopted into this research and addresses how the weaknesses of the applied methods were handled.

3.3.3 The research methodology

The methods in Section 3.1 and 3.2 were structured into steps for the five-stage methodology as shown in Figure 3-3 and explained below.

Stage I: Investigate simulation techniques

This stage corresponds with the first objective of this research. The intention of this stage was to seek for an appropriate simulation technique that enables the characteristics and dynamic behaviour of a product-centric PSS to be captured. Therefore, the main research question of this stage is:

What are the strengths and weaknesses of each simulation technique in modelling service contracts?

The research question implies a need of insight knowledge on simulation techniques. To answer this question, both theoretical and practical investigations were conducted. Simulation literature was initially explored, followed by actual model development in correspondence with each technique. The actual model developments were necessary as the PSS context spans beyond a manufacturing context and there has been no analysis on technique capability in PSS.

Three potential techniques have been identified from Section 2.5: SD, DES, and ABS. Even so, ABS is often found embedded inside a larger system or built upon other techniques (Macal and North, 2010). Accordingly, four models were examined in this stage based on SD, DES, hybrid ABS-SD, and hybrid ABS-DES.

Within development of each model, the key modelling processes described in Section 3.1 were applied. These models were constructed based on the understanding of product-centric PSS offers from the literature, and focussed on different sectors to gain general knowledge. The verification and validation methods performed included conceptual validation with PSS experts, break-down analysis of models, the use of extreme situation, debugging walk-through, verification with a simulation expert, documentation, and external presentations. The use of experts and external presentations also allowed member checking technique to be undertaken.

To evaluate the capability of the techniques, they were assessed against the strengths and weaknesses in PSS modelling literature analysed in Chapter 2. The most appropriate technique was considered to be the one that better represents the PSS context.

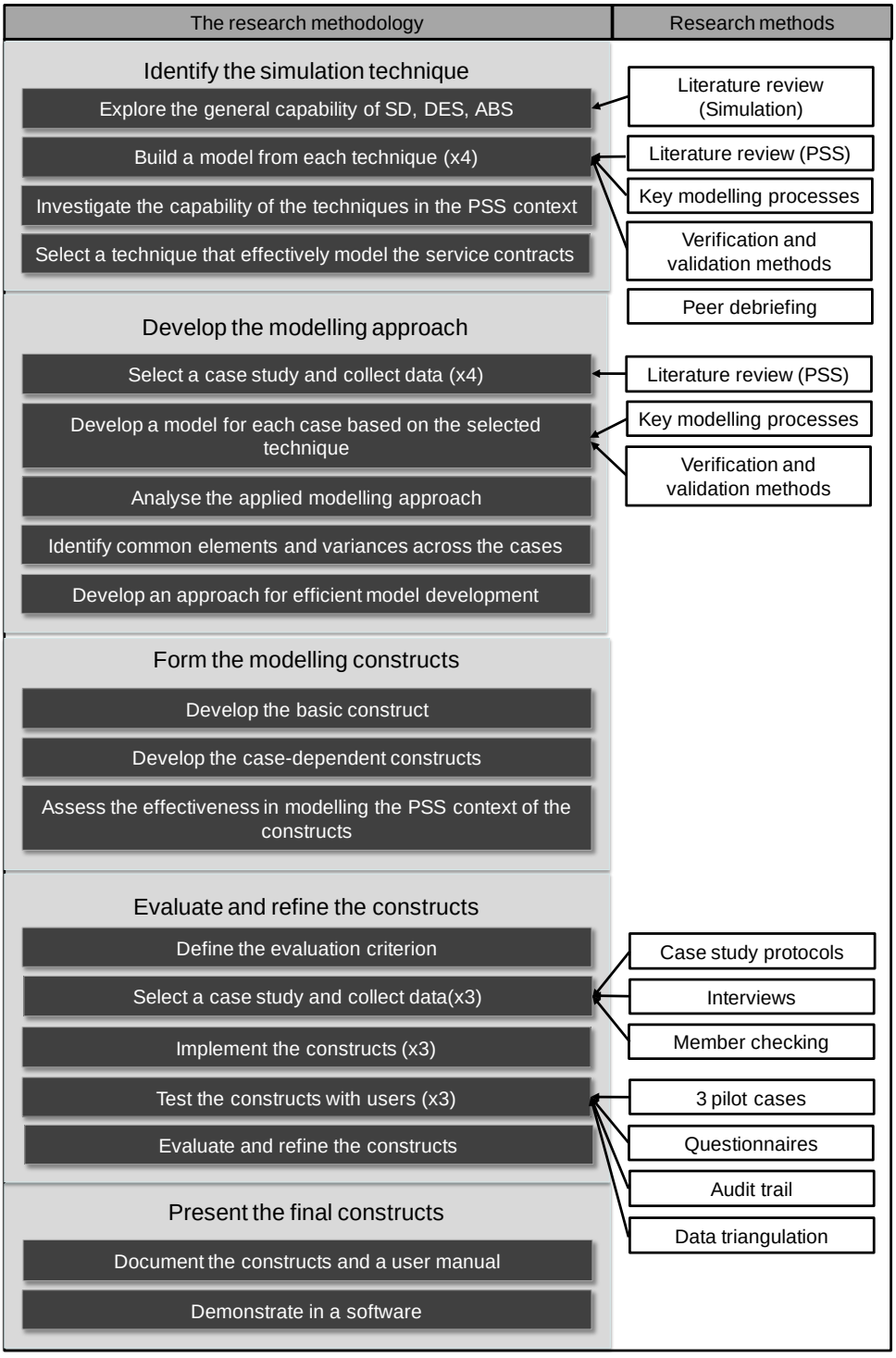


Figure 3-3: The research methodology adopted in this thesis

Stage II: Develop the modelling approach

This stage corresponds to the second objective of this research. It aims to develop an appropriate modelling approach that leads to effective and efficient development of simulation models within the PSS context. Therefore, the main research question for this stage is:

How should service contracts be modelled for efficiency?

This question requires experiences on using the selected simulation technique. The approach should also incorporate levels of generality. To achieve these, several model developments from various case studies which apply the final selected technique were necessary. Therefore, four cases were selected from the aircraft, the photocopier, the underground, and the carpet sectors.

Similar to the previous stage, the data used in the models were in terms of modelling scenarios and key decision parameters. These data were collected from literature, and based on product-centric PSS. The verification and validation processes covered conceptual validation with PSS experts, break-down analysis of models, the use of extreme situation, debug walk-through, code walk-through with a simulation expert, documentation, and external presentations.

Having developed the models, the applied approaches were analysed and refined to enhance efficiency in model development. The final approach was assessed against the strengths and weaknesses in PSS modelling literature to primarily evaluate its effectiveness in modelling PSS business.

Stage III: Form the modelling constructs

This stage is linked with the third objective of this research which aims to form modelling constructs. Other approaches were also considered, for example, templates and workbook. However, the number of variants across cases can result in excessive number of templates. Moreover, a template gives limited flexibility for further customisation which can be caused by different operating policies inside each company. For instance, a ship building company may aim to adjust resource levels based on ship utilisation which depend largely on the weather condition whereas a train company may maintain the same number of staff throughout a year. Using a template, these decision rules can be hard to modify. On the contrary, a workbook may better handle the variants. However, using a workbook, modellers may spend too much time in understanding the generic information about their cases. For these reasons, modelling constructs were considered as a compromised option. The constructs comprise modelling elements (e.g. source, sink, state, event, transition, etc.) and methods (e.g. message passing mechanism, conditional trigger, action firing mechanism). Following the refined approach from the previous stage, this stage

implemented the approach in a software package. Therefore, the research methods at this stage relate to technical programming.

Stage IV: Evaluate and refine the constructs

This stage is associated with the final objective of this research. The purpose was to evaluate the developed constructs in terms of effectiveness and efficiency in enabling model developments. The validation outcomes led to the refined constructs including instructions how to use them. Accordingly, the main research question of this stage is:

Do the constructs really enable effective and efficient development of service contract simulation models?

To evaluate effectiveness and efficiency, the criteria for assessment were required. Efficiency was measured by comparing the model development times between using the constructs and without using them. The effectiveness at this stage was evaluated from three criteria:

- The applicability of the constructs to existing cases
- The practicality of the constructs as an aid in making contracting decisions
- The feasibility in developing a model based on the constructs

Therefore, the data required at this stage deal with the evaluation of the constructs based on these criteria.

Multiple case studies were selected within product-centric PSS. The focus of this evaluation was to validate the extent of efficiency, applicability, practicality, and feasibility of the constructs. Within the case study validation, interviews with three existing cases were conducted to obtain the data for model customisation. Case study protocol was used across cases to ensure consistency, thus, enhancing trustworthiness of the results. The protocol was structured for system understanding, and mapping case characteristics with the details in the constructs. Again, all cases are different and fall under the product-centric PSS context. Peer debriefing was also performed to enhance trustworthiness, by cross-checking the collected data with another student who was also in the interviews. Additionally, the developed models from each case were re-checked with the interviewees.

Another form of evaluation was carried out using third parties. User validation was conducted in three pilot sessions which correspond to the tests by a simulation learner, a DES expert, and a simulation expert. The participants were selected as they have been involved in PSS research and all have different levels of simulation background. This selection aimed to gain insight feedback and generalise results. Data triangulation was performed based on direct observations and user feedback, and the

activities were recorded. Questionnaires were given to the participants to structure their feedback and enable the assessments of the four criteria.

Based on the evaluation, the amendments to the constructs were identified to enhance effectiveness and efficiency in developing service contract models.

Stage V: Present the final constructs

This stage corresponds to the fifth research objective which relates to presentation of the final modelling constructs. It was intended to instruct users how to use the constructs to build a model. Therefore, the presentation included the overview of the constructs, instruction how to use it, the constructs, and example of the implementation on a software package. The outcome of this phase is the main contribution of this research.

3.4 Chapter summary

This chapter detailed the research programme. Section 3.1 clarified the research aim and the five research objectives. Section 3.2 defined the context of this research as product-centric PSS. Section 3.3 dealt with justification and structuring research methodology, in which methods that centred around case studies and simulation were described and led to a five-stage methodology. The next section presents the first stage of the methodology, which relates to investigation of an appropriate simulation technique.

4 Investigation of simulation modelling techniques

Chapter 2 described a primary evaluation of PSS modelling techniques and tools in terms of potentiality from the theoretical background. The outcome identified three potential techniques; SD, DES, and ABS from existing techniques in PSS context. This chapter corresponds to the first objective of this research, which aims to select an appropriate simulation modelling technique as the core architecture for the modelling constructs. To achieve this, Section 4.1 presents general strengths and drawbacks of these techniques from simulation literature. Section 4.2, guided by the literature findings, illustrates model developments based on these techniques in various PSS cases. Section 4.3 provides assessment of the techniques in the PSS offering context against the current state of PSS modelling, which led to the selection of the final technique. Finally, Section 4.4 concludes the chapter.

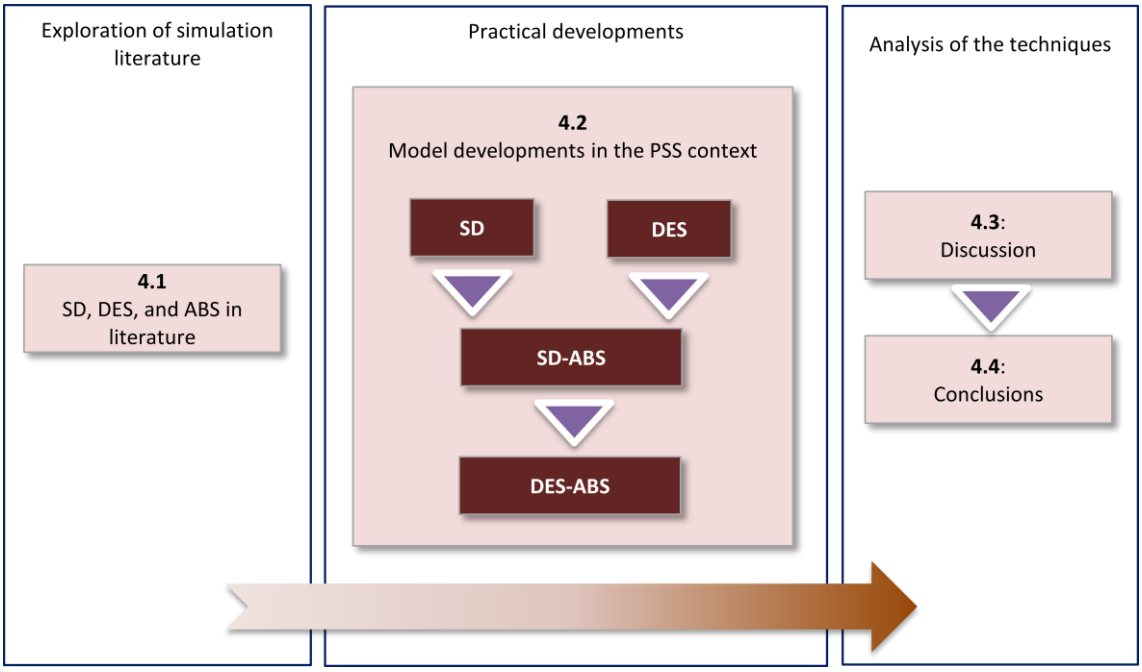


Figure 4.1: Chapter 4 outline

4.1 SD, DES, ABS in literature

This section deals with general strengths and weaknesses of each simulation technique, obtained from simulation literature.

Both SD and DES focus on a system level whilst ABS takes into account complex relationships caused by interaction among agents (people, products, assets, etc.). These agents can have different histories, intentions, desires and individual properties, and are able to influence each other. By capturing interactions between these individuals, ABS allows unexpected phenomena to emerge (this type of phenomena is often referred to as a non-linear relationship). This capability can enable non-rigid or

non-defined relationships to be investigated which is lacking in other two techniques (Jenning, 2001). Additionally, an agent can sense the environment, decide whether the information matters to its own goal, and behave accordingly (Macal and North, 2010; Guessoum and Briot, 1999). Therefore, ABS is flexible to changes and provides a natural description of a close-to-reality system (Bonabeau, 2002). This agent capability contradicts the usual passive nature of an object in other techniques. However, a major drawback of ABS was reported by Yu (2008) in terms of the unrepeatability of experiments due to the non-rigid interaction between agents.

Several studies further compared SD and ABS from a practical viewpoint, ABS encourages the modeller to focus on defining agents and their behaviour rules, whilst SD provides conceptual description and force the modeller to consider at an appropriate level of aggregation (Datta, 2007; Wakeland et al., 2004). Therefore, ABS investigates leverage points in complex aggregate systems through rules and agents, while SD discovers the points through a feedback structure of the systems. Accordingly, SD is deductive in that individual agents or events do not have a lot of influences (Scholl, 2001).

In terms of DES and SD, several comparisons have been made (e.g. Chahal and Eldabi, 2008; Morecroft and Robinson, 2005; Tako and Robinson, 2005). Among these publications, Chahal and Eldabi (2008) addressed the general differences between the two approaches from a literature survey and grouped them into methodology, problem, and system perspectives. The methodology viewpoint involves, for example, philosophical assumptions, technical capabilities, limitations and characteristics of the modelling techniques. The problem context justifies the reason why each technique is considered appropriate. Finally, the system perspective is associated with the nature, representation, and views of the system in real world. Key strengths of SD lie in the fact that its structure is relatively flexible, a standard format can exist, the effect of outputs to the system can be captured through a feedback loop, and the system is not isolated but connected to the 'world'. These features are not commonly found in DES (Chahal and Eldabi, 2008). Nonetheless, DES structure is clearly defined, well animated and tangible, and more importantly, the effect of randomness on model outputs can be comprehensively examined. For these reasons, SD can potentially cope with dynamic complexity whereas DES is powerful in handling detail complexity. The confidence in using an SD or DES model remains an ongoing argument and case-dependent. While the finding from Chahal and Eldabi (2008) gave more credit to DES, Tako and Robinson (2005) and Han et al. (2005) revealed that similar levels of details and accuracy can be produced from both techniques.

Overall, Siebers et al. (2010) suggests that technique selection is dependent on the problem, and not application. Macal and North (2010) stated that the modellers should stick with the familiar technique and seek for others when that applied technique cannot fully describe the problem. Along this line, several propositions have been made to combine techniques, for example, Rabelo et al. (2005), Borshchev et al.

(2004), Schieritz and Globler, (2003), Lee et al. (2002), and Scholl (2001). Based on these findings, the models developed from a single technique were explored prior to those from hybrid techniques. The next section describes this process.

4.2 Model development in the PSS context

As there was no technique analysis within the PSS context (Phumbua and Tjahjono, 2011a) and the simulation literature revealed its dependencies on cases, it was inadequate to identify an appropriate technique for the PSS context merely based on the literature. Therefore, actual model development was necessary in this research.

The investigation on single simulation technique is first presented, followed by hybrid simulation developments. Each single technique assessment responds to each independent PSS offering problem, whereas the contexts for hybrid technique arose from the insufficiency of the single techniques. The structure of this section is divided into several sub-sections corresponding to each technique assessment. Only the analysis will be presented in this chapter as other details are not pertinent with the focus of this chapter. However, more information is provided in the Appendices.

Robinson (2004) stated three methods for developing a simulation model; spreadsheet, general language programming, and software packages. As software packages provide better ease of use and validation and spreadsheets have limited capability in showing animation and queue (Robinson, 2004), a software package was used throughout this research. The introduction to software elements, based on AnyLogic®, is provided in Appendix A. Anylogic® is a multi-paradigm simulation modelling tool which enables SD, DES, and ABS to be implemented simultaneously. The notations used in the software will be also adopted in this thesis.

The verification of each model follows the research methods detailed in Section 3.1, which is summarised in Table 4-1. The empty cells imply the methods that are not applied to a particular model. At least six methods have been applied to ensure the model functionality. In this chapter, the business cases are drawn from the theoretical need for the study, and supported by existing cases in the literature.

Table 4-1: Summary of applied verification and validation methods

Validation method	SD model	DES model	SD-ABS model	DES-ABS model
Business case exists	■	■	■	■
Validate concept with PSS experts	■			
Breakdown the model into several functions and check one by one	■	■	■	■
Simplify the input value, force extreme condition and observe expected outcomes	■	■	■	■
Manually step through the models and check expected outcomes	■	■	■	■
Verify model with a simulation expert	■	■	■	■
Validate the model with a practitioner				
Document the model and recheck it twice	■	■	■	■
Present the model to an external organisation	■		■	■

4.2.1 Business case I: SD technique

This study was carried out jointly with a group project at Cranfield University, which aimed to develop general PSS business models for OEMs who are keen to implement PSS. In this project, three PSS business models were constructed using influential diagrams in order to capture three different servitisation levels. This difference is a result from skilled workforce, monitoring technology, OEM-customer relationship, and OEM-supplier relationship. The servitisation level can attract more customers, thus, this interconnection is represented by stock and flow diagram. Details can be found in Appendix B. Three case studies were conducted to demonstrate the concept, followed by presentation to several PSS experts. Feedback from the project was evaluated, which led to the final models used in this research.

The findings were analysed based on the feedback and the weaknesses identified from PSS modelling literature. In terms of feedback, the subjective weighting and scaling of SD appeared as a major issue in the model's applicability. The interviewees were unsure how to estimate the four factors. For instance, the difference between values for the relationship with the customer between five and six could not be clearly justified. Moreover, the values of weight importance can be given differently by different people from the same sector. In terms of modelling, the SD structure was

simple to form in this case and powerful in demonstrating the impact between decision parameters. The latter aspect is beneficial in strategic investment.

To compare against the weaknesses of existing work as dictated in Figure 2-1, this model could present service efficiency measures (response time, availability), and expose connections between 1) OEM-customer relationship and product performances, 2) customer involvement and redesigns, 3) technology and the company's capability, and 4) transformation of asset and service support. Nonetheless, the technique was not appropriate to illustrate decentralised decision making, the individuality of assets, as well as the stochastic nature of in-service activities. The connection between 'value' and product/service could be captured vaguely at a high level. Asset life cycle and interactions between stakeholders could not be illustrated clearly. Therefore, SD on its own was not considered sufficient to be used in the PSS context.

4.2.2 Business case II: DES technique

The second study looked into different asset usage scenarios of PSS contracts presented in Section 2.2. Asset usage was selected since it appears as the key issue to be specified in all reported contracts. It can also directly affect payments and scope of OEM responsibility. For example, a failed asset outside contract hours does not affect availability level or cost OEMs any penalty. In this study, assets in scenario one are contracted all the time, as in military aircraft contracts. In scenario two, contracts are made on an hourly-basis, as in the case of Xerox and LUL. Finally, scenario three combines both scenarios, in other words, some assets are contracted all the time and some are not. This scenario enables customers to chase demands in their businesses. Three models were developed in correspondence with the three scenarios, explained in Appendix C.

Observations related to technique capability were drawn in terms of level of workarounds made to the models, insufficiencies to present PSS, and contribution to PSS modelling.

Workarounds beyond modelling typical manufacturing system are required due to the shift in modelling principle. In the manufacturing context, the common elements typically include parts and their attributes (e.g. part number, due date, etc.), resources (e.g. machines, workers), and queues or buffers (Chung, 2004; Law, 2007). A part is created in the model, moved through a system in which resources are requested, and usually discarded from the model. An arrival of entities (or passive objects) generally activates activities in the manufacturing system, which dictates the 'demand signal'. If the resource is available, the part will be processed, otherwise it is queuing for operation. In terms of performance measures, the typical outputs described in Chung (2004) are in-system time, waiting time, average number of parts in queue, and resource utilisation. Additionally, Law (2007) listed other measures, for examples, throughput, timeliness of deliveries, numbers of process inventories, proportions of

parts that are reworked or scraped, and proportion of time a machine is not working. In summary, the outputs of a model are based on time, number of parts, and resource status. In PSS, an asset is equivalent to a 'Part' in the model. These parts are moved back and forth between the OEM and customers, depending on whether they are in the contract period and capable for an operation. Therefore, the shifts in the modelling principle are as follows:

- From an element viewpoint, 'Delay' elements in a customers' site are considered as in-service assets rather than process activities. Therefore, the element's cycle time does not imply the activity duration but either the remaining life of an asset or the contract hours.

- The model's demand signal in PSS is based on the asset operating schedule. In other words, contracted assets are retrieved back to customers every time the contract period starts and can be moved elsewhere when the period ends.

- Assets always remain in the system, unless they reach the end of their life and are not recyclable. In other words, the assets are not shipped out from the models.

Secondly, DES is inadequate to model PSS offering since:

- The assets are a moving object, whilst the OEM's service facility and the customer's facility are fixed in the DES models. Therefore, it is not natural to describe the cases that in-service assets are used by contracted customers at the OEM's facility, for example, a case of machine tool reported by Azarenko et al. (2009). The assets in this case are more appropriate to be fixed whilst the customers should move in the models.

- DES is a system modelling approach, thus, the decision structure in the models is controlled by one system. In other words, it is not appropriate to address the decentralised decision making. For this reason, DES lacks the capability in exposing the fact that in-service assets are autonomous and cannot be managed by the OEM.

- Assets in DES are passive and their attributes (e.g. MTBF, usage) are hidden inside the objects. Thereby, asset life cycle could not be explicitly exposed.

- Though asset performance triggered OEM's service process in the models, the actions could not be activated by the assets but by the 'Delay' element. This means either the element must keep monitoring asset status to perform the triggers (which can cause some signal delays), or the action must be predetermined at the 'Delay' element in advance of the actual event. In the first case, even though the delays could have been minimised by reducing the monitoring interval, the speed of the models would be tremendously slow as a

result. On the other hand, the second approach requires extensive programming. In this study, the first approach was applied to the third usage scenario to capture the multi-level usage and multi-period contract requirements.

Lastly, in comparison with existing techniques in the literature, this study was able to capture the influence of product performance on customers-OEM relationship, 'value' parameters (agreed availability), as well as service efficiency (actual availability, missed hours). Nonetheless, several factors need to be dropped from the models to avoid extensive complexity, for example, redesigns from customer's involvements, effect of monitoring technology, and connections between asset transformation and service support.

4.2.3 Business case III: Hybrid SD-ABS technique

The previous two studies explored the capability in using single simulation techniques within the PSS offering context. The results revealed the capability to describe PSS business, yet, a few characteristics could not be effectively incorporated. Therefore, this study employed ABS to cope with the inadequacies, using an aero-engine as the modelling domain. Model description and programming code is detailed Appendix D.

This study revealed the capability of the technique in modelling the PSS offering decisions via the embedded 'value-in-use' parameter (agreed turnaround time), service efficiency measure (average delay, average availability), and the input's uncertainties. It could also expose asset life cycle, their autonomy in operating independently from the OEM, and their active nature in initiating OEM service process with no additional workaround. The assets could remain in the system as well as being disposed, depending on the state definition.

Moreover, the hybrid technique enabled interactive changes of inputs during model execution. This implies that redesigns from customer involvement can possibly be modelled by this technique. The multi-layer design suggested that decentralised decision making may potentially be exposed. The engine-OEM relationship depicts the potential in representing interactions between stakeholders in the whole network, including OEM-customer, as well as influences among the engines themselves. The interactive capability in combination with the interaction feature may enable the connection between asset transformation and service support to be explored. These possibilities were investigated in business case IV below.

Despite the numerous potential benefits, the hybrid SD-ABS technique has two major drawbacks. First, SD is a continuous modelling approach, thus, OEM workloads appeared as real numbers. This means that the number of engines was shown in decimal places rather than integers. Secondly, the 'stock' variables are aggregated; there was no rule to define queues. As a result, the software may randomly arrange

queuing assets, which in turn, can affect penalty cost. These limitations were also handled in business case IV below.

4.2.4 Business case IV: Hybrid DES-ABS technique

As service contracts have been implemented repeatedly in the aircraft sector, it was chosen as the modelling domain for this study. Based on reported cases in the aircraft sector (Section 2.2), PSS contracts exist in both fleet-basis and asset-basis. The fleet-based contracts (widely adopted in the military sector) guarantee availability of the entire fleet of aircrafts. Differently, aircraft-based contracts focus on turnaround time and are usually more appropriate for commercial aircraft as it is critical to fly according to their flight schedules. Also, the traditional business scenario is included in this study to enable comparison of financial benefits across various offering alternatives. Traditionally, an airline operator carries out maintenance internally or by employing service partners. In both cases, OEMs are not responsible for supporting the sold assets. For these reasons, three business scenarios were defined under aircraft business in this study: traditional scenario, fleet-basis contracts, and aircraft-basis contracts. The modelling approach is further discussed in the next chapter, and the programming code is provided in Appendix E.

The model enables user's interactive adjustment of the following inputs during the model execution:

- Contractual inputs: payments, penalty, turnaround time. This implies that the model enables contract renegotiation after the contract is initiated.

- Market inputs: subsystem price, inventory holding cost per item, average profit from sold flight ticket per flight. This suggests that the model enables the marketing condition to be tailored during contract execution.

- Activity inputs: numbers of OEM and MRO technicians, time to perform each maintenance task, lead time and inventory management decision. This means that the model enables operational policy changes to be captured during contract execution.

Additionally, this hybrid technique is powerful in describing the following requirements:

- Service efficiency can be evaluated in forms of availability, demand satisfaction rate, and delay hours, while financial risks and benefits can be examined from spare cost, penalty, and revenue. The value parameters were represented by turnaround time or fleet availability depending on the scenarios.

- Decentralised decision making can be represented by the hierarchy between OEM service function and OEM inventory management function, and between the central airline operator and MRO function.

- Aircraft and subsystems can be created as agents to represent their individuality and heterogeneity and to be restored in the system throughout the contract. The agents can be modelled using state modelling to expose their life cycles and internal function.

- The model can randomly generate obsolescence and asset failure, thus, their implication to financial benefits can be explored.

- The model enables user's manual adjustments during execution time to modify and terminate contracts, as well as to change operational decisions.

- The model can capture the impacts of asset's performance on OEM-airline relationship by defining communication protocol between the agents. This is also applied to the interconnections between subsystem-aircraft, airline's MRO-central airline, and OEM's service department–OEM's inventory department.

The major drawback of this model is in terms of the complexity that comes from an aircraft's structure. An aircraft is made of several major subsystems (e.g. engines, wings) and their components (e.g. turbine blades) which are all interconnected and influence aircraft performance. During a flight, an accident to the aircraft affects all subsystems and their components whereas a non-functional subsystem deactivates its components but may or may not change aircraft's state. Once a subsystem is broken, it still needs to wait until landing to be repaired. Similarly, even if it is ready, it may not be operated if the other subsystems are not functioning. These interdependencies inside the aircraft agents require extensive coding and verification.

Due to these complexities, redesigns from the customer involvement, and the link between asset transformation and service support have not yet explored. Still, this model could indirectly capture the effect of design improvements in the form of obsolescence.

4.3 Discussion

This section provides a summary of the major capability of each technique versus the strengths and weaknesses in PSS modelling literature analysed in Section 2.5.

Table 4.2: Summary of model's capability

No.	Criteria	SD model	DES model	SD-ABS model	DES-ABS model
1	Generalise to all PSS	Out of scope			
2	Incorporate some analytical techniques				
3	Extend life cycle perspective from product selling	■	■	■	■
4	Address value parameter explicitly		■	■	■
5	Highlight interactions between parties in supply chain and customer.	Out of scope			
6	Include both economic and environmental measures				
7	Demonstrate the link between asset transformation and service support	■			
8	Present service efficiency measures	■	■	■	■
9	Capture link between product performance and customer-manufacturer relationship	■	■		■
10	Illustrate redesigns from customer involvements	■			■
11	Demonstrate decentralise decision making				■
12	Represent cultural mind frame, social habits, and influence between customers	Out of scope			
13	Capture effect of technology on company's capability	■			
14	Incorporate government influence	Out of scope			
15	Embed input uncertainties		■	■	■
16	Explicitly present asset's lifecycle			■	■
17	Expose asset's autonomy			■	■

According to the table, the six out-of-scope areas were excluded from this research as clarified in Section 3.2. All models were capable in presenting the extended asset life cycle beyond product selling and visualising service efficiency outputs, whereas decentralised decision making was only better represented in the hybrid DES-ABS model. Redesigns from customer involvements were encompassed in the SD and hybrid DES-ABS models. The link between asset transformation and service support has only been explored in the SD model as it could increase complexity in other models. Along this line, an asset's autonomy and its life cycle could only be highlighted by incorporating ABS.

Overall, the analysis revealed the capability of incorporating ABS in DES over other techniques. Accordingly, the hybrid technique was chosen as the backbone for the modelling constructs. It should be noted that the capability analysis limit to these

studies, nonetheless, the drawbacks revealed from these model developments were sufficient to direct this research to the hybrid technique.

4.4 Chapter summary

This chapter presented the use of simulation techniques in the PSS context and an analysis of their capability. Their general strengths and drawbacks were examined from the literature (Section 4.1) and actual model developments were conducted in the context of PSS (Section 4.2). Among the selected techniques, a hybrid technique developed from ABS and DES was considered the most appropriate to model PSS offering decision. This outcome accomplishes the first objective of this thesis. The next chapter focuses on the modelling approach using this hybrid technique.

5 Development of the modelling approach

This chapter deals with how to use the hybrid DES-ABS technique to ultimately enhance effectiveness and efficiency of the modelling constructs. This responds to the second objective of this thesis. Section 5.1 presents the implementation of the hybrid techniques in various cases, Section 5.2 discusses the applied modelling approach, Section 5.3 refines the approach for effectiveness and efficiency, and Section 5.4 concludes this chapter.

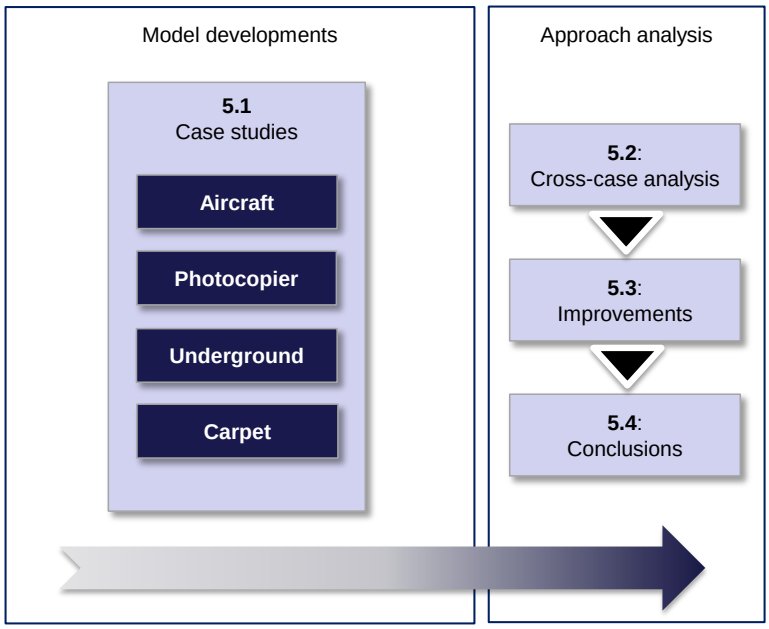


Figure 5-1: Chapter 5 outline

5.1 The hybrid DES-ABS technique in case studies

This section applies the hybrid technique selected from Chapter 4 to various cases so that the lessons learnt in terms of detailed modelling methods in representing PSS characteristics can be captured and generalised for developing the constructs. All models were developed from existing business cases obtained from the literature review in Section 2.2.1 and 2.2.2 which include aircraft, photocopier, underground train, and carpet sectors. The majority of the applied approach is repeatable across cases. *Therefore, this section details the approach for the aircraft case and highlights only the differences in other cases that result in the model variety.* The verification and validation can be summarised as presented in Table 5-1. All models have been verified and validated using at least six methods. The models have not been validated with companies as this stage aims for technical developments and the accuracy of numerical input values in the models is not the key accuracy of this thesis.

Table 5-1: Summary of applied verification and validation methods

Validation method	Case I	Case II	Case III	Case IV
Business case exists	■	■	■	■
Validate concept with PSS experts		■	■	
Breakdown the model into several functions and check one by one	■	■	■	■
Simplify the input value, force extreme condition and observe expected outcomes	■	■	■	■
Manually step through the models and check expected outcomes	■	■	■	■
Verify model with a simulation expert	■	■	■	■
Validate the model with a practitioner				
Document the model and recheck it twice	■	■	■	■
Present the model to an external organisation	■			

5.1.1 Case I: Aircraft

The first model was developed to initially investigate the capability of the hybrid DES-ABS technique in modelling PSS contracts as described in Chapter 4. Three modelling scenarios have been covered in this model; traditional scenario, fleet-contracts, and aircraft-contracts. This chapter presents this model from a modelling approach perspective. Details of the model coding are provided in Appendix E.

The first step recommended in building an ABS model is agent identification (Macal and North, 2010). Driven by the gaps of knowledge and weaknesses identified from the PSS modelling literature, the model consists of the following agents:

- To enable product performance monitoring, aircraft's behaviour should be exposed. Therefore, an aircraft should be modelled as an individual entity. Generally, an aircraft is scheduled for routine maintenance which may be changed to improve contract performance during the delivery phase. For these reasons, **aircraft** was modelled as an agent.

- Aircraft are composed of heterogeneous subsystems (power system, structural system, avionic system) which exhibit different failure patterns. Thus, each

subsystem should be modelled as an individual entity. The OEM and the airline's MRO disassemble and service these subsystems therefore communication is required between subsystems and the parties. For these reason, **subsystem, OEM, and MRO** were modelled as agent.

- Customers tend to monitor OEM performance and renegotiate or continue a contract based on that performance. Thus, **customer** needs to be highly adaptive, hence, an agent.

- Design changes can take place with particular spare parts therefore the OEM's **part stock** was modelled as an agent.

These resulted in six agents in total, summarised in Figure 5-2.

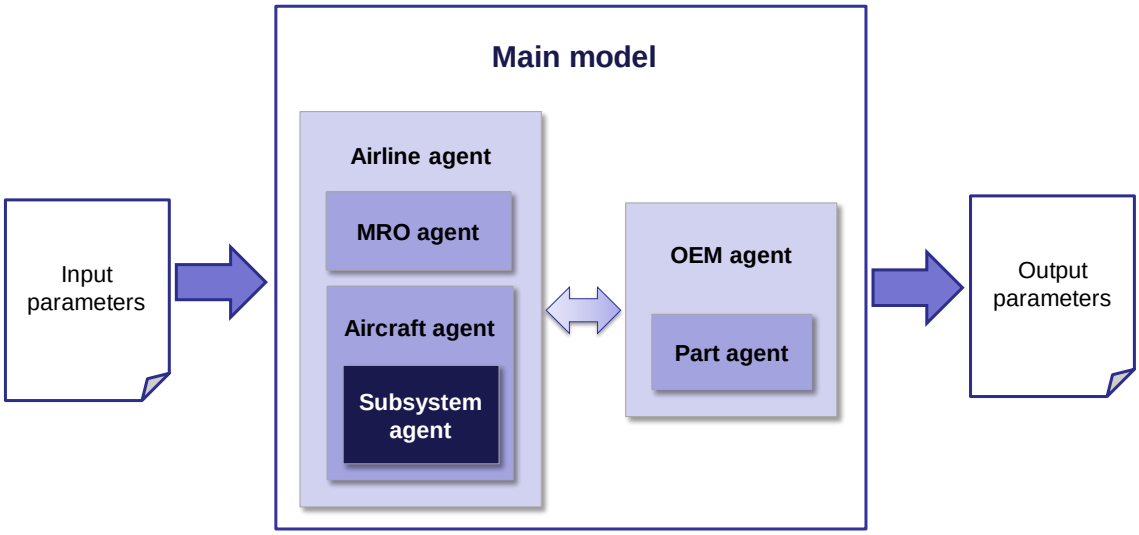


Figure 5-2: The aircraft contract model structure

Figure 5-2 illustrates the structure of a multilayer model with the input and output parameters. The model presents a hierarchical structure of agents. The top layer represents Airline agent and OEM agent as the two main actors in the simulation environment where aircraft operation, maintenance activities, and business transactions take place.

Agents have behaviour (i.e. anything that an agent does), for instance, the OEM agent performs aircraft maintenance, the Airline agent monitors contract performances, etc. An agent may encapsulate other agents; for instance, the Airline agent encapsulates MRO agent and Aircraft agents. In this case, the MRO agent illustrates the maintenance function carried out by the airline exclusively from the normal aircraft operations. Similarly, within the OEM agent, the Part agent illustrates the spare part stock function performed by the OEM in addition to the usual maintenance function carried out by the OEM.

Input parameters

To a large extent, the input parameters are led by the report cases which include maintenance cycle, life-time of the assets, required availability level, penalty charges etc. These parameters were assigned to the Airline, OEM, Aircraft and Subsystem agents in correspondence with the modelling scenarios, and consequently one parameter can be used by more than one agent. Examples of input parameters related to the OEM agent include service cycle time, number of technicians, obsolescence rate etc. Parameters linked with the Airline agent cover the required fleet availability, contract price, etc. Typical input parameters for the Aircraft agent are product family, maintenance cycle, turnaround time etc. Finally, life time, Mean Time Between Failures (MTBF), and spare part costs are input parameters for the Subsystem agent (e.g. engines). The full set of input parameters are described in Table 5-2. The empty cells imply the absent parameters in the model subject to the particular modelling scenario.

Table 5-2: Summary of input parameters

Model	Input parameter	Parameter description	Scenarios *		
			SC1	SC2	SC3
Main	Spare cost	Cost of stocking one unit of spare part	■	■	■
Aircraft	Product family	Different designs of aircraft	■	■	■
	Maintenance cycle	Flight hours between maintenance schedules	■	■	■
	Emergency rate	Frequency of random events that interrupt flight operations	■	■	■
	Number of subsystems	Number of the subsystems in an aircraft	■	■	■
	Contract price	The monthly payment that the OEM receives to sustain aircraft's capability			■
	Penalty charge	The payment incurred to the OEM for failure to achieve the required performance			■
	Turnaround time	Duration that an aircraft is not ready for operation			■
Subsystem	Asset price	The price for buying the subsystem	■		
	Lifetime	Flight hours that the subsystem is capable of operating	■	■	■
	MTBF	Flight hours between the subsystem's failures	■	■	■
OEM	Number of technicians	Number of maintenance staff		■	■
	Obsolescence rate	Frequency of product design changes		■	■
	Service cycle time	Duration that maintenance staff takes to perform service		■	■
	Reorder interval	Duration in which the stock level is regularly monitored		■	■

Model	Input	Parameter description	Scenarios *		
	Reorder quantity	The quantity of spares to be refilled		■	■
	Base level	The stock level at which a replenishment is suggested		■	■
Airline	Loss of opportunity cost	Expenses incurred to an airline due to delays	■		
	Number of technicians	Number of maintenance staff	■		
	Skill	Relative skill of airline's technicians to OEM's technician	■		
	Average profit per flight	Profits an airline generally generates each flight	■	■	■
	Reorder level	The stock level at which stock needs to be refilled	■		
	Reorder quantity	The quantity of spares to be refilled	■		
	Delivery lead time	Time waiting for spares to be delivered	■		
	Required availability	Percentage of time the aircrafts are required to be capable for operation		■	
	Contract price	Monthly payment of a contract an airline make to the OEM		■	
	Penalty charge	Charges incurred to OEM for failure to achieve the required performance		■	
		* **SC1 = Traditional scenario** **SC2 = Fleet-based contract** **SC3 = Aircraft-based contract**			

The agent-based model

The **main model** (Figure 5-3) represents the top layer simulation environment comprising an OEM and several airlines. This approach implies that these agents are independent and can exist without one another, yet, interact under the same environment.

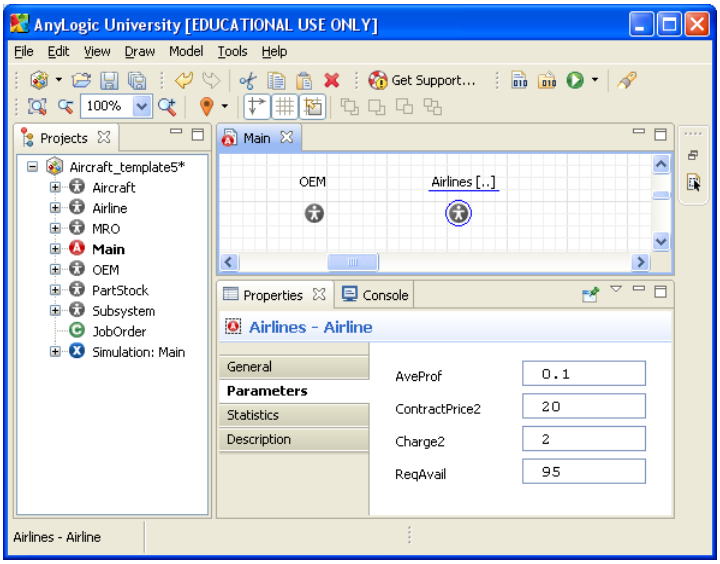


Figure 5-3: Main model

Airline agent (Figure 5-4) monitors financial benefits and operational performances of the three business scenarios. Availability can be described as the fraction of available aircraft in the fleet against the requirements and the payment depends on available aircraft, mean time between critical failures, and demand satisfaction rate (Richardson and Jacopino, 2006). Besides the monitoring function, the Airline agent also encapsulates the Aircraft and MRO agents, which implies that the airlines can manipulate aircraft operations and MRO activity separately from one another. Fleet-contract requirements can be amended inside the Airline agent.

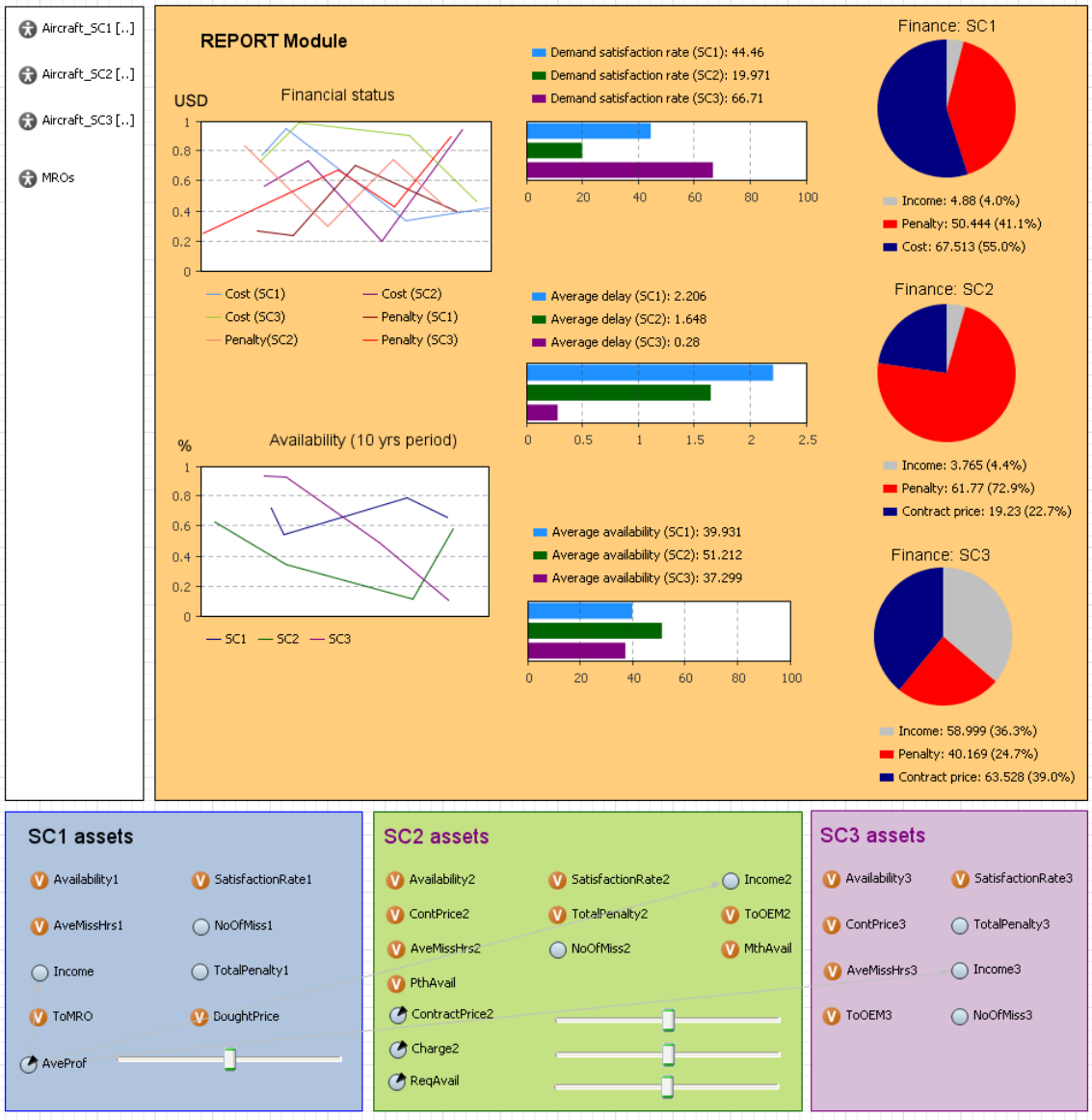


Figure 5-4: Airline agent

The airline's **MRO** carries out maintenance and inventory management activities under the traditional scenario. Thus, it represents service processes and stock condition. Generally, three levels of scheduled maintenance are performed (Pall, 2008). These are also known as the A, B and C checks, and reported to take place after 50, 100 and 600 flight hours (Bazargan and McGrath, 2006). In addition to scheduled maintenance, unplanned maintenance can take place randomly. Therefore, four lines of service processes are modelled (Figure 5-5). Regarding the inventory management function, the model is simplified by having only three aircraft families which result in three types of spares. The three state charts describe states of each spare stock. An order is taken place once the stock level falls below the defined level.

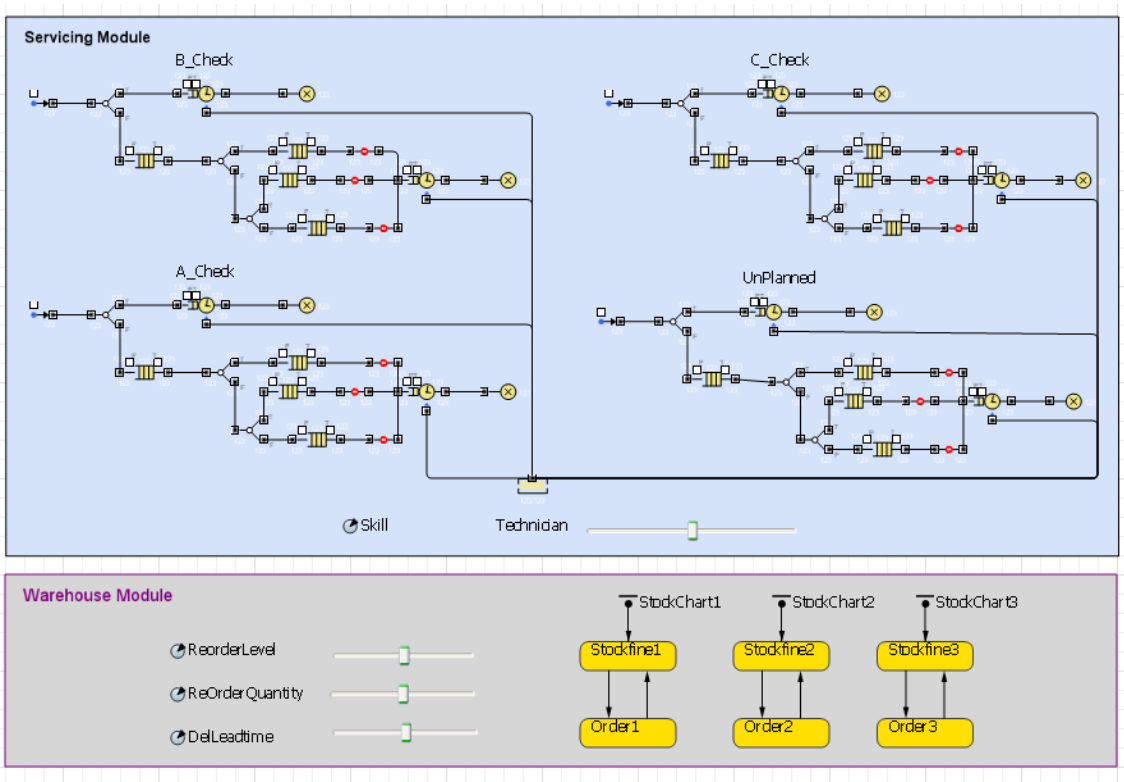


Figure 5-5: MRO agent

The **Aircraft** agent can possess operating state (*Fly*) and non-operating state (*Maintenance*), as shown in Figure 5-6. This is governed by the condition of its subsystems, maintenance schedule, and external emergencies. Emergency landing can directly be a major risk to the OEMs in delivering contracts. For this reason, Subsystem agents are embedded, a maintenance cycle (*RoutineCheck*) is defined, and an emergent rate is encompassed inside this agent. Furthermore, inputs for the aircraft-contract are incorporated within this agent.

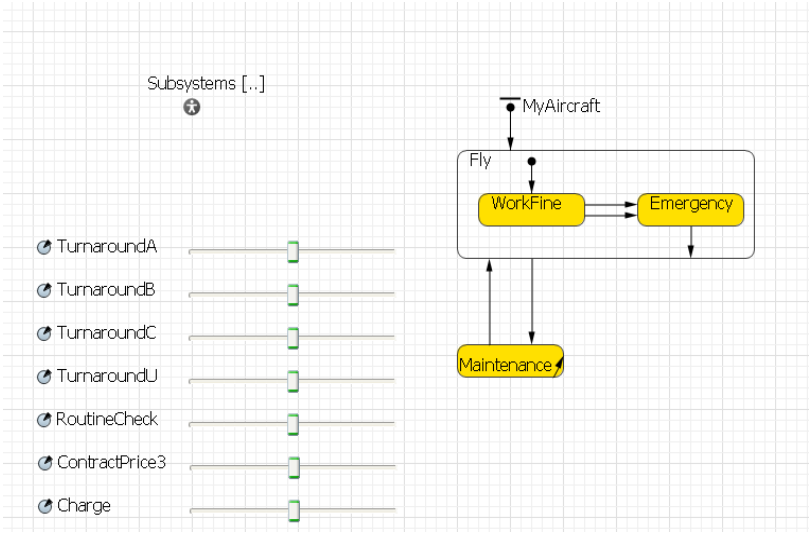


Figure 5-6: Aircraft agent

The **Subsystem**'s health defines the customer-OEM relationship, thus, two major states (Figure 5-7) refer to subsystem authorisation by the customer (*CanWork*) and OEM (*Maintenance*). If the subsystem is with the customer, actual asset usage must be recorded because it affects the aircraft's health directly. Accordingly, the state when a subsystem is operating (*Working*) must be separated from when it is not operating. Similar to the Aircraft agent, an external event can stop the subsystem operation (e.g. a bird strike on an engine), represented by *Stop*. Besides, a subsystem may need to wait for landing or wait for others to be assembled, and may be scraped after some time, depicted as *Waiting* and *Dead* respectively.

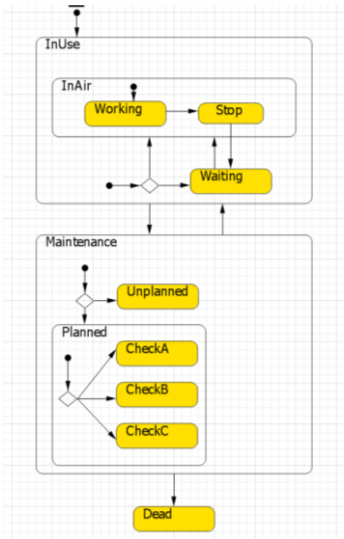


Figure 5-7: Subsystem agent

Similar to the MRO agent, the OEM agent illustrates service processes, and stock management. The OEM manages stock separately from service operations, thus, Part agents are encapsulated inside the OEM layer. The three Part agents correspond to the three types of spare parts. In addition to the two functions, design change can take place, thus, it is embedded inside the OEM agent. Once this happens, the agent sends a message to the obsolete Part agent. The OEM can also analyse financial benefits from the contracts. Therefore, the resulting OEM model was developed, as represented in Figure 5-8.

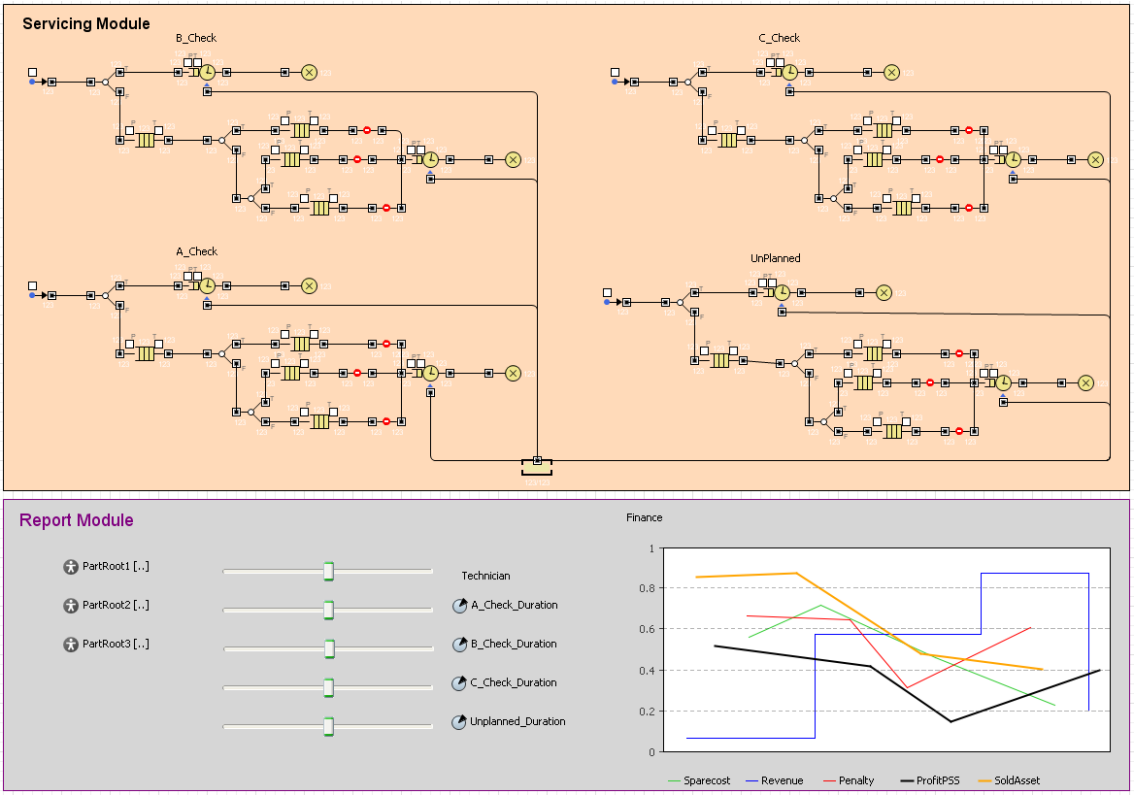


Figure 5-8: OEM agent

There are two important aspects addressed in the **Part agent**: stocking policy and obsolescence. Stocking policy affects the time to recover an aircraft whereas obsolescence can reflect redesigns and also the recovery period. Therefore, the agent was modelled as presented in Figure 5-9.

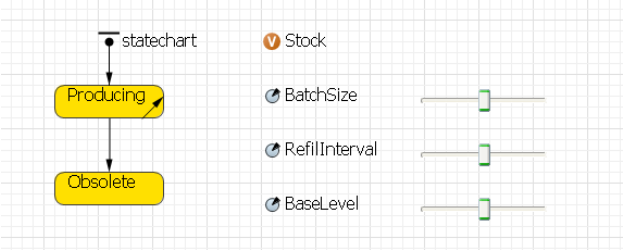


Figure 5-9: Part agent

A Java object is initiated from the Subsystem agent to the MRO agent (in the traditional scenario) or to OEM agent (in contracting scenarios) to provide asset information and specified services prior to servicing.

Output parameters

Output parameters refer to performance measures related to demand satisfaction rate, average missed hours, spare cost, penalty, revenue, and availability. The formulas to calculate these measures have been derived from many PSS cases, e.g. Harding and Watts (2000), Xerox (2010c), EMSA (2006) etc, as well as from PSS experts. The formulas are hypothetical to prove the concept and can be adjusted if necessary: (therefore) they are not directly associated with the modelling approach. These details, as well as examples of experimentation are attached in Appendix E.

5.1.2 Case II: Photocopier

In this case, the modelling scenario is based on the DocuCare contract (Xerox, 2010c). Service information was collected from the Xerox case study reported by Watson et al. (1998), and refined with researchers involved in Xerox service contract.

Xerox stocks and supplies consumables and parts for customers on their sites and has proactive asset management software to monitor asset condition in real time. These assets are contracted for specific period such as between 9 am – 5 pm. The key measures deal with the percentage of services responded within a one hour period, equipment uptime, and technical response time. All the time dimensions are calculated from the point of receiving notification from customers. Uptime is monitored monthly, and considered against contract hours.

The conceptual and detailed models can be presented in Figure 5-10 and 5-11 respectively. These are based on the modelling approach for the aircraft case with some amendments due to the following differences from the aircraft sector:

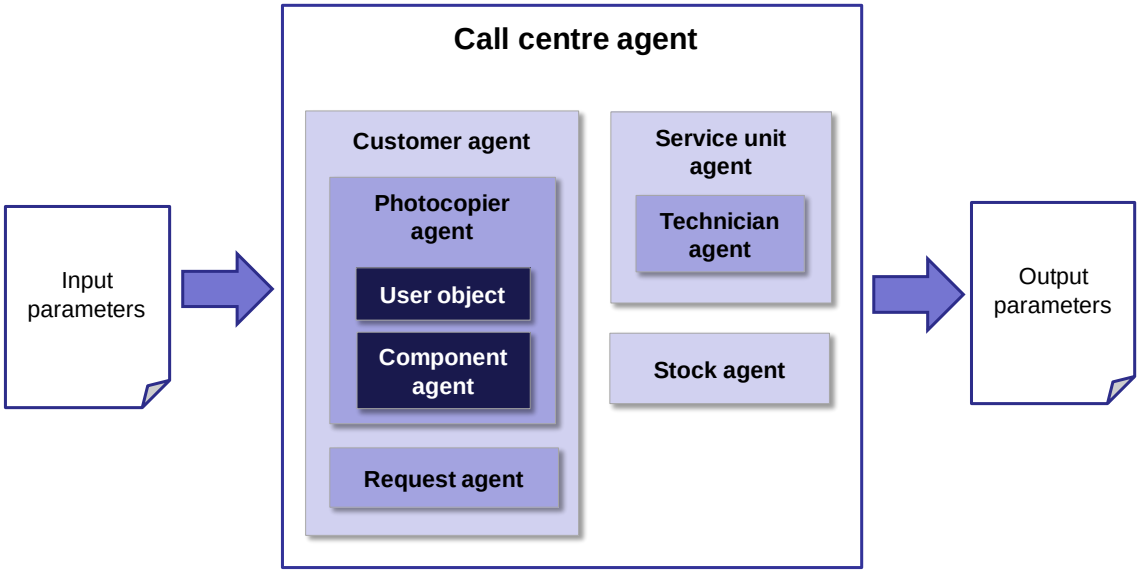


Figure 5-10: The photocopier contract model structure

- Unlike the aircraft sector, field service is common and arranged by local service unit in the photocopy (or printing) sector. Therefore, **service technician** and **local service units** should be created as individual entities. Besides, the two entities need to communicate with one another and with the inventory function, thus they were created as agents.

  At the service unit, the job due in one hour is first allocated to an idle technician. If there is no such a job, the technician can pick the closest to his location. A technician works according to the shifts and it is assumed that an ongoing task will be continued until the completion.

- The OEM-customer relationship is not only governed by an asset's health, but also other services such as technical advice and training provided by a **call centre** and service units. This communication with the call centre suggests that it should be modelled as agent. On top of that, the OEM manages the assets throughout their life cycle and all functions are primarily handled centrally by the call centre. Therefore, the call centre was modelled as an agent and acts as the main model.

  The call centre (**Main model***)* forwards the request to the local service unit if a site visit is necessary. However, in the case that a part replacement is also required, the call centre forwards the job to the inventory department (i.e. **Stock** agent) first. The service unit will be notified only if there is available part in stock.

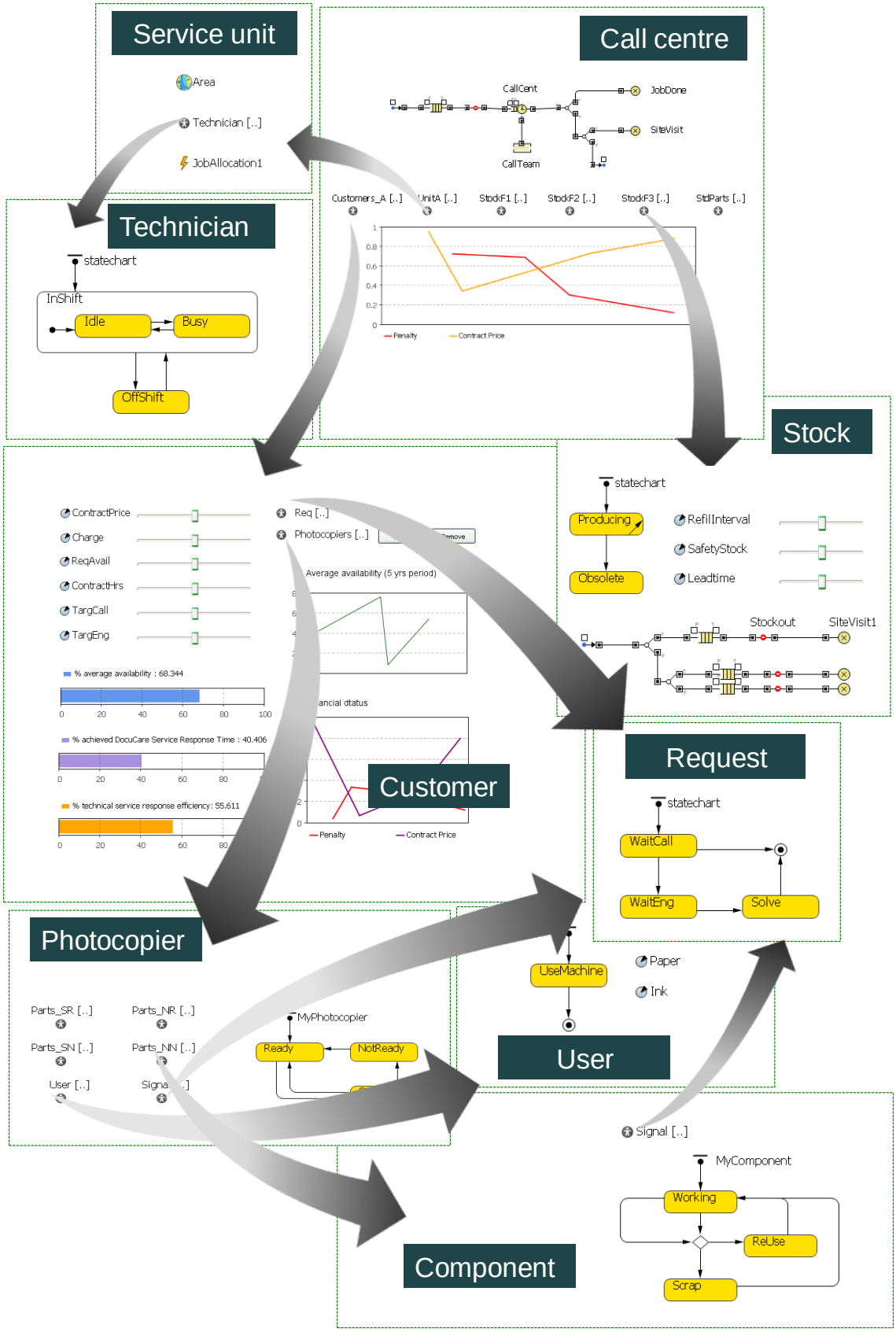


Figure 5-11: The photocopier service contract model

- A response from the OEM is monitored twice – one after being handled by the call centre and the other by a service technician. Besides, there are different forms of enquiry. For these reason, **Reques**t agent was created to establish interactions and record these differences.

  Within the Request agent, job progress is modelled using state modelling. Once the Request is initiated (by *customer*, *photocopier*, or *components)*, a Java object which contains the request information is sent to the call centre to be proceeded. Four types of requests were modelled: general enquiry from customers solved by called, general enquiry from customers which requires a site visit, misuse signals sent by photocopiers and solved by call, and component replacements.

- The utilisation of a photocopier is not predefined, unlike a flight schedule. Therefore, the usage depends on the end users. Each usage also affects levels of paper and ink. A simple way to illustrate the usage and its impacts is to model a **user** as an individual object containing the ink and paper required.

- Hardware upgrade in the printing sector does not tend to be included in a service contract, unless a malfunction is detected. Therefore, obsolescence only occurs in a software component. As a result, the model separates different types of components within the **Photocopier agent**. Similarly, emergent events rarely happen with photocopiers. Therefore, its presence is not necessary.

- In the printing industry, remanufacture/recycle/recondition of components (or subsystem in aircraft) can take place in addition to maintenance. This means the after-life state should be modelled within the **Component agent** and also linked to OEM's stock function. However, some components are disposed. Thus, each component must be defined whether it is recyclable or not.

  Additionally, all components must be working for the photocopier to be functional; therefore the idle sub-state of the component is not necessary to model as in the aircraft case.

  Similarly, there is no predefined level of maintenance. Hence, maintenance services do not have to be segregated. Therefore, it was not necessary modelled a sub-state inside maintenance state of the Component agent.

The model code is provided in Appendix F.

5.1.3 Case III: Underground

The case for underground train service contracts was based on the northern line service contracts between Alstom and London Underground (Harding and Watts,

2000). The data was also cross-checked with the PSS expert who conducted a case study with the company.

Generally, the financial organisation named as ROSCO (e.g. HSBC bank) buys assets from the OEMs and leases it to train operators. The operators may then sign service contracts with the OEMs to sustain a train's operation. In the Northern Line service contract, Alstom guarantees an agreed level of fleet availability to London Underground Ltd (LUL). The availability is considered against contract hours. LUL takes the trains every morning and returns to Alstom at night. The contract covers repair services and cleaning of trains and associated trackside equipment, which can happen at two depots. The key performance parameters involve a guaranteed number of trains in peak service, reliability in service and the management of depot stocks, which are all linked to Lost Customer Hours (LCH). This LCH is a metric for calculating penalty and can vary from one occasion to another.

At a high level, this case differs from the other two cases as follows:

- There is no call centre involved in this case. Thereby, the main model can represent the OEM's maintenance. In effect, Call centre and Service unit agents were excluded.

- Similarly, upgrading service is outside the OEM's responsibility, and stocks can be differentiated using an array. Thus, Part agent is not necessary in the model.

- The operator only operates on a contracting scenario, hence, no MRO agent as in the aircraft model is needed.

- The operator is not interested in the job progress as long as the agreed availability is achieved hence no Request agent is needed.

Consequently, the model structure of this case is presented in Figure 5-12.

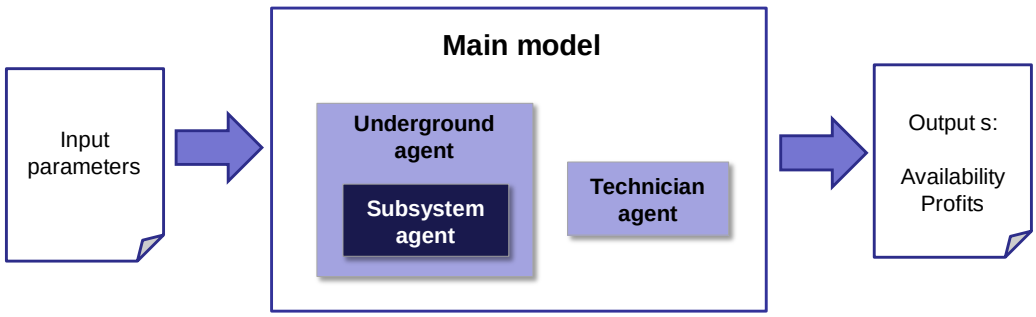


Figure 5-12: The underground contract model structure

The underground train service contract model was developed as shown in Figure 5-13.

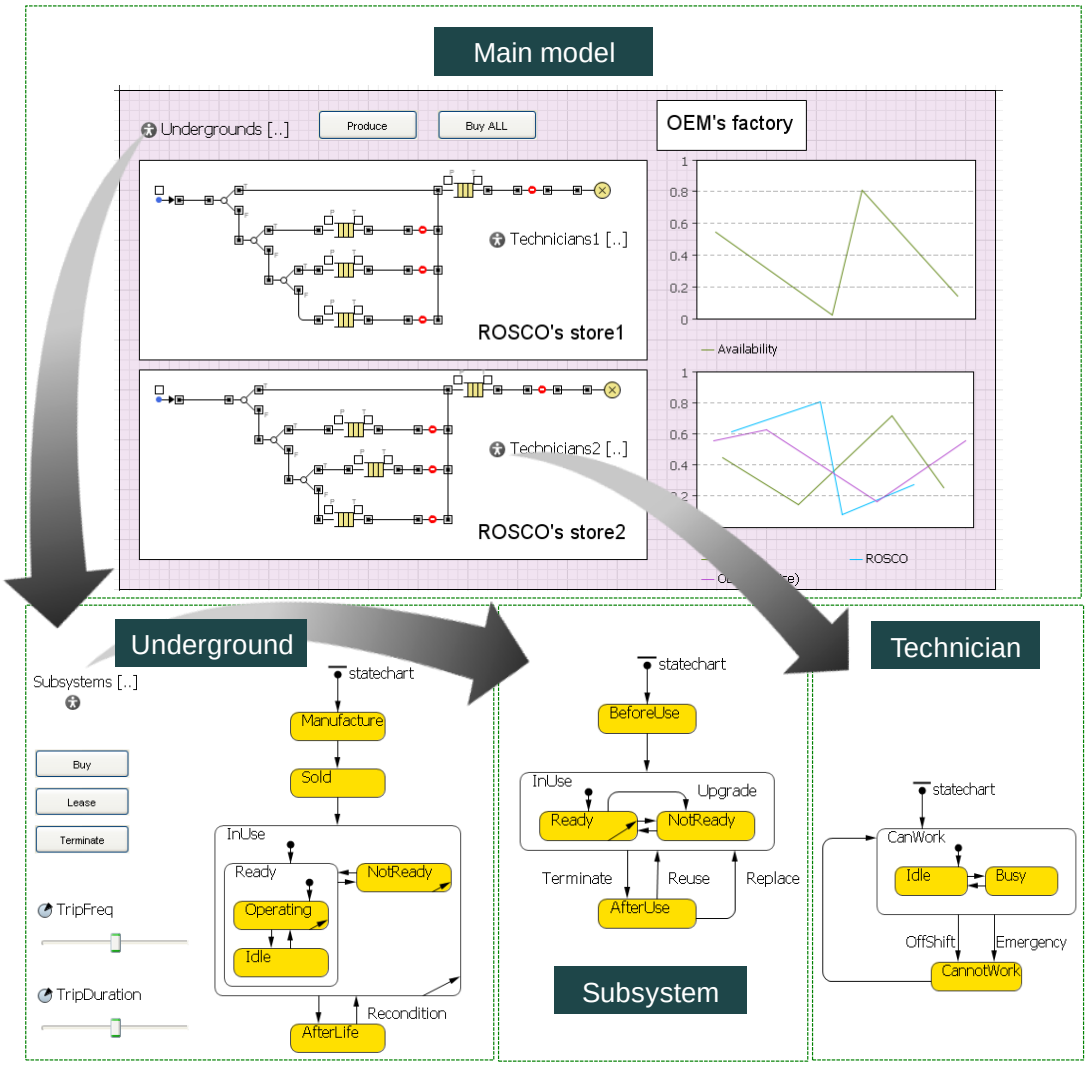


Figure 5-13: Underground train service contract model

According to Figure 5-13, **the main model** presents OEM's services happening at the two depots owned by ROSCO. The four lines refer to the jobs that required no part replacements and the jobs required three different parts. Service technicians are located on both sites to perform these jobs.

Also at the depots, spare parts are stocked and supplied by the OEM's factory when 1) there is a job in the system, 2) the required part is unavailable at the depot, and 3) the required part is available at the OEM's plant. Availability and cash flow related to this contract between the OEM, ROSCO, and the operator are monitored monthly.

Underneath the main layer are **Underground** and **Technician agents**. The underground trains are manufactured by Alstom, sold to ROSCO, leased by LUL and terminated by LUL or ROSCO. This model provides user interactive input for buying, leasing, and termination, using buttons and represented in the state chart. During the use-phase, an **Underground agent** can be ready for operation or not ready. If it is ready, it can be operating or waiting for operation. This mechanism is governed by trip frequency and duration which can be adjusted by users during model execution. In the case of being not ready, the OEM is in charge of asset recovery. If the agreed recovery period is reached, an inner transition is activated to penalise the OEM. The agents can be terminated and fall into the after-life phase. Nonetheless, it can be reconditioned and used again.

A train's health is dependent on major subsystems which also have a life cycle (before-use, being used, and after-use). A **Subsystem agent** can also be ready or not ready for operations. The agent may not be ready due to routine check, breakdown, or being upgraded. If the train is terminated, these subsystems can be reused directly or replaced with a new one.

In terms of the **Technician agents**, this model incorporates emergent events that can happen with staff. A Technician agent cannot work outside the shift period or due to an emergency. Otherwise, the agent is either busy or idle, depending on whether a job order is received or not.

Model code is provided in Appendix G.

5.1.4 Case IV: Carpet

The underlying concepts behind the carpet model can be summarised as follows:

- Unlike other previous products, carpets are not assembled from several modules but considered as one unit after being produced. Accordingly, Component agents can be discarded.

- It is not necessary to separate Customer agents from Carpet agents. One Carpet agent can be treated as one Customer agent.

- Services such as installation, cut-out or repair of the damaged part, and disposal tend to be carried out by the central plant. Additionally, routine cleanings are likely to be assigned to local staff just after making a new contract. Therefore, a Service unit agent has a role to assign jobs to staff and supply consumables. It should be noted that the staff are not technicians, thus, the service staff in this model is named as Staff agent.

- A carpet is changed after some time regardless of the condition, unless some random events cause damage which require cut out and replacement. Once

this happens, the centre arranges transportation for collection and installation. Therefore, recycling is also a key function in the model.

For these reason, the model structure and screenshots were developed as shown in Figure 5-14 and 5-15 respectively.

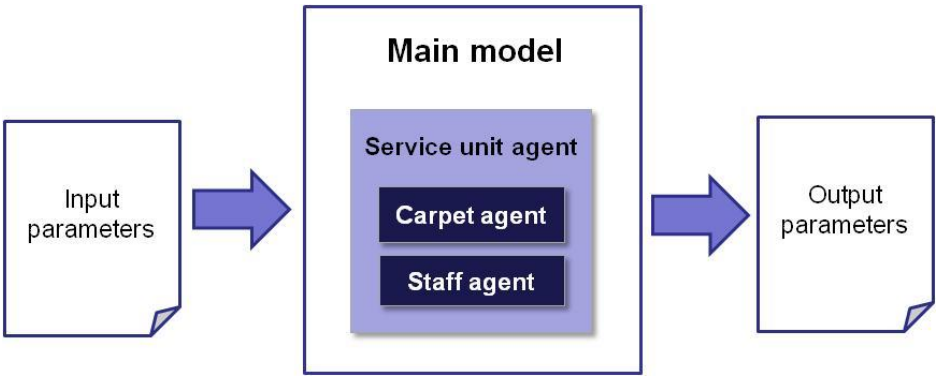


Figure 5-14: The carpet contract model structure

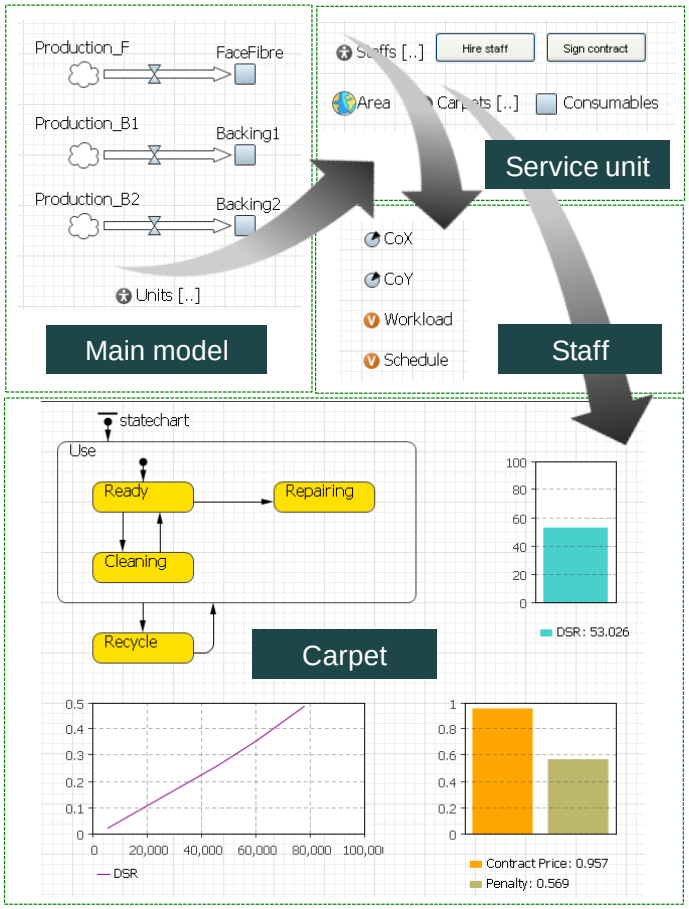


Figure 5-15: Carpet service contract model

As a carpet can be made from a specific combination of face fibre and two backings, the **main model** (Figure 5-15) illustrates production of carpets from these contents using SD.

Underneath the main model layer is **Service unit agents** at which staff allocation to a particular carpet takes place. This process happens after a contract is signed and staff is hired, manually initiated by users. A business unit also stores some consumables for cleaning.

In terms of cleaning staff**,** a record of their location, cleaning schedule and their workload is kept inside the **Staff agent**.

Once a contract is signed, a **Carpet agent** is activated to the use-phase. During this phase, a carpet can be cleaned by local staff, repaired by the central OEM, or be in a ready state. Besides being repaired, the central OEM is responsible to recycle the carpet if it has been used for a while. The OEM is charged if a carpet's replacement can not be made within the agreed period. Performance and payment are updated monthly.

Model code is provided in Appendix H.

5.2 Cross case discussion

This section summarises the approach used to develop the four models.

5.2.1 Summary of the applied modelling approach

At a high level, the approach attempts to capture the common elements in a PSS contract. These elements include asset characteristics, OEM's service processes, relationships between OEMs-customer, and customer requirements, which have been modelled in all cases as follows:

In terms of **asset characteristics**, an asset's health directly influences success or failure in delivering the required capability in the contract. The models apply state modelling to the asset life cycle to illustrate this health which is covered in the in-service and afterlife phases (It should be noted that an asset can be a product as well as its subsystem and the term 'subsystem' is used interchangeably with 'component' in this thesis depending on the sector). The in-service phase included operating, idle, and under different services performed by OEM, while the afterlife phase incorporated recycling and disposal. The actual asset usage was recorded on leaving the operating state. The change in states inside the state chart triggers the events and interactions in the models.

The **OEM's service process** implies the capability to sustain a contract. The structure of these service processes outlines OEM agent behaviour. The OEM central function was modelled using DES, where working shift is controlled by 'Hold' elements, whereas the staff agent (named as Technician agent or Staff agent in the model) is modelled with state modelling to illustrate his current working state. Furthermore, the structure of these service processes could also be linked to asset's state-chart, hence the agent model. For example in the aircraft context, service offerings are centred around maintenance and different levels of maintenance result in sub-states inside Subsystem agent.

With regards to **relationships between OEMs-customer**, the OEM's commitment and relationship with customers highlights agent interactions. The OEM is obliged to take care of the contracted assets, thus, this relationship needs to link with asset agents. The models capture this issue by establishing a communication protocol between asset and OEM agents.

Customer requirements embed stochastic dynamic behaviour in the contract performance as this demand can change over time, influenced by factors such as market conditions and willingness to pay. Therefore, contract modification was enabled manually during model execution in terms of the number of contracted assets, maximum asset's unavailable duration, and the penalty cost. External risks were captured at different levels of assets (product, component), either randomly (e.g. accident rate) or statistically (e.g. MTBF).

The requirements can also be linked to contract performance monitoring which can be described by output parameters and were related to measure settings and calculations in the models. The contract-related parameters were presented in customer agents, monitored regularly using a synchronous programming method. The parameters exhibited the following variations across contracts:

- Unit of contract: fleet level (aircraft fleet, a system of printers and photocopiers, Northern Line undergrounds), platform level (aircraft, photocopier, carpet, underground), component level (aero-engine)

- Penalty structure (time basis, level basis, both)

- Payment structure (monthly basis, single use)

At a detailed level, the approach incorporates several methods to model agent behaviours and interactions. These can be summarised as follows:

- Asset information was encapsulated as a Java object and passed between asset agents and OEM agents (and the MRO agent in case of aircraft) during servicing activities. The message passing method was preferably adopted as a trigger rather than a variable trigger method to reduce the number of variables.

- The jobs allocated to technicians were modelled using an asynchronous modelling method. In other words, the function is activated when an OEM agent receives the information Java object.

- Stock-out was presented using DES by a 'Hold' element inside an OEM agent or as a Part agent, triggered by an 'Event' element.

- Obsolescence was generated from an OEM agent and triggers the state chart inside a Part agent.

- A Part agent was replicated and assigned different attributes to demonstrate different types of spares.

- Applying state modelling inside a Request agent allows job progress to be monitored. The type of the request (general enquiry, asset breakdown etc.) was distinguished by the agent's attribute.

- Inputs could be presented in three ways, named in this thesis as user interactive inputs, independent inputs, and embedded inputs. Interactive inputs can be used by users who are not familiar with the models to adjust the input values within a defined range. If these adjustments take places during model execution, it can imply contract renegotiation. Independent inputs can be applied to important parameters, defined by modellers. This method allows modellers to change input to any value easily. Finally, embedded inputs are defined in the model logic and not presented explicitly in the models. This was applied when the input is required but its value does not normally change.

- A component agent could be modelled using state modelling or SD depending on the nature of products. In case of the carpet sector, the components are continuous entities and recycling can be the key service. Therefore, the carpet model applies SD to capture carpet production/recycling processes.

The next section examines this approach against the current state of PSS modelling literature in terms of capability in describing PSS characteristics and dynamic behaviour.

5.2.2 Assessment against strengths and weaknesses in PSS modelling literature

Table 5-3: Summary of the model's capability

No.	Criteria	SD model	DES model	SD-ABS model	DES-ABS model
1	Generalise to all PSS	Out of scope			
2	Incorporate some analytical techniques				
3	Extend life cycle perspective from product selling	■	■	■	■
4	Address value parameter explicitly	■	■	■	■
5	Highlight interactions between parties in supply chain and customer.	Out of scope			
6	Include both economic and environmental measures				
7	Demonstrate the link between asset transformation and service support				
8	Present service efficiency measures	■	■	■	■
9	Capture link between product performance and customer-manufacturer relationship	■	■	■	■
10	Illustrate redesigns from customer involvements	■		■	
11	Demonstrate decentralise decision making	■	■	■	■
12	Represent cultural mind frame, social habits, and influence between customers	Out of scope			
13	Capture effect of technology on company's capability				
14	Incorporate government influence	Out of scope			
15	Embed input uncertainties	■	■	■	■
16	Explicitly present asset's lifecycle	■	■	■	■
17	Expose asset's autonomy	■	■	■	■

The six out-of-scope areas and the extended life cycle perspective were excluded from the scope of this research. There are seven areas that have been demonstrated in all models.

1. Value parameters can be clearly defined using input parameters such as required availability (aircraft, photocopier, underground train, carpet), target respond time (photocopier), and target recovery time (aircraft, carpet).

2. Service efficiency measures can be presented using output variables which are linked to the 'value' parameters, such as demand satisfaction rate (carpet,

photocopier, aircraft), average delay (aircraft), and availability (underground train, photocopier, aircraft).

3. All models enable influence of product performance on OEM-customer relationship by linking asset state chart with OEM agent.

4. All models can represent decentralised decision making, for example, MRO (aircraft) and servicing staff (carpet, underground train, photocopier) by representing them as individual autonomous agents.

5. All models can encapsulate input uncertainties using input distribution.

6. Asset life cycle can be exposed in the asset's state chart.

7. Asset autonomy can be highlighted by creating it as an individual autonomous agent.

Nevertheless, effects of technology on OEM's capability have not yet entailed in any model to avoid extensive complexity.

Along this line, the link between asset transformation and service support can be illustrated in the models via asset recycle in the photocopier, train, and carpet cases. Redesigns from customer involvements can be captured as a random obsolescence event in case of aircraft and underground train sectors.

It can be seen that the applied approach contributes to PSS modelling knowledge significantly. However, there is also a need to refine this approach to enhance efficiency in building models.

5.2.3 Complexity analysis

The purpose of this section is to clarify complexity that can be handled in the scope of this research, so that an efficient modelling approach can be defined prior to the formation of modelling structures. Based on the model developments, the complexities in the models can be related to the technique itself, the nature of problems, and the approach.

The complexities from the technique were mainly caused by two reasons. First, ABS is still an emerging technique in modelling products and services. Software support is still limited in relation to DES and SD. Therefore, the models heavily rely on extensive use of programming code. Second, ABS is a bottom-up modelling approach which enables modellers to focus on one agent at a time (Grimm et al., 2005; Bonabeau, 2002; Macy 2002; Jennings, 2001). When it is applied to the PSS context, which involves many actors and where the overall system behaviour is not entirely emergent from individual rules, the author could lose sight of the 'big picture' easily. Eventually, the

models become over complicated. These complexities are technical and out of this research's scope.

The complexities caused by the problem mainly relate to contracted assets and service activities. In terms of assets, the contracts can be based on the product itself (e.g. aircraft, photocopy) or the entire fleet. The scope may include spares support (e.g. turbine blades) in which a spare replacement can incur significant costs and each spare can influence one another. Besides, the product's performance can rely on the overall condition of subsystems but may not be influenced by single subsystems (e.g. aero-engine). Therefore, all assets (aircraft, engines, blades) could be agents and their interdependencies were difficult to model.

With regards to service activity, decentralised decision making increased the number of individual entities. Thus, job allocation and message passing mechanism must be defined. Furthermore, services can be performed differently depending on the cause of problem and can take a different length of time. Therefore, each service must be treated separately. As shown in the photocopier case, the signal for replacement service was sent by the Component agent, the signal of misuse was activated by the Photocopier agent, and the signal for general enquiry was initiated from the Customer agent. Moreover, recycling and obsolescence influence stock level, thus, exhibit additional interconnections. This type of complexity can be organised, and consequently, it is handled in Section 5.3.1.

Finally, the complexities from the approach resulted from the attempt to present close-to-reality models to resolve the weaknesses identified in PSS modelling literature. Some characteristics and dynamic behaviour may not significantly affect contract performances, yet, it was presented in the models. Furthermore, some agents could have been excluded from the models as their autonomies were not necessary to expose. This type of complexity can also be managed and is handled in Section 5.3.2.

5.3 Managing modelling complexity

This section details the process of the approach's refinement based on the complexity analysis above.

5.3.1 Handling problem-related complexities

As with other complex problems, complexities often arise due to the increase level of detail. Therefore, the first attempt to refine the approach was to classify the details in the four models into common PSS elements, case-dependent PSS elements, and as-is traditional business elements. The results are presented in Table 5-4.

Table 5-4: Analysis of the elements

Common PSS elements	Case-dependent PSS elements	As-is tradition business elements
Asset health and life cycle	Contract modifications	Inventory management
OEM service process	Contract termination	Job allocation algorithm
In-service asset information	Contract creation	Customer service department
Service efficiency measures	Decentralised service decision making	Call centre
Usage as a unit of analysis	Asset structure	
	Track of work-in-progress	
	Adaptive capacity	

With subject to Table 5-4, since the as-is traditional business elements do not highlight the shift towards PSS adoption, they were excluded from the scope of the modelling constructs and will not be further described in this thesis. On the contrary, the common PSS elements exhibit the standard elements of PSS offering models, and those of case-dependent PSS exhibit variation across PSS models. Therefore, the two parts were included.

5.3.2 Handling approach-related complexities

The second improvement concerns the approach-related complexities. In ABS models, the complexities are often closely associated with the number of agents as it may complicate message passing and verification methods. Accordingly, the second dimension for improvements was to reduce the number of agents. To do this, the understanding of agents was revised and the reasons why agents are generally needed in a model were examined from literature.

Bonabeau (2002) clarified that agents can be simple in which their simple interaction rule can assist in problem solving, or they can be sophisticated in which reality can be closely described via complex learning and adaptive rules. Guessoum and Briot (1999) stated that agents differ from objects as they have reasoning ability. Agents can have different behaviour, described as autonomy, proactivity, sociability, and adaptability. Autonomy refers to an agent's ability to perform functions without external intervention. Proactivity describes an agent's ability to respond to the information only according to their goals. Sociability illustrates interactions between agents, and finally, adaptability represents an agent's ability to adjust itself from the current situation. Shehory (2001) highlighted an agent's concurrency ability, in other words, agent can perform several tasks simultaneously. In addition, Macal and North (2010) added that agents are identifiable and situated in an environment. Identifiable can be seen as a description of agents which contains information about its attributes, behaviour rules, decision logics, and so on. This description shows the boundary of an agent, and the size of this description implies the level of detail in a model. With this, modellers can also determine what can be shared between agents. Being situated means an agent

can be interacted by others agents and be able to respond to the environment. It also implies that the agent is able to recognise and distinguish other agents, which indicates a communication protocol.

Therefore, it can be seen that a general conclusion on the difference between agent-oriented and object-oriented views is still lacking in literature. On one hand, agent-oriented philosophy is implemented using object-oriented modelling. Therefore, it can imply that the difference between the two is only because one is a philosophy and the other is an approach. On the other hand, an object is an agent that is always reactive. Thus, an agent has a proactive capability that an object does not have. From the model development in this thesis, an object was often defined as an agent to enable a message passing mechanism. Consequently, the agents in the four models are simple agents as described by Bonabeau (2002).

In a manufacturing application, the agent's interaction functionality was used as a coupling between manufacturing plants and the enterprise to enable hierarchy of resource planning, hence, an estimation of long term earnings (Rabelo et al., 2005). A life cycle cost of a product can be calculated from the resulting interactions between the plant agents and the enterprise agent involved during the in-service phase (Schumann et al., 2011; Yu, 2008). This functionality, in combination with, the agent's heterogeneity was incorporated into the customer agents to examine impacts from different customer movements and their interactions in a café (Robinson, 2010). This study can enable the café's layout to be designed. The same concept was also adopted to design a location for an emergency exit (Borshchev et al., 2004). From a supply chain perspective, a decision rule was defined to each supply chain agent (Schieritz and Globler, 2003). The simulation result indicates the final arrangement of the supply network based on the defined rules. Similarly, an aero-engine overhaul could be rescheduled based on a scheduling algorithm inside the fleet manager agent (Stranjak et al., 2008).

These examples illustrate various use of an agent's functionality. The next step was to refine the approach to capture the PSS elements in Table 5-2. By doing so, it was first assumed that "*The problem can be captured merely in one layer, thus, no agent is needed".* The disapproval to this proposition identified the need to decompose the main model to a series of agent models. The insufficiencies of applying the single techniques were addressed in Chapter 4, which identified the need for an OEM agent and asset agents as fundamental modelling elements. Based on this structure, the characteristics in Table 5-4 were examined to see whether additional agents are needed. This analysis is summarised in Table 5-5.

Table 5-5: Analysis of agents

Requirements	No additional agent is needed?	
	TRUE	FALSE
Asset health and life cycle	■	
OEM service process	■	
In-service asset information	■	
Service efficiency measures	■	
Usage as a unit of analysis	■	
Contract modifications	■	
Contract termination	■	
Contract creation	■	
Decentralised service decision making		■
Asset structure		■
Track of work-in-progress		■
Adaptive capacity	■	

The majority of requirements can be fulfilled based on the fundamental model structure. However, the exception applies to decentralised decision making, asset structure, and the tracking of work-in-progress.

In terms of decentralised decision making, three methods could have been used to capture variation among staff. First, the OEM agent could have been replicated in the main model (which also contains asset agents) and assigned different attributes to represent individual staff. However, this would imply that the central OEM would be absent and visualising the overall OEM's financial evaluation would be difficult. Secondly, a staff agent could have been embedded in the main model which would become an OEM agent. Again, it would mean that an asset can exist only if the OEM exists. This can contradict a contracting scenario where the OEM only assists customers in a specific service (such as depth maintenance). The third option was adopted in this research, which encompasses both staff agents and an OEM agent in addition to the main model. This approach eliminates the drawbacks from the other approaches.

Regarding asset structure, the fact that the contracted asset may or may not be influenced by its components could not be easily performed in one level. Besides, the components may also have their own attributes which can vary significantly among themselves, and the condition of one component may be affected by others. Lastly, as work-in-progress naturally involves a number of states, it cannot be modelled explicitly and individually without an additional agent.

5.3.3 The refined modelling approach

As a result from the two steps, the refined approach can be summarised in Figure 5-16. Three additional agents were incorporated in the basic architecture to capture the variances caused by decentralised service decision making, asset breakdown structure, and the track of work-in-progress. It should be pointed out that the three variances can be combined, which can ultimately add a number of different model structures.

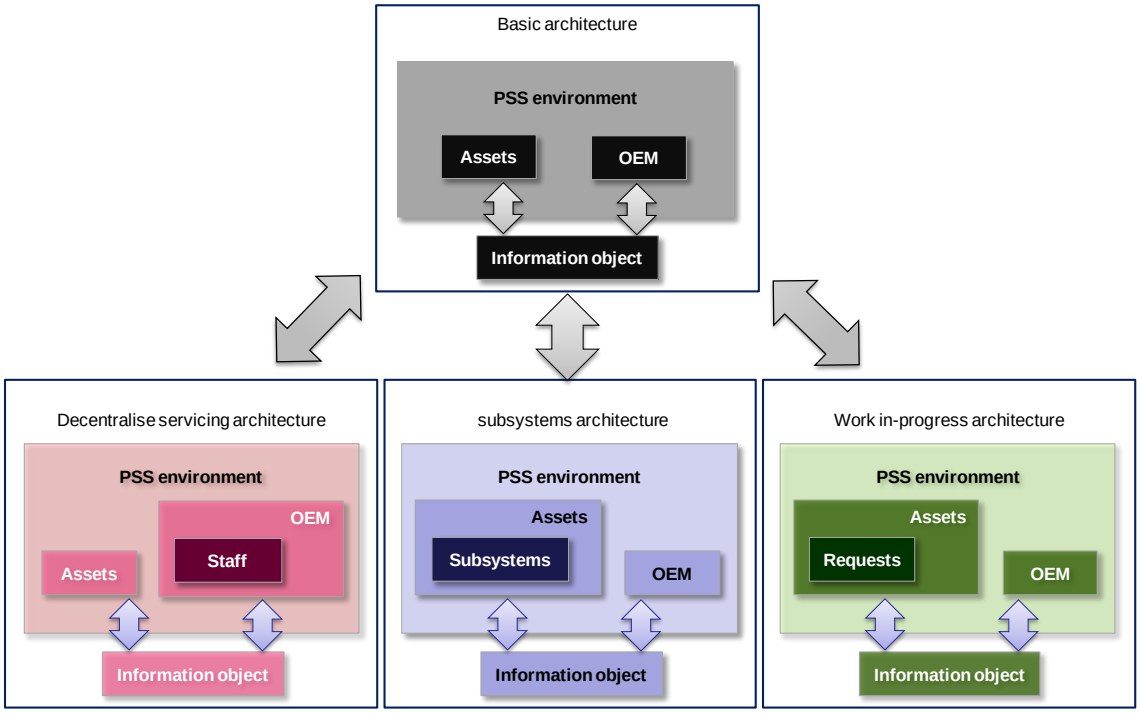


Figure 5-16: Four variations of agent architecture

5.4 Chapter summary

In summary, the applied approach based on the hybrid modelling technique in Chapter 4 was analysed in this chapter. In Section 5.1, the application of this technique to various cases was presented. The cases include aircraft, photocopier, underground train, and carpet sectors. The results were discussed in Section 5.2 in terms of modelling methods, contributions to PSS modelling research, and complexities. This led to the improved approach in Section 5.3. The approach presents the common PSS elements and the variances which can lead to various model structures. All model structures are evolved from a Java information object and a main model which consists of an OEM agent and asset agents in a PSS environment. These structures underlie the modelling constructs presented in the next chapter.

6 Formation of the modelling constructs

In the previous chapter, the refined approach suggested a fundamental model structure for service contracts and its variations for case customisation. This chapter is in accordance with the third objective of this thesis which aims to form modelling constructs, based on the refined approach from Chapter 5. To achieve this, Section 6.1 gives an overview of the constructs, Section 6.2 is associated with the basic service contract constructs, Section 6.3 provides a description of case-dependent variances and their constructs, and Section 6.4 presents the summary of this chapter.

6.1 Overview of the constructs

The term construct in this thesis covers modelling elements and modelling methods. Examples of modelling elements involve source, sink, state, transition etc., whilst modelling methods relate to message passing, object creation, condition-based triggers etc. A three-stage roadmap, presented in Figure 6-1, illustrates how to use the constructs.

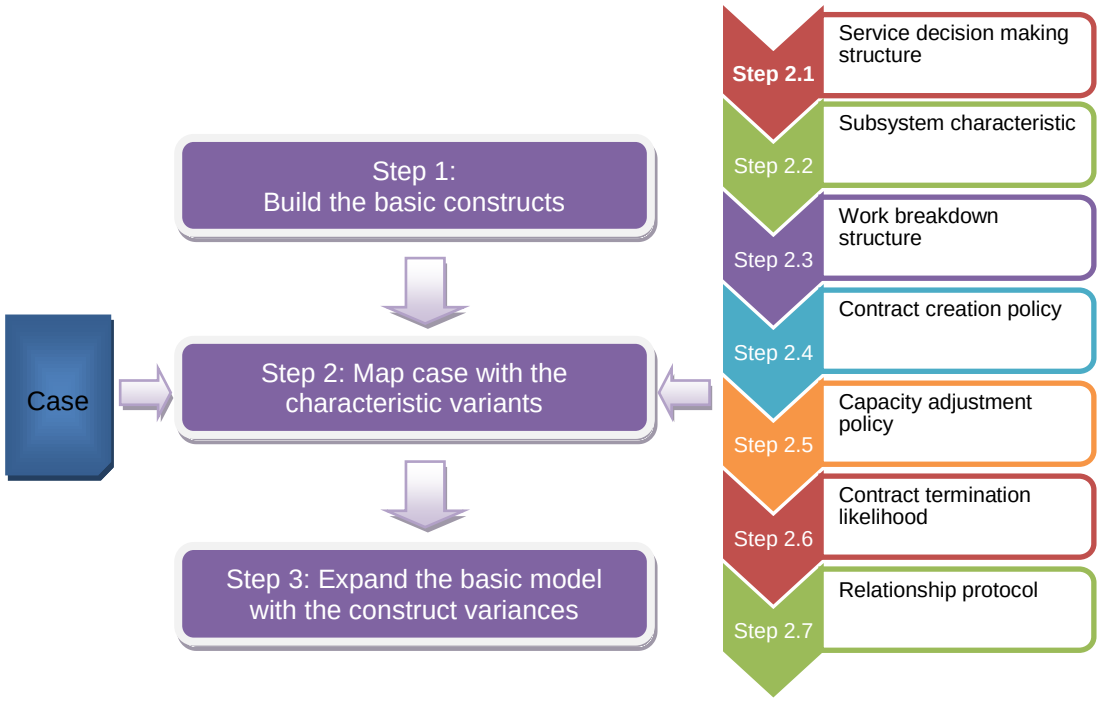


Figure 6-1: The three step roadmap

1. The first stage is to develop *a basic service contract model*. This basic model captures the common PSS elements among service contracts.

2. The second stage is to customise the model to suit the business case. At this stage, the characteristics of the case are mapped with the given variants.

3. The last stage is to modify the basic model using the case-dependent constructs provided for the selected variants. Unless stated and highlighted in black, the constructs are identical to the basic model. Users can also apply this resulting model for further detailed developments.

The next section describes the first stage of this roadmap.

6.2 Development of the basic service contract construct

The shared PSS elements are associated with asset health and life cycle, OEM service process, in-service asset information, service efficiency measures, and usage unit. To account for these PSS elements, the basic construct consists of two layers and encapsulates four fundamental model components.

The first layer contains the following components: an *OEM agent* and *Asset agents* in a *PSS environment*, whereas the second layer details the OEM service process inside the OEM agent and asset's states inside the Asset agents. The last element is *a Java object* which refers to the asset information passed to the OEM prior to servicing. This information relates to communication method in the constructs which represent OEM-customer relationships in PSS businesses.

The two-layer design implies that the in-service assets are not managed by the OEM. In other words, they are independent but interact under the PSS environment. This design can also expose individual asset information, and demonstrate that interactions between the OEM and customers during the in-service phase are triggered by the asset state.

The relationship configuration and the agent role in the model were designed using a role-based agent modelling approach, and presented by Business Process Modelling Notation (BPMN) in Figure 6-2. The role-based approach identifies agent's interactions in a model as a result of agent's recurrent activity (Kendall, 1998). This coincides with the fact that in-service assets and OEM operate independently but interact on the servicing basis. According to Figure 6.2, the simulation engine keeps monitoring the service schedule input by users, and generates unplanned services randomly. Simultaneously, assets operate according to their own schedule until either the operation is completed or the simulation engine notifies a servicing event. In the latter case, the Asset agent issues a Java package which contains asset information so that the OEM agent can perform the appropriate service. This mechanism also automatically activates the OEM's servicing process. The role of the OEM is to carry out necessary services and approve that the asset is capable for operations after

having been serviced. The next step is the detailed modelling of corresponding asset behaviour following this role-based design.

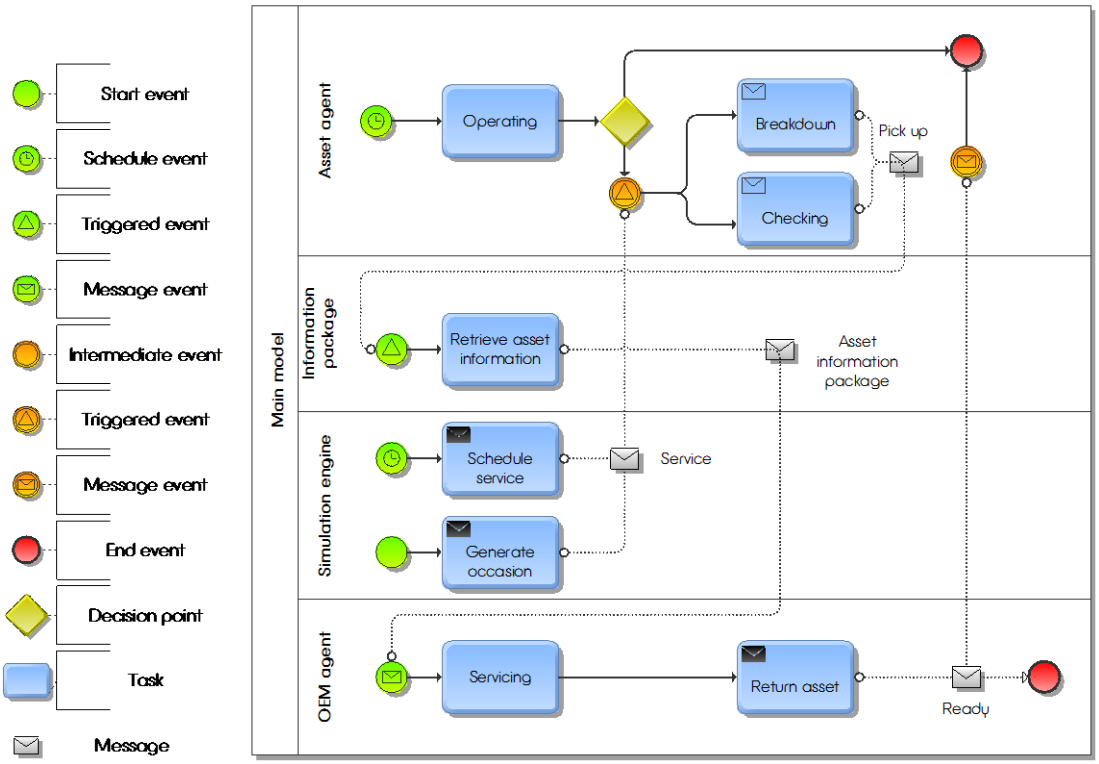


Figure 6-2: High-level relationship of the basic service contract constructs

The basic construct is shown in Figure 6-3. State modelling is implemented inside Asset agents to visualise the asset's state within the in-service phase. The asset construct also includes essential attributes of an individual asset. For example, *ReqAvail* refers to an availability agreement input by the customer. This parameter is also compared against the actual service performance to calculate the penalty after being returned from the OEM. The actual service performance is set to depend on delay shown as *MissedHrs* and updated continuously during the *NotReady* state. Assets are operated once notified by the event *Demand.* Upon the change in state, the operating condition (*OpCon*) is defined. After the operation, the asset's usage is updated. The construct results in the model in which its elements are captured as shown in Figure 6-3.

The programming methods such as message-type trigger are typical in an agent-based model. The choice of these methods is based on lessons learnt during the model development in Chapters 4 and 5 and aims for case generalisation. As it is not directly linked with the contribution to knowledge, the details will be excluded from this thesis.

As for the OEM agent, the personnel are often trained to follow a procedure when carrying out services. Thus, these tasks can be process-driven, in which process modelling methods (typically based on DES) can be used to model the agent behaviour. By using DES, it can be implied that the Asset agents become passive and lose their autonomy once they enter the OEM's system. The cycle time of the service is linked with Mean Time To Repair (MTTR) which also indicates the time taken to recover an asset. This scenario results in the elements shown in the OEM agent in Figure 6-4 and the modelling methods detailed in Figure 6-3. In the figure, the time assets entering the OEM system is recorded via the 'Enter' element, which allows recovery period to be calculated (in the 'Exit' element of Figure 6-4).

To enable a record of the individual asset, a Java object is created prior to the servicing task to represent the actual asset and it behaves as if the Asset agents themselves are moving in and out of the OEM's process. The fact that this object is created from the Asset agents (enabled by *From* in Figure 6-3) and destroyed by the OEM agent, only at the point of interactions, distinguishes this approach from the traditional approach. Traditionally, the asset information is often encompassed inside the asset and cannot be visualised.

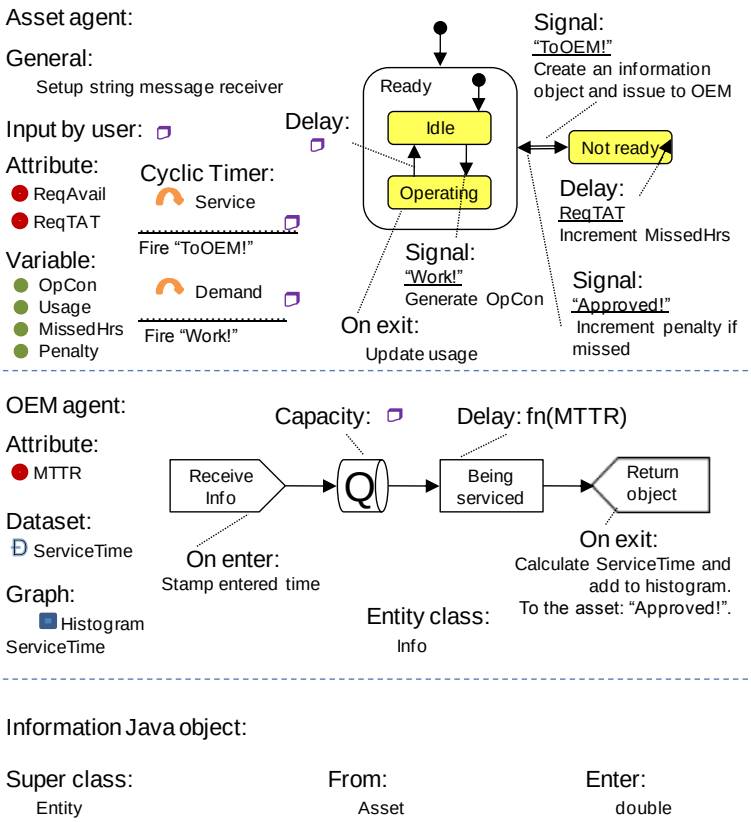


Figure 6-3: The basic service contract construct

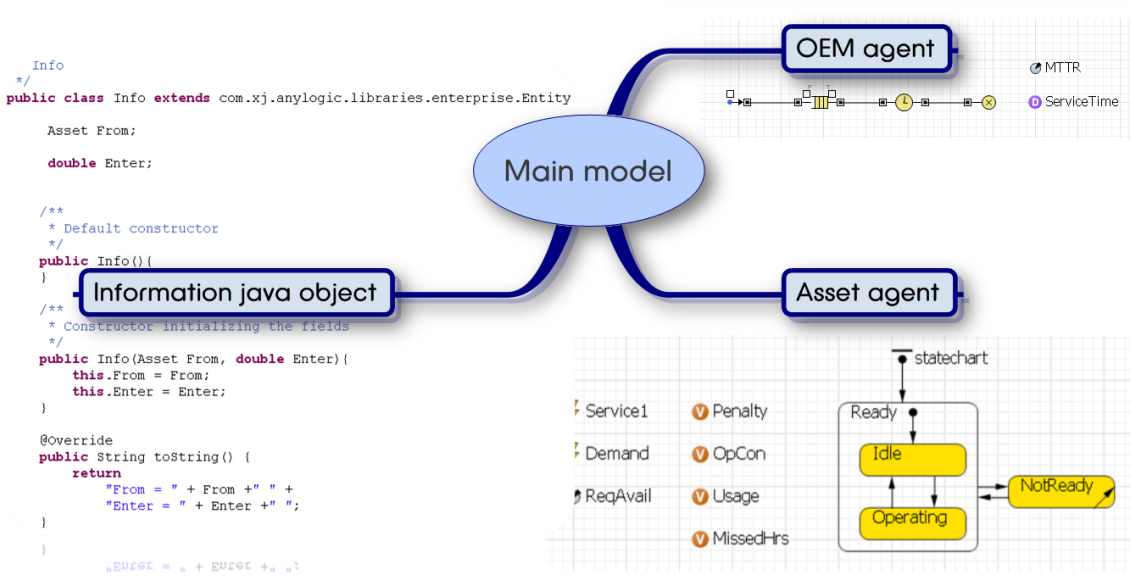


Figure 6-4: The basic model

Next, the second and third stages of the roadmap are explained.

6.3 Development of the case-dependent constructs

The case-dependent constructs capture differences among service contracts, using the basic construct as the baseline. Therefore, the variances are categorised as follows:

- Service decision making structure
- Subsystem characteristic
- Work breakdown structure
- Contract creation policy
- Capacity adjustment policy
- Contract termination likelihood
- Relationship protocol

The variants within these categories are summarised in Table 6-1, along with the resulting modelling variances from the basic constructs.

Table 6-1: Summary of the variances from the basic contract

Characteristic variance	Characteristic variants	Construct variances
Service decision making structure	A0: An OEM has a fixed routine in performing services and each service has standard time	Use the basic model with an input distribution in the OEM's delay element
	A1: An OEM has a fixed service routine, but with adaptive productivity upon the global view of situation	There are two levels of inputs in the OEM's delay element
	A2: An OEM has adaptive productivity and flexible routine	Staff are created as another agent under the central OEM
Subsystem characteristic	B0: The contracted product's state can be predicted on an aggregate level	A rate event is created inside the Asset agents to account for non-scheduled services
	B1: The contracted unit requires breakdown analysis into subsystem levels.	Subsystems are created as another agent under the Asset agent
	B2: Subsystem behaviour can influence one another and the contracted unit	In addition to B1, Interaction between the Subsystem agents are enable
Work breakdown structure	C0: Service performance is measured only at the end of all operations	n/a
	C1: Jobs are preceded by several departments and service performances are measured separately (A1)	Jobs are created as a separate agent issued by the Asset agent to track different performance requirements
	C2: Jobs are preceded by several departments and service performances are measured separately (A2)	Based on A2 structure, Jobs are created as a separate agent issued by the Asset agent to track different performance requirements
Contract creation policy	D1: A new contract is signed based on staff utilisation	OEM agent reacts to real time utilisation
	D2: There is no predefined situation when the OEM should make new contract	User can interact to create new contract
Capacity adjustment policy	E0: There is no rule when to adjust capacity (A1)	User can interact to adjust number of staff
	E1: There is no rule when to adjust capacity (A2)	Based on A2 structure, user can interact to adjust number of staff
	E2: Capacity is regularly adjusted based on certain rule	OEM agent learns and adjusts number of staff in real time.

Characteristic variance	Characteristic variants	Construct variances
Contract termination likelihood	F0: Contract termination is not possible	n/a
	F1: Customers can negotiate to end the contract	The Asset agents monitor OEM performance and reacts in real time.
Relationship protocol	G0: The OEM proceeds all demands from the contracted customers	n/a
	G1: The OEM may subcontract or reject the demands from the contracted customers, but it may cost the OEM a penalty (in case of A1).	The Asset agent state chart's are modified to include outsourcing, and OEM agent has another servicing choice in addition to basic structure
	G2: The OEM may subcontract or reject the demands from the contracted customers, but it may cost the OEM a penalty (in case of A2).	Following A2 structure, the Asset agent's state chart is modified to include outsourcing, and OEM agent has another servicing choice in addition to basic structure

6.3.1 Service decision making structure

The variation in this category results from various level of decentralised decision making. Three scenarios can be evolved from this category.

A0: An OEM has a fixed routine in performing services and staff productivity is constant for the same type of service.

This refers to the case in which the OEM's servicing staff are trained to follow a sequence of tasks and are not allowed to deviate from the instructions. The staff are restricted in the sense that they will not be able to speed up the job as the workload increases. In other words, the productivity does not change for the same type of job. For instance, in aero engine overhaul, staff are obliged to carry out the full sequence of inspection procedures and the judgement on asset condition is based on the predefined property. The construct of this type of system can be based on the basic OEM construct, in which a statistical distribution can be encompassed inside the delay element to capture cycle time variation among staff. In the case that various services are offered by the OEM, for example machine tools (Meier et al., 2010), services cover function maintenance, planning, logistic, and training, the replication of a delay element can capture these different services.

A1: An OEM has a fixed service routine, but with adaptive productivity upon the global view of situation

This refers to the case in which the productivity can change significantly in the same job depending on the overall workload, and the staff can realise and share the

workload equally. In other words, staff can shorten service cycle time (per one job) once they realise that they have more assets to be serviced than they usually do.

This characteristic can be captured through minor modification to the basic OEM construct (Figure 6-5). The cycle time is self-adjusted upon the overall workload. This implies that the staff are aware of the queuing jobs in the system at the beginning of each operation, and adopt their productivities accordingly. Here, the parameter *Adaptive* depicts the baseline of the number of jobs the staff often perceive as a high workload.

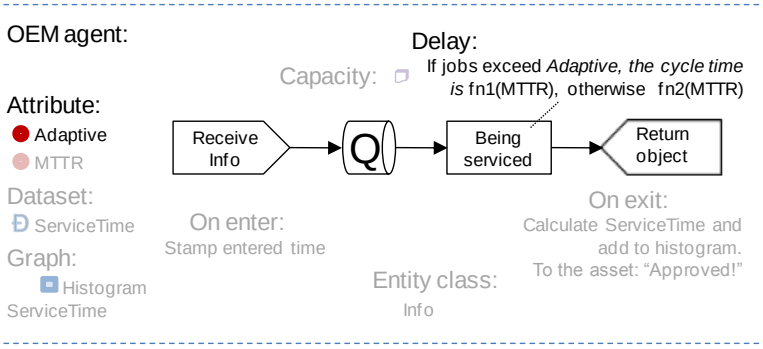


Figure 6-5: A1 construct

A2: Adaptive productivity, flexible routine, no global view of situation

In this case, jobs are assigned to servicing staff from the central OEM thus the staff have no knowledge about the overall situation. Therefore, their productivity is according to their jobs in hand. The staff can use their own judgement, based on their skills and experiences, to perform necessary tasks. An example of this situation was described by Watson et al. (1998) in case of Xerox field services.

As the staff become more autonomous and decentralised in this case, the basic OEM construct is not suitable. Following the approach from Chapter 5, the staff are created as agents embedded inside the OEM model (Figure 6-6). However, the central OEM model still needs to play its role in assigning jobs to the field staff. Therefore in the model, the incoming jobs are collected at the *AssetInQ. O*nce this happens, the OEM agent activates the *AssignJob* to allocate the job to field staff based on the staff's workload. The algorithm inside the *AssignJob* can be implemented as shown in Figure 6-7. Here, the staff have flexibility to perform tasks. Once completed, the asset is approved for operational-readiness.

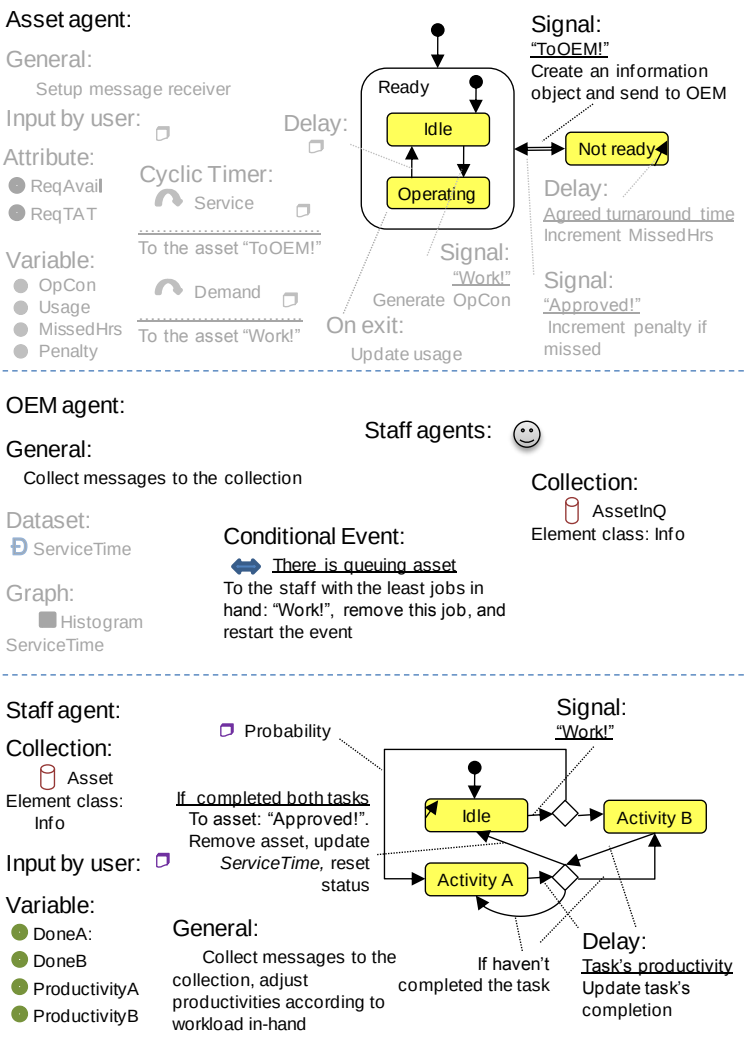


Figure 6-6: A2 construct

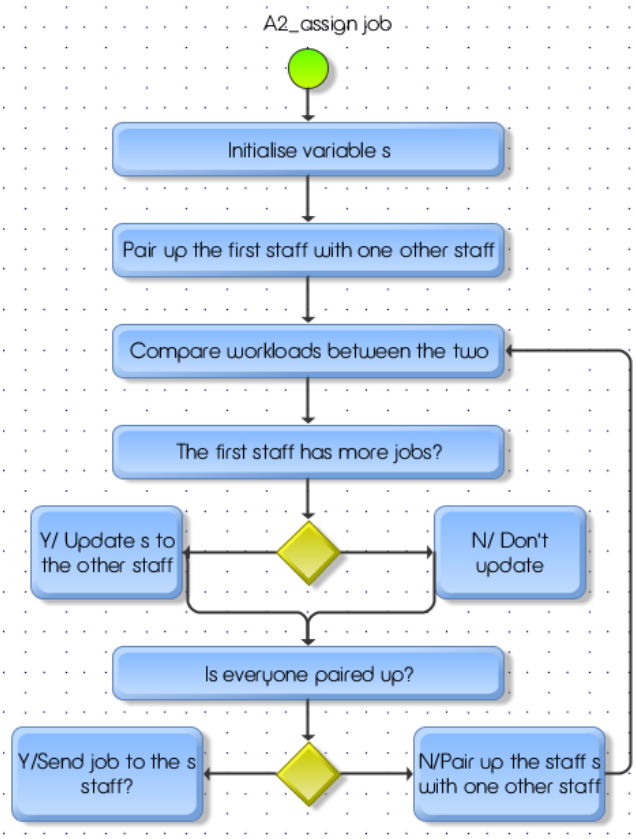


Figure 6-7: Job allocation logic

In summary, this case-dependent characteristic focuses on OEM processes. It highlights staff's autonomy in performing services and staff's learning ability to adapt themselves according to workload. This corresponds with the agent's capability in being flexible, described in Section 5.3.1.

6.3.2 Subsystem characteristic

The variants in this category are linked with product complexity and contracted unit, which can be divided into three scenarios.

B0: The contracted product's state can be estimated on an aggregate level

This characteristic refers to the case that the contracted product has a failure pattern and it is sufficient to consider on a high level. Therefore, it is not necessary to decompose asset analysis into individual subsystems. Examples of this case are carpet, washing machine, and a software package in which the PSS providers can estimate time between services quite accurately from historical data.

The basic asset construct is applicable to this classification. As shown in Figure 6-8, the *Service* events are created to trigger an asset's state both randomly and once the MTBF is reached.

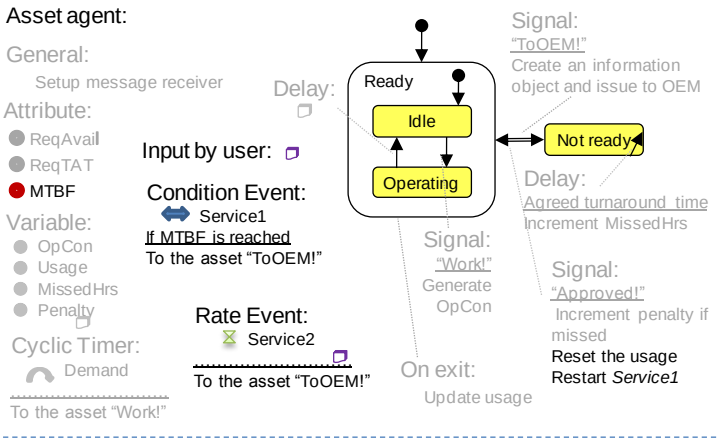


Figure 6-8: B0 construct

B1: The contracted unit requires breakdown analysis into subsystem levels.

This variant can be applied if the contracted assets are made up of heterogeneous subsystems which have significant differences in their failure pattern. This can refer to a launderette which consists of several washing machines and dryers, and a computer network which comprises a number of computers and printers. Note that the subsystems in this variant are independent and not influenced by one another.

In this case, the contracted asset's state depends on its subsystem behaviour. Therefore, these subsystems can be defined as agents encapsulated inside the contracted product (Figure 6-9). Each subsystem can encounter different degradation rates from other subsystems, and adjust itself differently on various operating conditions. Once servicing is performed, degraded subsystems may be replaced or not, depending on the variable *ChangeLikelyhood.* This variable is driven by the remaining useful life which is a function of the past operating conditions (*AveOpCon*) and the estimated life (*Life*).

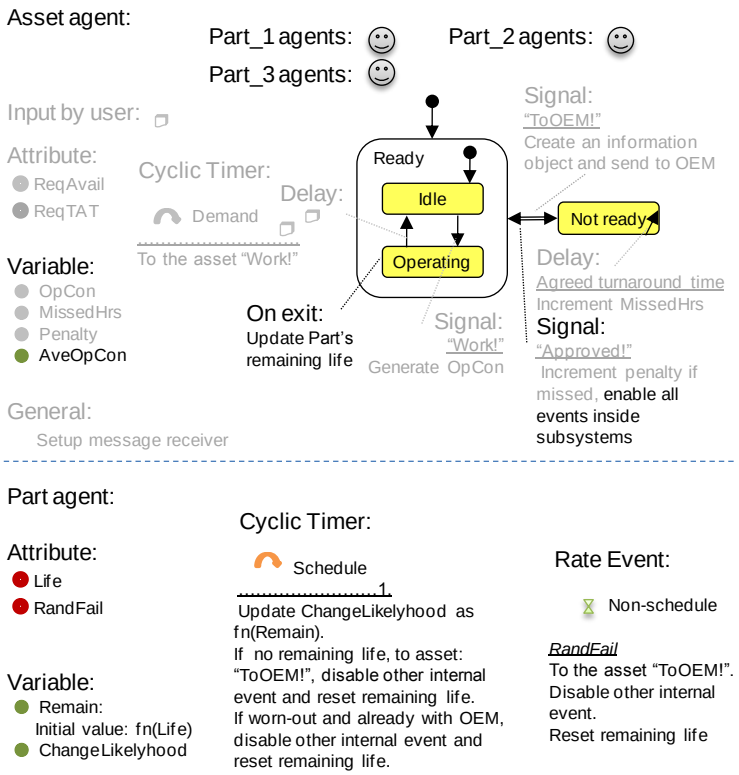


Figure 6-9: B1 construct

B2: Subsystem behaviour can influence one another and the contracted unit

This situation can happen, for example, in aircraft when one failed engine may carry on flying and does not affect the whole aircraft for a short period of time, but causes other engines to operate at a higher load.

In addition to B1 construct, a state chart is added to the subsystem model (Figure 6-10). The subsystem can be exhausted from itself or from others (through the message passing mechanism).

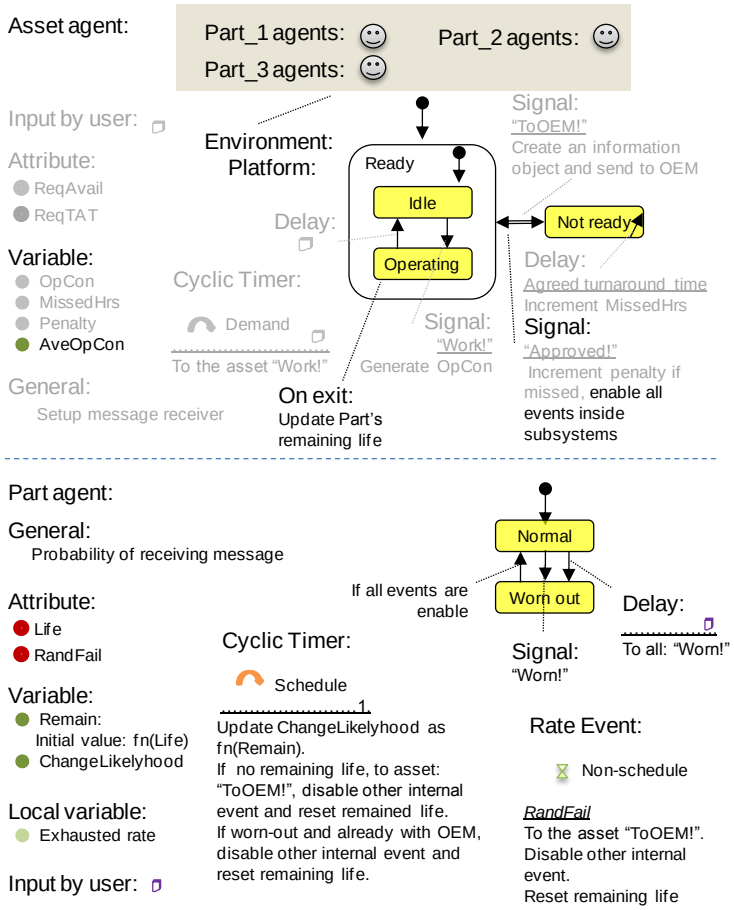


Figure 6-10: B2 construct

In conclusion, this category adopts agent functionality in capturing subsystem's heterogeneity and loose interaction, since predicting system behaviour from an aggregate level can be a tedious task.

6.3.3 Work breakdown structure

C0: Service performance is measured only at the end of all operations

This case refers to the basic construct in which intermediate servicing states do not have to be monitored.

C1: Jobs are preceded by several departments and service performances are measured separately (A1)

Examples of this category are Xerox DocuCare package, in which service performance is evaluated twice: upon issue response and technical response.

Based on the refined approach in Chapter 5, an agent, named *Request* is embedded inside the Asset agents. The following modifications to the basic construct are required:

1. Identify that the information package is issued by the Request agents instead of the Asset agents (see information Java object in Figure 6-11).

2. Add a delay element to represent different departments of the OEM agent and declare that the information package is passed along these departments (see OEM construct in Figure 6-11).

3. Decouple all communication between the OEM and the Asset agents and centralise all interactions to the Request agent. In other words, all types of requests are created by the Asset agents once the assets are not ready (see the asset construct in Figure 6-11). Following the creation, the information object is issued by the Request agent and passed to the OEM agent. The OEM agent handles the request and updates the status of the request by each department (see OEM construct in Figure 6-11). Therefore after all services are performed, the Request agent signals the Asset agent and destroys itself (see Request construct in Figure 6-11).

4. The delays (*MissedHrs*) are updated by the Request agents at the end of each state, rather than by the Asset agents at the completion.

These results are shown in the construct in Figure 6-11.

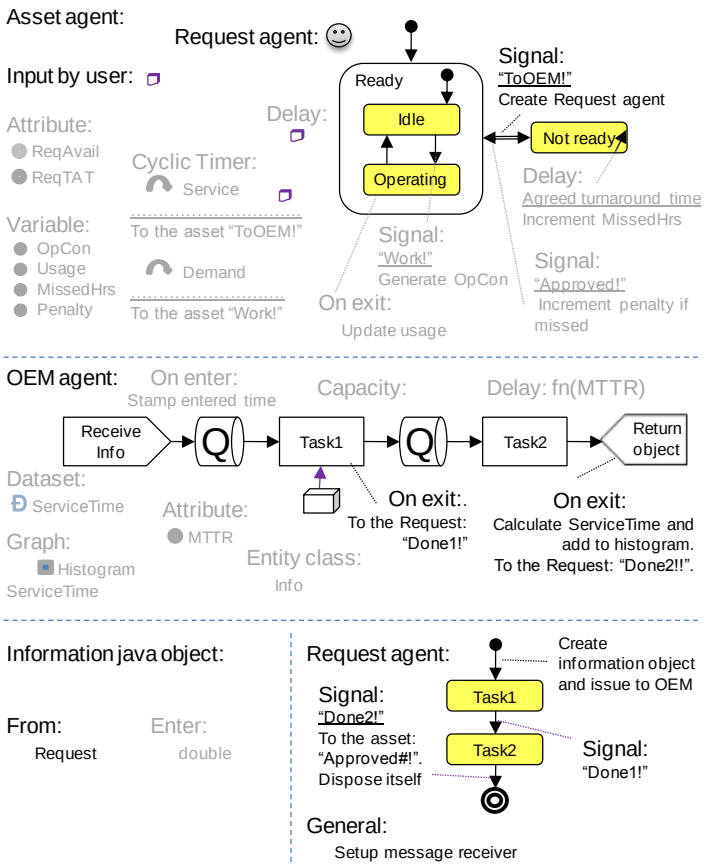


Figure 6-11: C1 construct

C2: Jobs are preceded by several departments and service performances are measured separately (A2)

This characteristic is identical to C1, yet, the construct is different due to the decentralised service structure in A2. The majority of methods are the same as C1, except that the job completion is no longer updated by the last service element of the OEM agent but the Staff agent.

The resulting construct are presented in Figure 6-12.

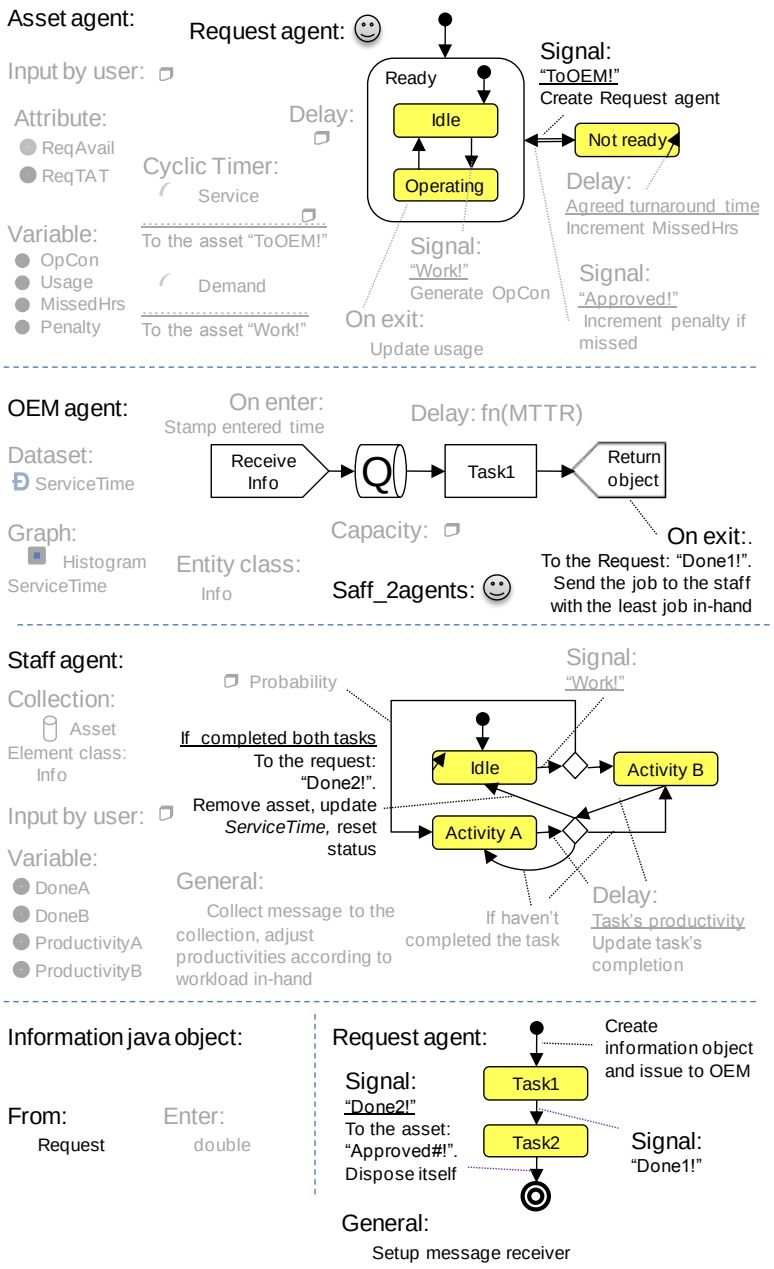


Figure 6-12: C2 construct

This category adopts an agent's capability to define and expose recovery progress of individual asset. The agent's functionality in socialising is also used to update and monitor the progress.

6.3.4 Contract creation policy

Two variants are introduced with regards to making new contracts.

D1: A new contract is signed once staff utilisation is low

In this case, the OEM regularly checks staff utilisation. If the value is low and the accumulated penalty is still acceptable, the OEM seeks for a new contract.

A *NewContract* event and an adjustable *AllowPenalty* variable are incorporated into the OEM model (Figure 6-13). The action code inside the event corresponds to the aforementioned constraint.

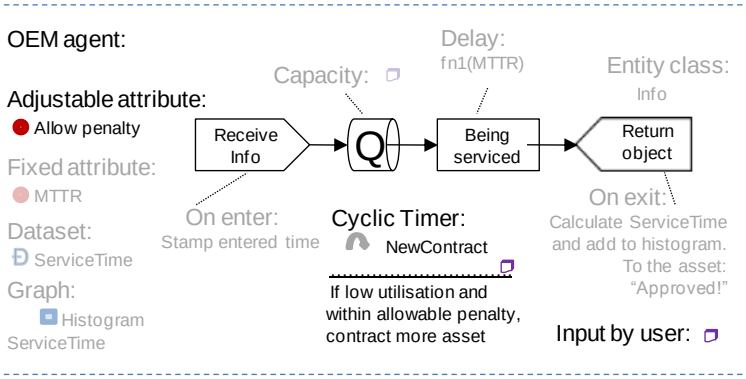


Figure 6-13: D1 construct

D2: There is no predefined situation when the OEM should make new contract

In this case, there is no predefined rule for contract creation. Therefore, the main model was modified to enable interactive creation by users via a button control (Figure 6-14).

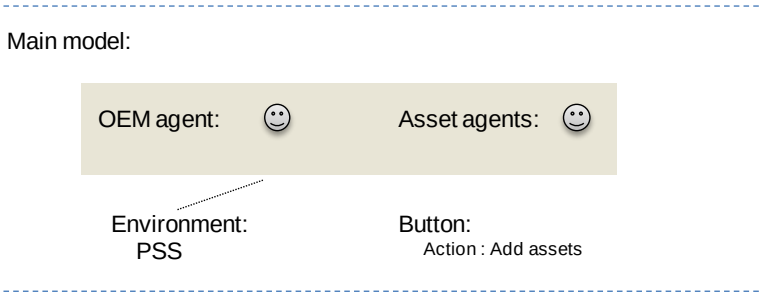


Figure 6-14: D2 construct

Overall, this category applies the agent's functionality to the OEM agent so that it can sense current performances and adapt the policy accordingly. Besides, D1 construct can be used to estimate an appropriate number of contracted assets as a result of a defined rule against current capability. On the contrary, A2 enables users to interact and make immediate decision.

6.3.5 Capacity adjustment policy

This category accounts for resource planning, which involves recruiting and laying-off servicing staff.

E0: There is no rule when to adjust capacity (A1)

This means the decision on capacity takes place in a flexible manner. There is no predefined standard why the OEM should increase or decrease capacity. Accordingly, the OEM construct is modified to allow an interactive capacity adjustment by the model users (Figure 6-15). The parameter *Capacity* is linked to the service element, and can be changed during the model execution e.g. by the connected slider.

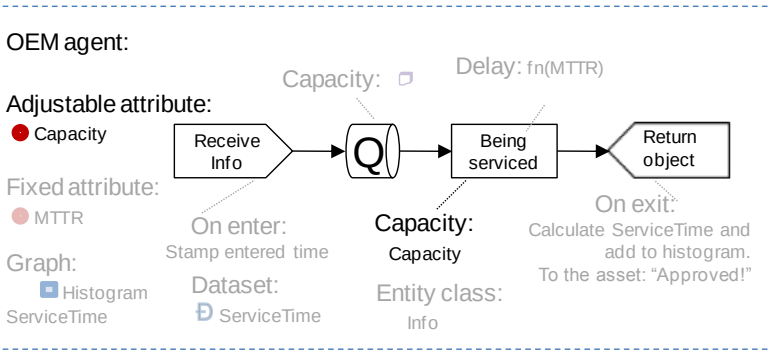


Figure 6-15: E0 construct

E1: There is no rule when to adjust capacity (A2)

This classification is identical to E0, but corresponds with the A2 servicing characteristic. The OEM construct is modified to enable user interactive adjustments (Figure 6-16). Here, an algorithm is required to ensure that the removal takes place only to idle staff, otherwise the job would be also removed without completion. An example of programming logic to handle this situation is shown is Figure 6-17.

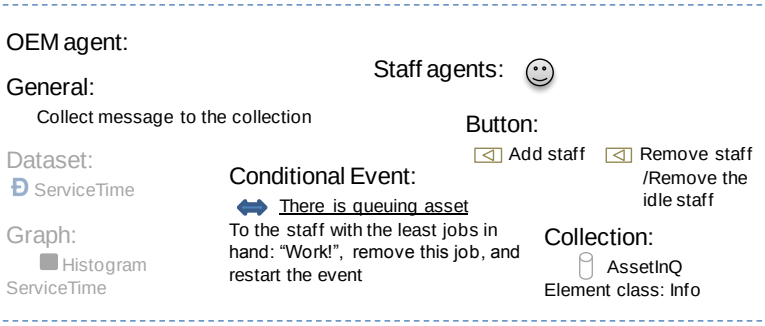


Figure 6-16: E1 construct

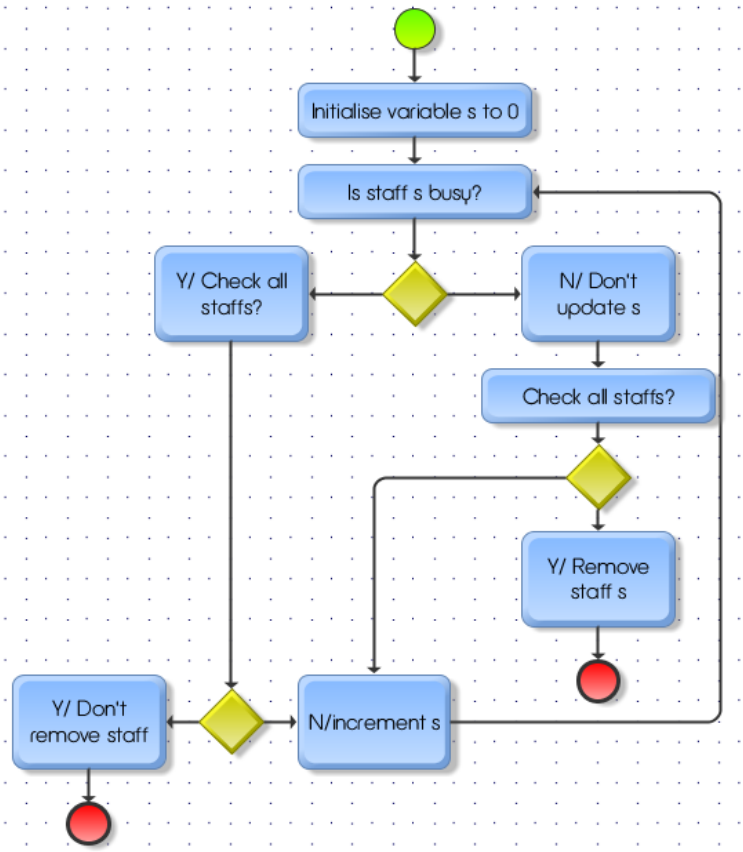


Figure 6-17: Staff agent removal logic

E2: Capacity is regularly adjusted based on certain rule

If the rule is based on utilisation, the OEM keeps monitoring utilisation levels and makes decisions based on the maximum and minimum levels. The *Adaptive* event updates the *Capacity* once the *MaxU* or *MinU* input by the user is exceeded (Figure 6-18).

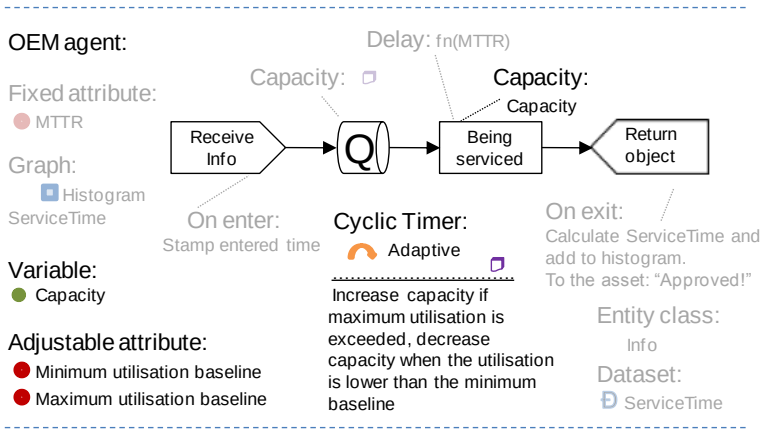


Figure 6-18: E2 construct

In summary, this category embeds the agent's capability to learn for demonstrating a demand chasing strategy in resource planning. The algorithm for staff removal, which prevents lost jobs, encapsulates the agent's functionality in being goal-directed. The self-adaptive functionality in E2 enables automatic adjustment based on the OEM's goal whereas the E0 and E1 variants embrace the flexibility in manually adjusting input.

6.3.6 Contract termination likelihood

F0: Contract termination is not allowed

This case ties up both parties until the end of the contract, which can be represented by the basic construct.

F1: It is possible that customers will negotiate to end the contract

For instance, if a customer is not satisfied by the increasing delays, he may request to terminate the contract. After receiving the request, the OEM may accept or negotiate to continue the contract.

In the asset construct (Figure 6-19), a cyclic event (*Learning)* and a parameter *(MaxMiss)* captures the customer dissatisfaction. The event keeps monitoring whether a delay exceeds the acceptable limit *(MaxMiss).* If so, a Java object *(Quit)* is sent to the

OEM agent to ask for termination. The OEM agent registers the request in the *EndRequest* which activates the *Termination* event. The role of this event is to consider whether to approve the termination.

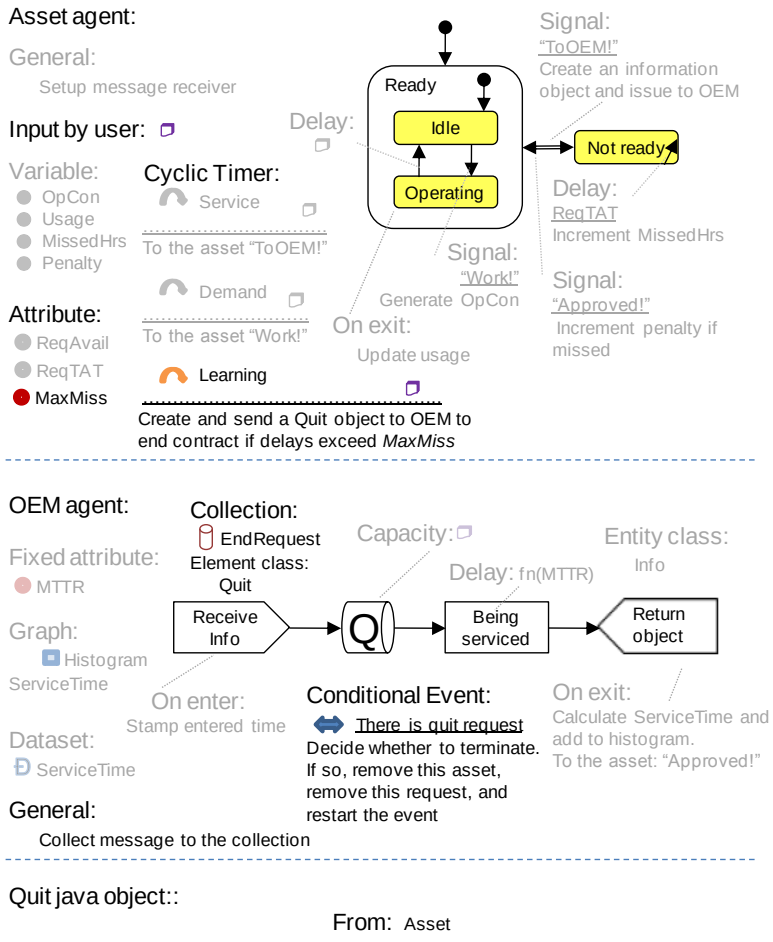


Figure 6-19: F1 construct

On the whole, this category embeds the agent's capability to learn inside the Asset agent and agent autonomy inside the OEM agent. As this mechanism can influence the number of contracts, F1 construct can be use to self-generate an affordable number of contracts.

6.3.7 Relationship protocol

G0: The OEM precedes all demands from the contracted customers

An example of this classification is Performance-Based Logistic (PBL) which commits the OEM to provide all support services to deliver mission capable assets. This classification can be demonstrated by the basic OEM construct.

G1: The OEM may subcontract or reject the demands from the contracted customers, but it may penalise the OEM (in case of A1).

For instance, the OEM is obliged to provide services, however, it may take longer than it was agreed. Thus the customer selects another service provider and charge the OEM according to the agreed penalty. Another example takes place in the Electrolux case who offered a contract that specifies certain number of services per year. The OEM can reject additional services if staff are unavailable.

A variable *JobIn* and one additional route are added to the basic OEM construct, as shown in Figure 6-20. The variable updates the number of jobs-in-hand. If it is beyond available staff, the job is routed away from OEM system and the asset's state is updated.

The asset state chart is modified to Figure 6-20. The Asset agent moves to the *WithOther* state once a rejection message is received from the OEM.

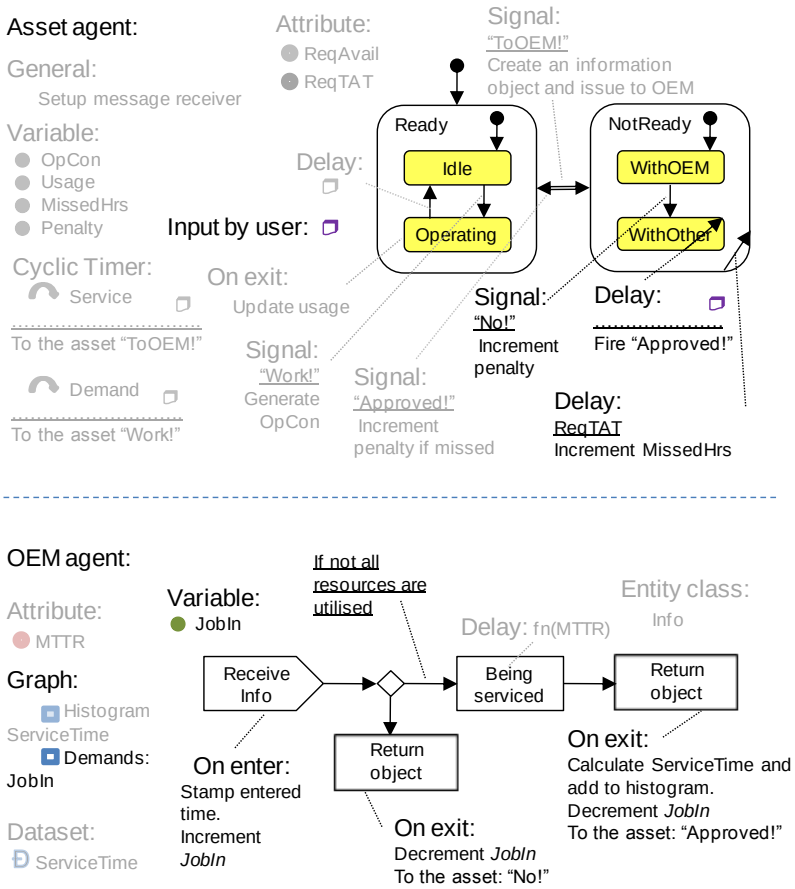


Figure 6-20: G1 construct

G2: The OEM may subcontract or reject the demands from the contracted customers, but it may penalise the OEM (in case of A2).

This characteristic is the same as G1 but corresponds to the A2 servicing characteristic.

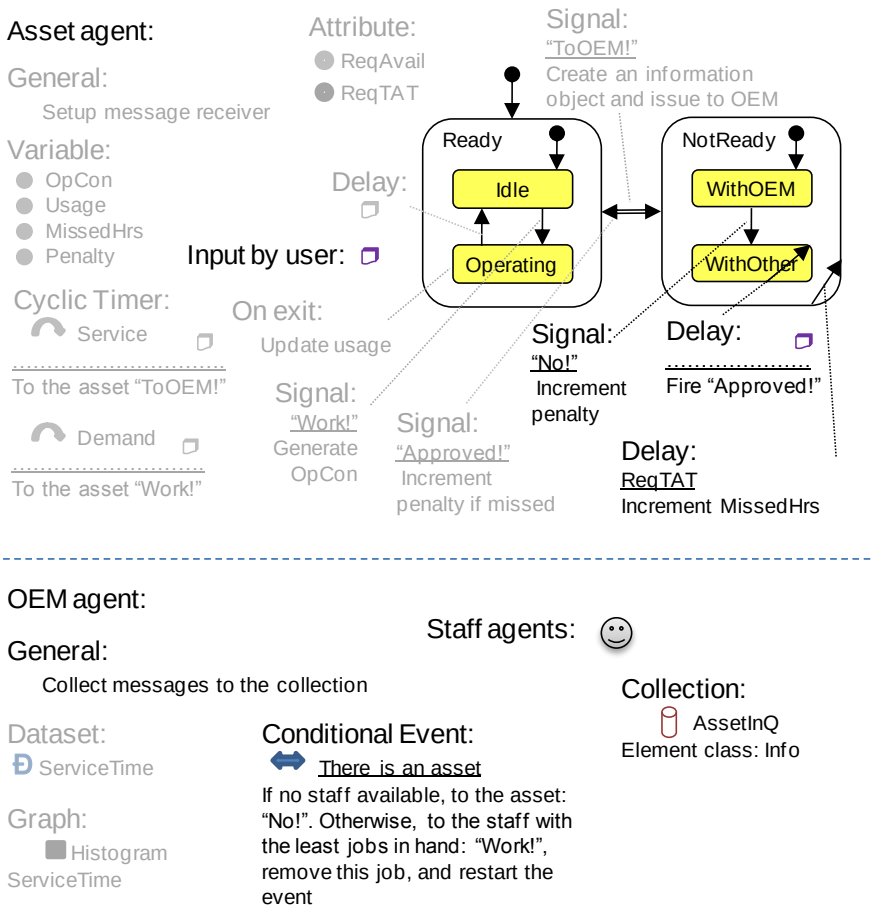


Figure 6-21: G2 construct

All constructs have been verified individually after being completed. Next, their contributions to knowledge in PSS modelling field are discussed.

6.4 Assessment of the constructs against the current state of PSS modelling

This section identifies the merits in the constructs in describing PSS business in relation to the current state of PSS modelling, as summarised in Table 6-2. The plus signs indicate strengths in the literature or covered issues in the constructs, whilst the minus signs dictate weaknesses in the literature or uncovered issues in the constructs.

Table 6-2: Summary of the constructs' capability against the current stage of PSS modelling.

No.	Criteria	Literature	The constructs
1	Generalise to all PSS	+	-
2	Incorporate some analytical techniques	+	-
3	Extend life cycle perspective from product selling	+	+
4	Address value parameter explicitly	+	+
5	Highlight interactions between parties in supply chain and customer.	+	-
6	Include both economic and environmental measures	+	-
7	Demonstrate the link between asset transformation and service support	+	+
8	Present service efficiency measures	-	+
9	Capture link between product performance and customer-manufacturer relationship	-	+
10	Illustrate redesigns from customer involvements	-	+
11	Demonstrate decentralise decision making	-	+
12	Represent cultural mind frame, social habits, and influence between customers	-	-
13	Capture effect of technology on company's capability	-	-
14	Incorporate government influence	-	-
15	Embed input uncertainties	-	+
16	Explicitly present asset's lifecycle	-	+
17	Expose asset's autonomy	+	+

It can be seen that the constructs contribute to the gap of knowledge to a great extent. The constructs can demonstrate ten issues in which six are lacking in literature, as follows:

1. Extended product life cycle from manufacturing (No.3) can be described in the Asset agents by their state charts.

2. The 'value-in-use' parameter (No. 4) can be defined, in this case, as required availability (*ReqAvail)* in the Asset agents.

3. The link between asset transformation and service support (No. 7) can be demonstrated through the Subsystem agent's lifetime and the OEM agent's MTTR as in B1 and B2 constructs.

4. Service efficiency measure (No.8) can be highlighted, in this case, as turnaround/ recovery time in the histogram inside the OEM agent, and delays (*MissedHrs*) inside the Asset agent.

5. Impacts of product performance on OEM-customer relationship (No.9) can be illustrated from the fact that a Java object is created from the Asset agent's state chart and passed along to the OEM agent.

6. Redesigns from customer involvements (No.10) can be changed in the Subsystem agent's life (as in B1 and B2 constructs) during model executions (by using sliders).

7. Decentralised decision making (No.11) can be demonstrated by decomposing the OEM agent to the Staff agent in the lower hierarchy (as in A2 construct).

8. Input uncertainties (No.15) can be incorporated using input distribution.

9. Asset life cycle (No.16) can be exposed using the state chart inside the Asset agents.

10. An asset's autonomy (No.17) can be implied by modelling the Asset agents in the same layer as the OEM agent.

However, four areas are excluded in the constructs but covered in literature. These relate to the scope of PSS offering (No.1), implementation of analytical methods (No.2), interactions in supply chain (No.5), and environmental measures (No.6). Along this line, three areas of improvements can be made in terms of social influences (No.12), technology (No.13), and regulation (No.14).

6.5 Chapter summary

To conclude, this chapter stated the development of modelling constructs which aim to enhance effective and efficient development of PSS offering simulation models. The constructs consist of two parts; the basic service contract construct and the case-dependent constructs. The basic construct incorporates common PSS elements whereas the case-dependent constructs capture the case-dependent elements analysed in Chapter 5. The contribution to knowledge of the constructs was examined against the current state of PSS modelling discussed in Chapters 2 and 4. The next chapter deals with validation of the constructs.

7 Evaluation of the constructs

This chapter corresponds with the fourth objective of this research, which aims to evaluate and refine the primary constructs. To do so, Section 7.1 describes the first evaluation process via multiple-case studies. In this section, the models were developed for these cases based on the constructs, and presented to industrial users. Section 7.2 deals with the second evaluation process, in which the constructs were tested by users. The findings are discussed at the end of both sections, which led to the refined constructs in Section 7.3. Finally, the chapter is concluded in Section 7.4.

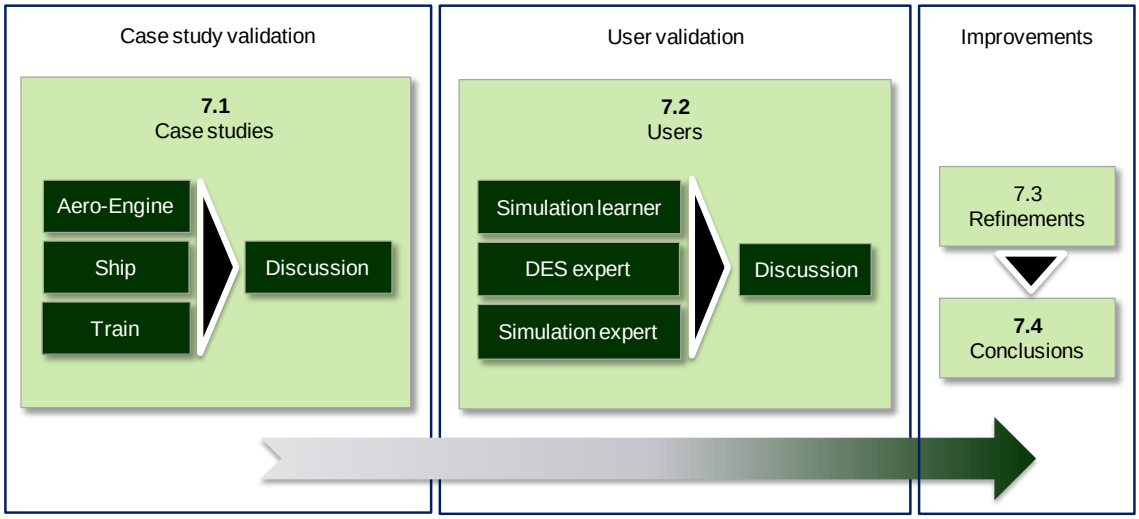


Figure 7.1 Chapter 7 outline

7.1 Case study validation

This section describes the first type of evaluation which is the case study validation. Three cases were chosen under product-centric PSS and from different industries to enhance generalisation. Case study protocol was used for all cases, which focuses on system understanding, and mapping case characteristics with the construct variants (see Appendix I). The interview data were cross-checked with another interviewer who was also present in the interviews to enable member checking technique. Having developed the models using the constructs, they were then demonstrated and rechecked with the case companies. The analysis was made in terms of efficiency, applicability, practicality, and feasibility.

7.1.1 Case I: EngineCo

Data collection process

EngineCo manufactures aero-engines and also provides services such as customer training, engineering support, fleet management and overhaul service. The company has offered service contracts for both military and airline customers globally in the

past five years. Generally, the contract period spans over five years with the value in the region of around $1b and the focus of services is centred around overhaul.

The validation followed an iterative process with a team leader of an overhaul centre. The interaction involved an introduction of this research, face-to-face semi-structure interview, facility visit, presentation of sample models, and follow-up emails for feedback. During the follow-up, a set of slides was attached along with the final model to introduce the user in how to use the model for actual decision making, realised through the interview. The total amount of interaction exceeded 15 hours.

In aero-engine service contracts, airline operators generally own the contracted engines, however, EngineCo is responsible for maintaining them. The contract specifies turnaround time (TAT) and the exhaust gas temperature (EGT) margin based on a given engine operating cycle. The margin is a comparison between the operating gas temperature (an indicator of engine performance) and the maximum allowable gas temperature during flight take-off, hence the higher the number the better.

Based on the requirements, the company gathers past usage information of the engine, checks the leftover cycles of each Life-Limited Parts (LLP), and forms the overhaul schedule to achieve the agreed engine cycle. LLPs are the critical components of an engine, for example, turbine blades. The leftover cycle of each component is obtained from testing, and adjusted by the condition where the engine will be operated. This operating condition is crucial as it can cost millions of dollars more than the expected expense. There are approximately 15 critical LLPs in an engine. The expected number of scraps during each overhaul is estimated and used for calculations of expense and resource required.

Once a contract is made, it is rarely renegotiated, and the contract lasts 5-10 years. The customer generally pays per cycle, but is obliged to pay at a minimum fee if the engine is not used. Contracts are not normally renegotiated and terminating a contract costs the terminator the predefined charge.

Regarding service operations, four major stages take place in an overhaul, referred to as Gate 0 to Gate 3. Gate 0 collects all engine information, which generally takes one day. Gate 1 disassembles and cleans the engine or passes the modules to partners. It also involves non-destructive testing and inspection. Gate 2 deals with repairs which may take up to 20 days. Finally, Gate 3 reassembles and tests the engine. Different groups of staff are assigned to different stages. However, staff are trained to be capable of performing any tasks.

Based on these data, the next section maps the case characteristics with the constructs.

Validation outcomes

Step 1: Formulate the basic model

At this step, the basic model was formed based on the basic construct as follows:

1. The main model was created, which consists of a PSS environment, an OEM agent, and Asset agents.

2. The information Java object was created.

3. The OEM agent was defined. At this stage, some amendments from the basic OEM construct were made to the model. Firstly, *BeingServiced* was divided into four elements to demonstrate the four gates of the overhaul service. Secondly, MTTR was no longer made as an independent attribute, but embedded inside the 'delay' elements. This is because each gate always has a fixed and predictable standard time.

4. The Asset agents were defined. Three modifications from the construct took place at this stage. Firstly, *OpCon* was created as a user's interactive input because the operating condition is specified in the contracts. Secondly, the performance requirement in this contract is not availability level, but turnaround time. Thirdly, the *MissedHrs* variable was eliminated as the penalty could be calculated directly from the delayed turnaround time.

This step resulted in the model in figure 7-2.

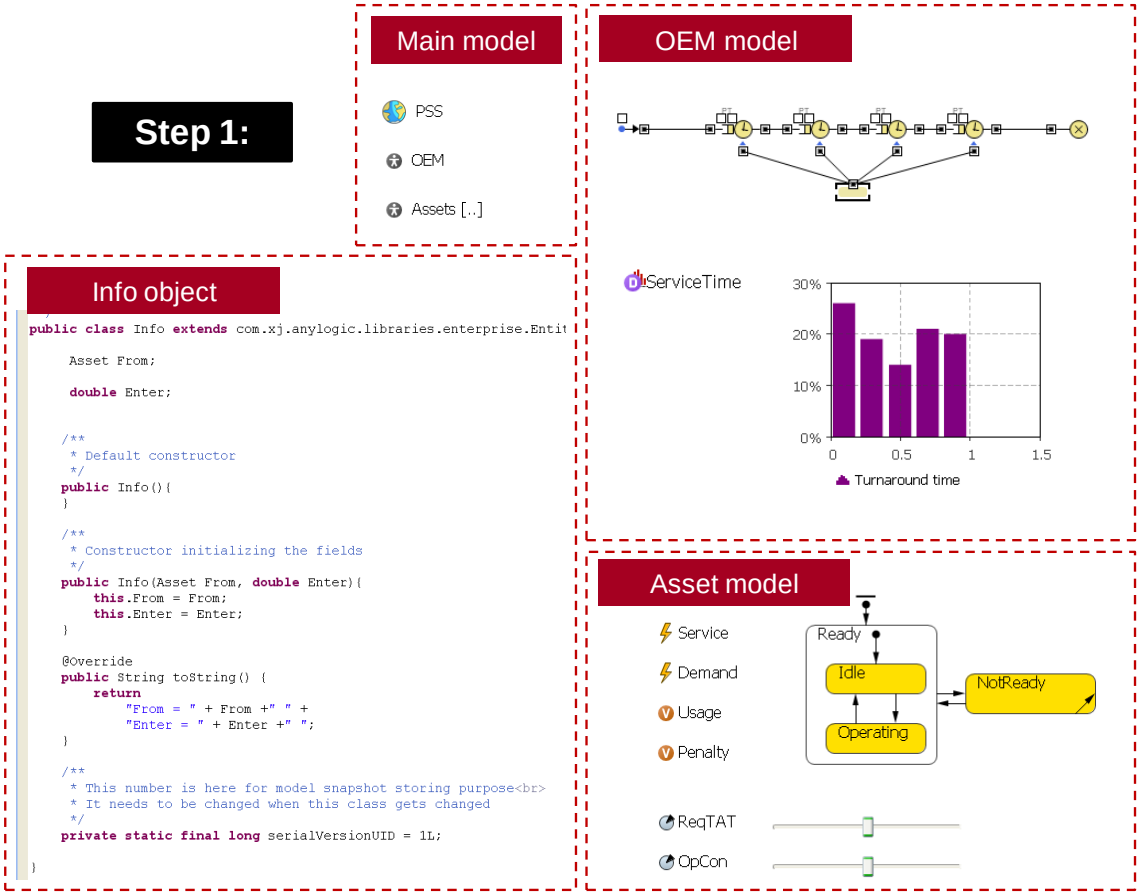


Figure 7-2: Step 1 – EngineCo model

Step 2: Apply service decision making construct

This case exhibits A0 variant as the overhaul service process follows a predefined manual and fixed routine from Gate 0 to Gate 3. Therefore, no change was made from Step 1.

Step 3: Apply subsystem construct

During a flight, an engine's health can be dictated by the LLP condition. If an LLP is cracked within an acceptable range, the engine may still function throughout the flight. However it can happen that the cracked LLP can affect other LLPs and cause them to fail. Therefore, an engine's subsystem exhibits B2 characteristics. In other words, each LLP can influence one another and the entire engine, yet, the interaction cannot be controlled. The following actions were made to the model:

1. The Part agent was created as defined by B2 construct.

2. The *Platform* environment was created, and all LLP agents were placed in that environment.

3. The *Service* event was removed from the Asset agent in B2 asset construct.

4. The interactive attribute was added for the users to capture the service cost of each LLP replacement, and summed up the total cost of all LLPs within the Asset agent.

This step produced the model shown in Figure 7-3.

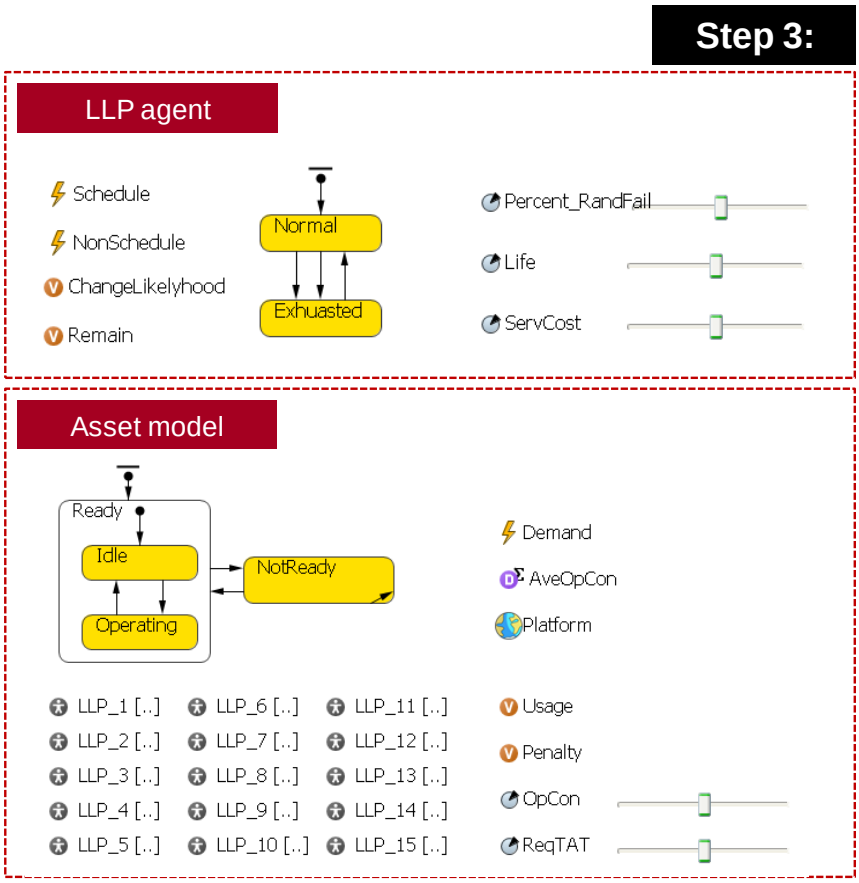


Figure 7-3: Step 3 – EngineCo model

Step 4: Apply work breakdown construct

All contracts only monitor an asset's turnaround time after the service is completed. Thus, this case exhibits C0 variant and required no further action from the previous step.

Step 5: Apply contract creation construct

EngineCo has no predefined policy regarding making new contracts. Accordingly, the case exhibits D2 variant. Users are provided with interactive capability to add a new contract during model execution. The resulting model from this step is shown in Figure 7-4.

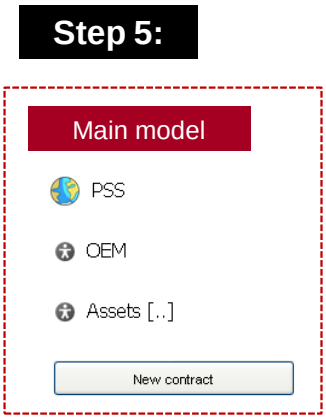


Figure 7-4: Step 5 – EngineCo model

Step 6: Apply capacity adjustment construct

This step embeds the capacity adjustment policy. EngineCo adjusts the capacity through staff working shifts, depending on the maintenance schedule. Therefore user interactive adjustment was applied rather than incorporating adaptive logic. This is because the OEM does not adjust capacity from a current situation but from a planned maintenance. As a result, the number of staff was created as an interactive attribute (Figure 7-5).

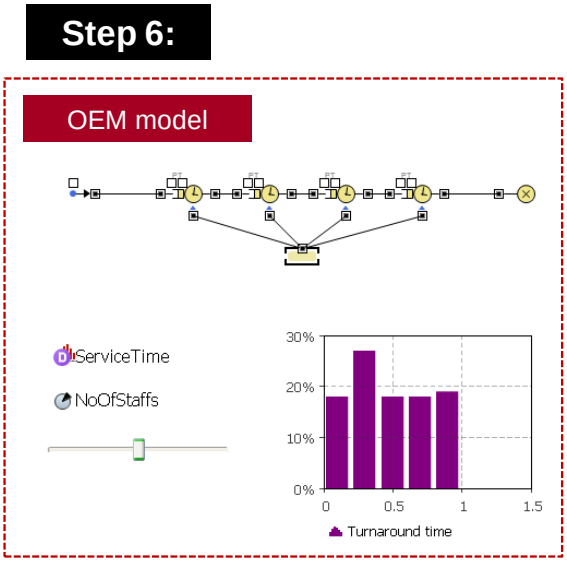


Figure 7-5: Step 6 – EngineCo model

Step 7: Apply contract termination construct

As there has been no case for early termination, this case demonstrates F0 variant and no further modification is necessary.

Step 8: Apply relationship construct

EngineCo can subcontract some activities to other suppliers in case of excessive demand. Therefore, the case represents G1 variant. The following actions were made at this step:

1. The Asset agent's statechart and transitions were modified as defined by G1 construct, presented in Figure 6-20

2. An additional branch for services and its programming logic were created in the OEM agent as defined by the constructs.

The resulting model is presented in Figure 7-6. In the **main model**, the OEM currently provides 10 service contracts which guarantee turnaround time (TAT) to customers.

In the **OEM model,** the OEM currently has 10 servicing staff for all gates. The total servicing period takes around 30 days. If there are more engines waiting to be serviced than available staff to service them, the OEM subcontracts the excessive demands (which will top up the penalty cost).

The **asset model** represents engine states. The customers pay per use of an engine. Demands for flights are randomly generated and the duration of each flight is predetermined. An engine enters the overhaul facility when an LLP is randomly broken or has no remaining life.

To simplify the model, each **LLP** can have from 1 to 20 replications and a life time between 500 and 1000 cycles.

Besides these settings, users can interactively adjust the agreed TAT, pay-per-use fee, operating condition, LLP's life, LLP's service cost, occurrence of random failures, number of staff, and number of contracted engines. Regarding outputs, the model can estimate service cost, penalty, actual TAT, revenue, and number of engines demanded for services, all in real time.

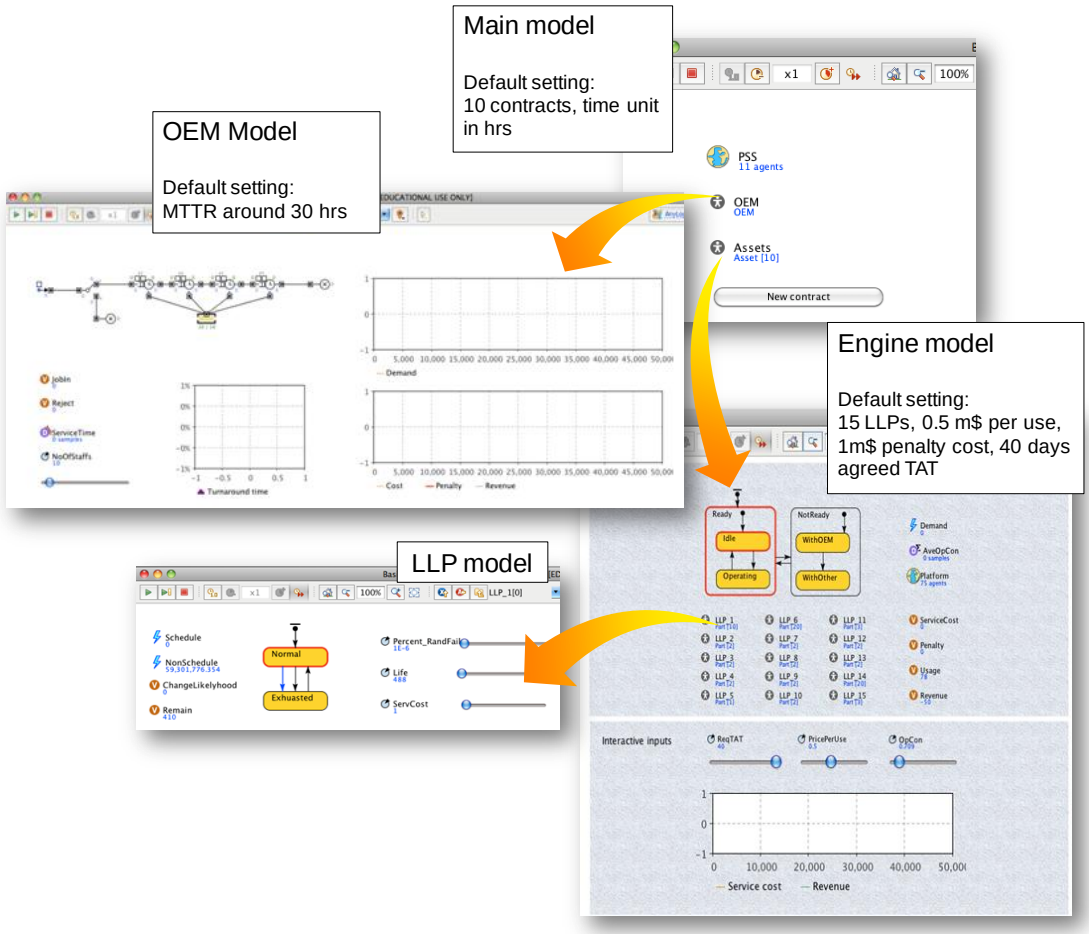


Figure 7-6: EngineCo model

In summary, the case study proved the applicability of the constructs to a great extent. Approximately three quarters of model development time could be shortened using the constructs. This approximation is based on a number of days taken to build the model by the author without using the constructs in comparison with using the constructs. Only few minor elements were amended for practical benefits, which include:

- The required availability was discarded as it is irrelevant to this case. Similarly, the *MissedHrs* variable could be eliminated as the penalty could be calculated directly from the delayed turnaround time.

- Service cost and real time demands were presented in this case, as different operating conditions of the engines can largely influence actual service costs and the overhaul schedule.

- The adaptive capacity could be performed manually by the user for convenience.

- In terms of operations, the maintenance function was broken down into four service elements to capture the four stages.

Practical implications

After the model had been built, it was sent to EngineCo along with instruction to use the model to assist decision making. Four samples of situation were demonstrated as follows:

In the first situation, the company aims to estimate profits between two alternative contracts.

A) The customer pays $0.5m per use on 40 days TAT guarantee and $1m per delay

B) The customer pays $0.7m per use on 30 days TAT guarantee and $1m per delay

An experiment was conducted to compare the alternatives.

To set up the experiments, the *PricePerUse*'s slider and *ReqTAT*'s slider were adjusted to 0.5 and 40 in Experiment A and 0.7 and 30 in Experiment B. After the model was executed, the result is drawn in Figure 7-7.

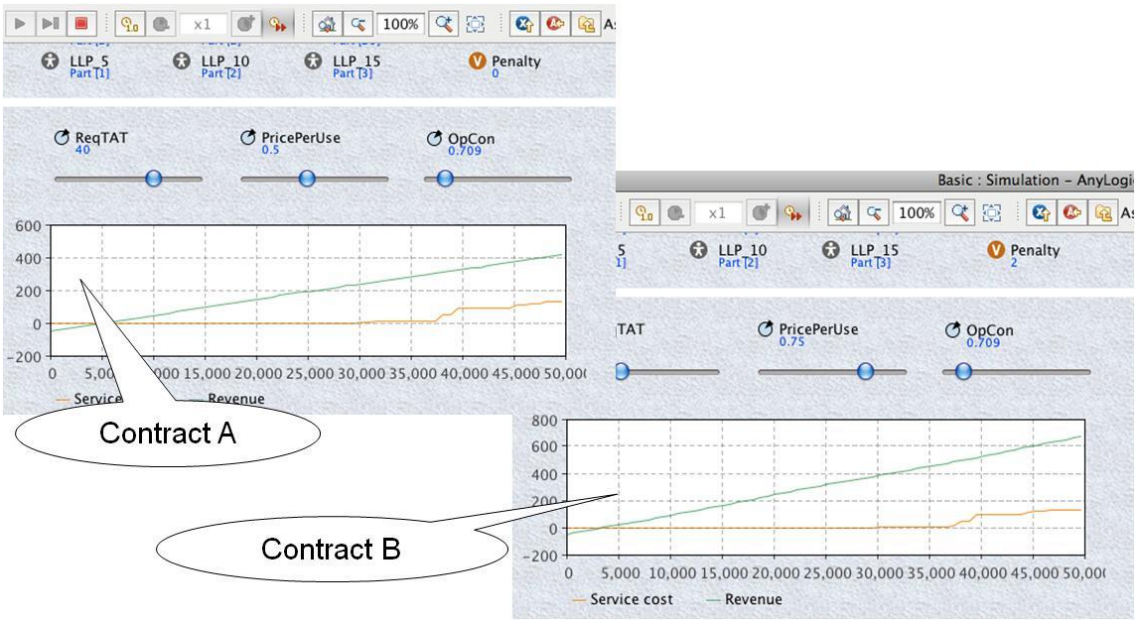


Figure 7-7: Demonstration 1 – EngineCo model

The result indicates no significant difference in service cost and penalty cost but almost double increase in revenue from the second option. Based on this outcome, the OEM may prefer to offer the second contract.

In the second situation, an airline company proposes a contract in which the engine would be used in a harsh environment (twice as a normal condition). Based on this requirement, the OEM estimates the price of this contract.

To set up this experiment, the *OpCon*'s slider was set to two before running the model. After completing the execution, the result is illustrated in Figure 7-8.

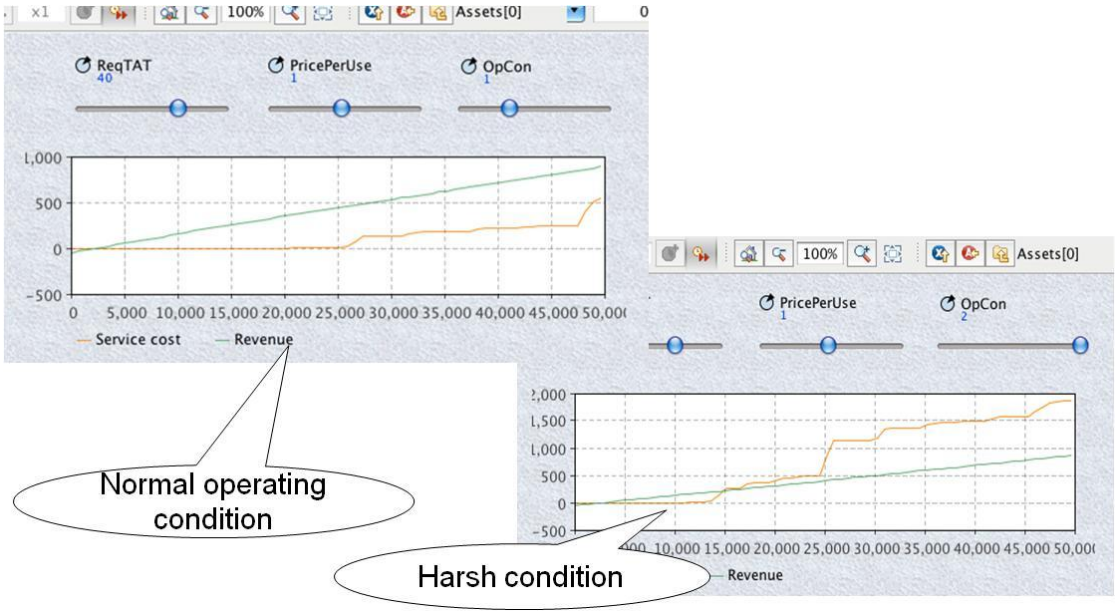


Figure 7-8: Demonstration 2 – EngineCo model

The experiment reveals that the accumulated cost would become four times higher under the harsh condition. Based on this estimation, the OEM may propose the contract at four-times the higher price than the normal contracts.

Thirdly, the OEM is deciding whether to invest in a research and development project. The company is querying if more profit will be obtained if an LLP's life time is extended from 500 cycles to 600 cycles after the first year of contract.

To set up this experiment, the model was run until the simulation time reached around 19000 (i.e. corresponds to approximately 2 years), then, the model was paused. Next,

the *Life* sliders for all LLP_1 agents were moved to around 600. After that, the model was continued until the end of the simulation. The result is presented in Figure 7-9.

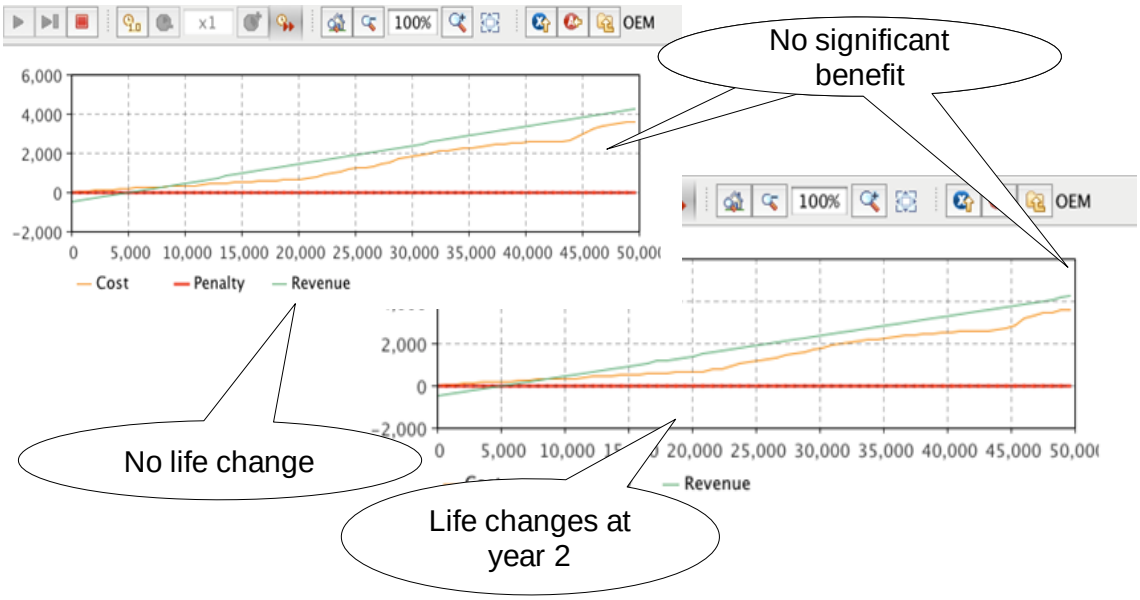


Figure 7-9: Demonstration 3 – EngineCo model

Surprisingly, the result indicates no obvious benefit from the improvement in the life time. Therefore, the company should not waste budget on this project unless there are other factors to consider.

The last situation refers to the growth plan of service contracts. In other words, the OEM is exploring if the current capability would be sufficient to support the growth of 10 new contracts per year.

In this experiment, the *New Contract* button on the main model was clicked repeatedly 10 times at every consecutive 9000 time unit. The result is illustrated in Figure 7-10.

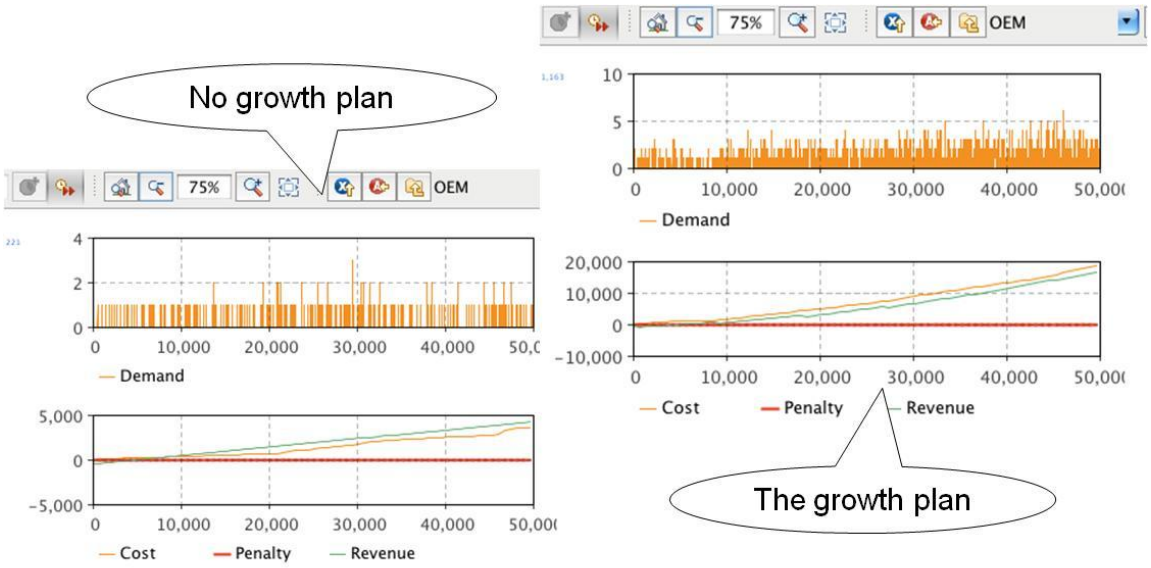


Figure 7-10: Demonstration 4 – EngineCo model

The outcome shows substantial increase in revenue as well as cost. This means the growth plan would lead to a major burden to the OEM despite benefits. Therefore, this plan is not recommended.

These demonstrations relied on one simulation run as it aimed to address potential benefits to EngineCo. In reality, more repeats should be performed to improve confidence in the estimations.

On the whole, these examples demonstrate the model's capability in comparing gains and losses between alternative offers, customising contract from different usage requirements, provisioning investment strategy, and evaluating operational capability against growth plans.

7.1.2 Case II: ShipCo

Data collection process

ShipCo operates its main business in the area of military ship building and provides through-life support to its customers. Three types of service contracts have been offered by ShipCo: after-sales, leasing and output-based contracts. Leasing contracts are made in the long term and guarantee availability of ships (for 25 years), whilst after-sales and output-based contracts generally span over 5 years and provide spares, maintenance, and technical supports at the customers' cost. The output-based contracts differ from the after-sales contracts as the ships are typically designed as specified by each customer. In this study, the model was developed only for the

leasing-type service contract. With this type, ShipCo has offered the contracts since approximately five years ago.

The validation followed an iterative process with the project leaders of service contracting. The interaction involved an introduction of the project, presentation of sample models, and a face-to-face semi-structured interview. The final model and the instructions as to how to use it were emailed to the participants for follow-up feedback.

ShipCo leases a fleet of ships to its customers based on the total required days in a month. Additionally, the company also rents these ships to commercial customers on a short-term basis, when the ships are not in use by the long-term customer. The payment is made on a monthly basis and considered from the actual available days of a ship in comparison with the agreed available days. The operating condition is predefined in the form of locations in which the ships will operate, for instance, 70% operating in the UK and 30% outside the UK. Yet, these numbers, as well as the required available days, can be renegotiated during the contract execution phase. Early termination of a contract has not happened in this case.

A ship is made of several heterogeneous subsystems which influence the maintenance schedule. Similar to EngineCo, the planned maintenance is formulated based on these subsystems' life time. Once the service is due, service engineers perform services as appropriate. The number of these engineers can be adjusted to support the desired utilisation level of the ship. When the level is low, ShipCo may have short-term contracts with commercial customers to increase ship utilisation. If there are more demands than available staff, the OEM may outsource service activities.

Validation outcomes

Step 1: Formulate the basic model

At this step, the following actions were made:

1. The main model was created as instructed by the basic construct.

2. An interactive attribute was added for users to capture short-term demands from commercial customers. Once there is a demand, a signal is sent to any available ship.

3. The information Java object was created.

4. The Asset agents were defined as described in the basic asset construct. However, some modifications were made as follows. Firstly, *OpCon* and *ContractPrice* were created as user interactive inputs. This is because the location where the ships will be operated from and the contract price are specified before making a contract

and can be customised for each contract. Secondly, the actual demands (*LT_demand)* for a ship depend on the agreed available day and the status of the ship. Thirdly, the variable *Available* was added to illustrate actual available days, which is compared against *ReqAvail* to calculate monthly availability (*Availability).* This monitoring activity is triggered by *MonthlyMonitor.* Finally, the *MissedHrs* variable was eliminated as the penalty could be calculated from actual availability.

This step resulted in the model in Figure 7-11.

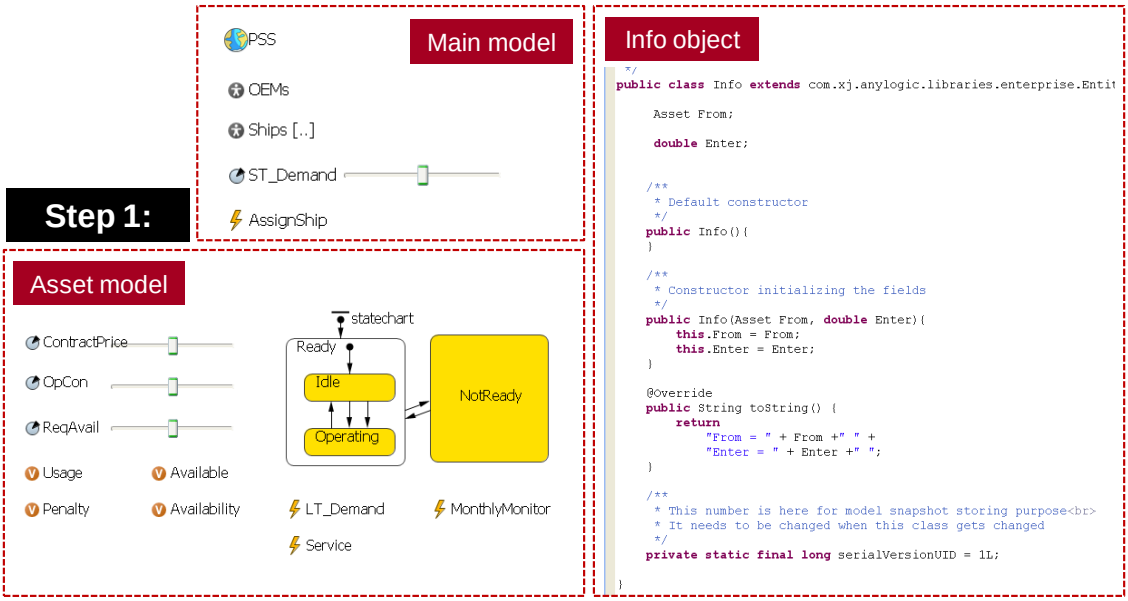


Figure 7-11: Step 1 – ShipCo model

Step 2: Apply service decision making construct

In terms of the service structure variance, this case reveals A2 variant as staff in the ship industry have more flexibility to carry out services. There is no predefined manual to follow, hence, a high variety in productivity. Based on the A2 construct, the OEM model and the staff model were defined as illustrated in Figure 7-12. At this stage, the activity's cycle times within the staff model were created as adjustable attributes to allow users to adjust these values.

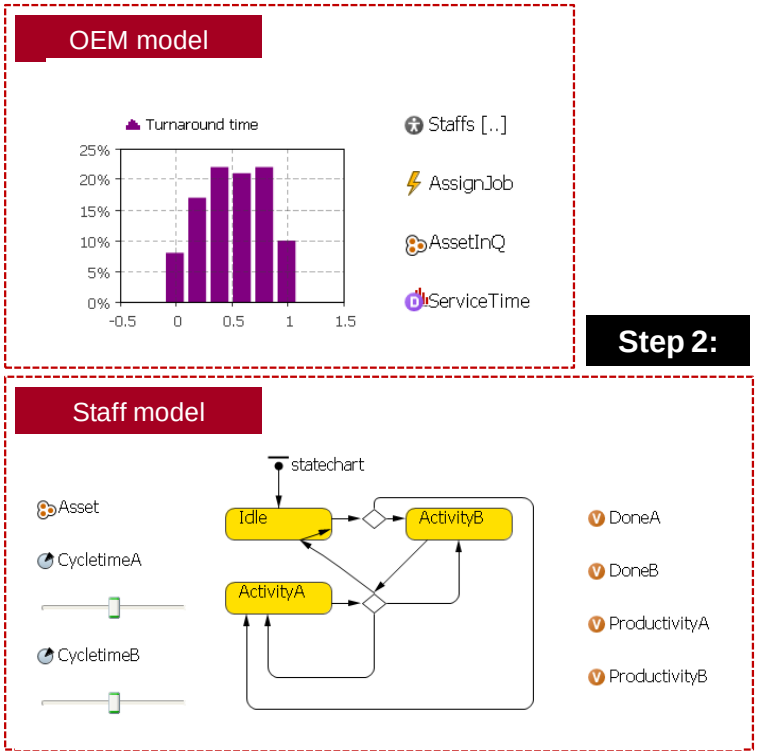


Figure 7-12: Step 2 – ShipCo model

Step 3: Apply subsystem construct

The ship's condition is largely affected by a few major components. Yet, each component does not tend to influence others. Therefore, this case exhibits B1 characteristics. The following actions were made based on B1 construct:

1. The Part agent was created as defined by B1 construct.

2. The service cost was created as an adjustable input.

3. The *Service* event was from the Asset agents which were modified according to B1 asset construct.

4. The revenue and maintenance cost variables were added, including graphs of availability and financial status for the benefit of analysis in the Asset agents.

This step produced the model as shown in Figure 7-13.

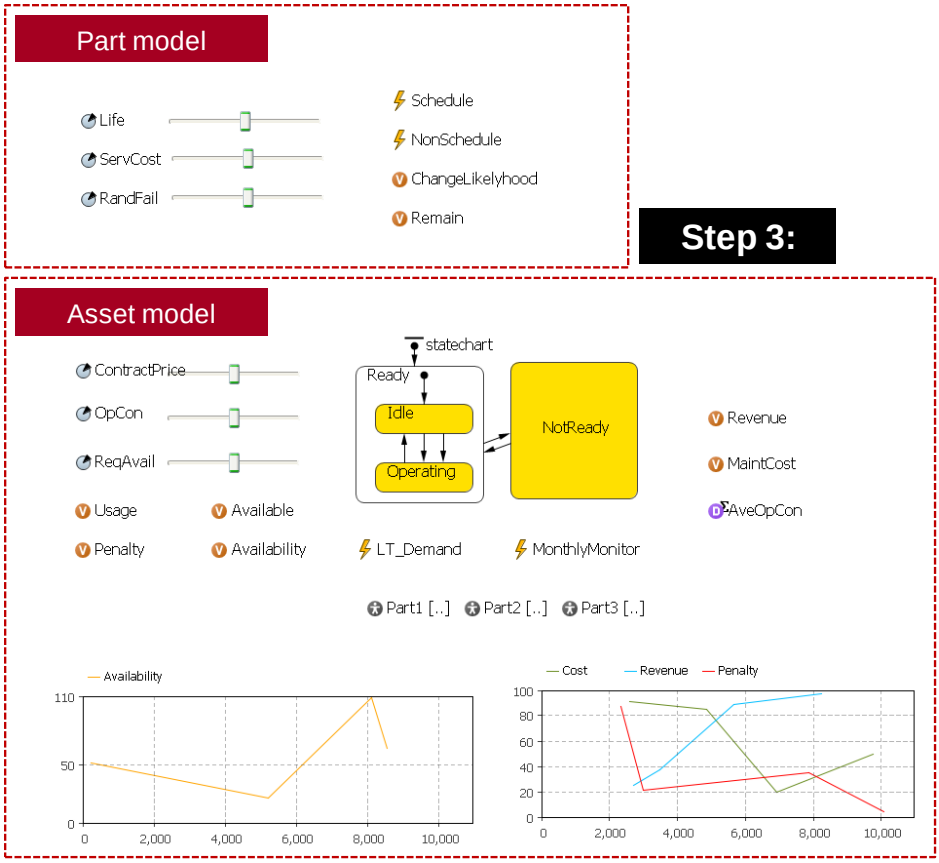


Figure 7-13: Step 3 – ShipCo model

Step 4: Apply work breakdown construct

ShipCo's customers are primarily concerned about available days for operations. Thus, there is no need to monitor a job's progress. This means ShipCo exhibits C0 variant and required no further action at this step.

Step 5: Apply contract creation construct

ShipCo has only one major long-term customer with a leasing-type contract as its main business is scoped down to military surface ships. It is unlikely that new contracts will emerge during the contract period. The emergence of short-term demands from commercial customers was already covered in Step 1. Therefore, no change from the previous step took place at this stage.

Step 6: Apply capacity adjustment construct

Similar to EngineCo, ShipCo also adopt service capacity through staff's working shifts, depending on the major components and actual usage information. Thus, this case reveals E2 variant. Nonetheless, this model was created to enable user interactive adjustment as in E1 variant rather than embedding an adaptive feature in the model's

logic as in E2. This is because E2 aims to capture a demand chasing strategy, yet, capacity adjustments are not an issue for ShipCo. Consequently, it is not necessary to include the adaptive logic. Instead, two buttons were added to control the size of workforce as shown in Figure 7-14.

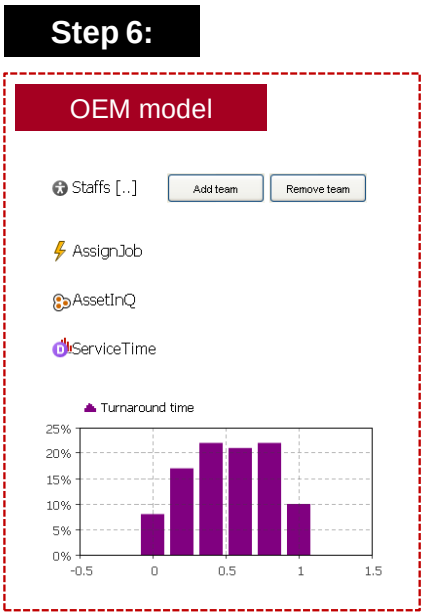


Figure 7-14: Step 4 – ShipCo model

Step 7: Apply contract termination construct

There was no history of early termination at ShipCo. Besides, it is unlikely to happen as the company has been in a strong relationship with the customer (who is a national military organisation) for several years. Therefore, this case demonstrates F0 variant and no further amendment took place.

Step 8: Apply relationship protocol construct

ShipCo can subcontract service operations to other suppliers when there is excessive demand. Therefore, the case represents G2 variant. The following actions were implemented at this step:

1. The Asset agent's state chart and transitions were defined as instructed in G2 construct.

2. The conditional event in the OEM agent was modified as described in the construct.

Figure 7-15: Step 8 – ShipCo model

Step 9: Add analysis elements

This step involves placing additional graphs and variables for the benefit of analysing output data. This includes placing a time colour chart showing each ship status (operating, idle, not ready), and a time plot of the OEM's financial measures (cost, revenue, penalty) in the OEM model.

The resulting model is presented in Figure 7-16. The time unit in this model is in days. The **main model** illustrates an OEM that provides ship leasing contracts. These ships

can also be rented out to other customers on a short-term basis if they are not being used by the contracted customers. Model users can change the rate of occurrence of this short-term demand during model execution by moving the slider of *ST_Demand* e.g. 5 ships a day.

The **ship model** describes each ship's state which could be ready or not ready for operations, influenced by its major components' behaviour. If it is ready, then it can be in operations or waiting for an operation. This is triggered by both short-term and contracted demands. The ship is not ready if it is under service by the OEM or subcontractor. The OEM outsources maintenance services if the maintenance staff are not available. This model monitors monthly performance of each ship in terms of achieved availability level, cost, revenue and penalty, throughout the contract. Users can interactively change the contract prices/charges, operating condition and agreed available days in a month during model execution. These changes can imply contract renegotiation. In this example, each ship is assumed to have three key components (e.g. vessels, controllers, gears), and each component has 3 replications (e.g. 3 vessels).

The spare **part model** depicts each Part agent's behaviour, based on lifetime, failures, and incurred service costs. Users can interactively change these values. Practically, these changes can be caused by redesigns. Ship maintenance takes place if one of its components has no remaining useful life (governed by *Schedule*) or simply fails randomly (controlled by *NonSchedule*). Nevertheless, once the ship is under the maintenance service, the OEM may decide to replace other degrading parts as well. This decision, denoted as *ChangeLikelyhood*, is considered from the remaining useful life of the part.

Within the **OEM model**, risks and rewards from signing the service contracts are visualised from total service cost, revenue, and penalty. The OEM can also monitor operational capability based on the recovery performance (represented as Turnaround time histogram) and the ships' behaviour. The OEM agent assigns jobs to the maintenance team that has the fewest jobs in-hand. It is assumed that the team can complete all required tasks. Users can change the number of teams during the model execution using the buttons. These changes can represent capacity adaptability of the OEM.

The **staff model** encapsulates the difference amongst teams of maintenance staff in performing services. This difference is depicted by a selected sequence of tasks and productivity. The model was simplified to have two maintenance tasks. Once the team receives a job order from the OEM agent, the job is restored in *Asset*. The team can speed up the task depending on jobs-in-hand. Having all tasks completed, the ship is in a 'ready-to-work' state. Users can interactively adjust the activity's cycle time in the staff model as required.

Figure 7-16: ShipCo model

To summarise, ShipCo utilises most elements defined in the constructs. Over half of model development time could be shortened using the constructs. Again, the approximation is based on a number of days taken to build the model by the author without using the constructs compeared to using the constructs. Minor modifications were made due to the following reasons.

- As the ships can be shared by short-term customers, it affects the pattern of operations. Therefore, short-terms demands were included but separated from contracted demands.

- As the user was not familiar with ABS, some input attributes were created explicitly for users to adjust the values during model execution rather than being input implicitly in the model prior to model execution. These attributes include contract price, cycle times, service cost, and workforce size.

- Some elements were added for tracking contract performances and analysis, for example: *MonthlyMonitor, Availability,* graphs.

Practical implications

After the model was built, ShipCo had four weeks to experiment with the model and provided feedback. The instructions to use the model were illustrated and three situations were proposed to demonstrate the use of the model.

The first situation involved an estimation of the potential maintenance cost of a ship, based on the usage requirement and life time of the critical components. The ship is to be used 70% in UK and 30% elsewhere. Currently, the life time of the three critical components are approximately 500, 800 and 1000 operations, and their associated replacement costs are estimated at $0.5 million, $1 million and $1.5 million per part per replacement respectively. The military customer required a guaranteed availability of 10 days for a ship in a month.

To set up this experiment, the *OpCon* slider in the ship model was set to 0.43 (i.e. 30/70), the *ReqAvail* slider was moved to 10, the *Life* sliders of all *Part1* models were set to around 500 and their *ServCost* sliders were set to around 0.5. The given values were also assigned to the Part2 and Part3 models. The result is presented in Figure 7-17.

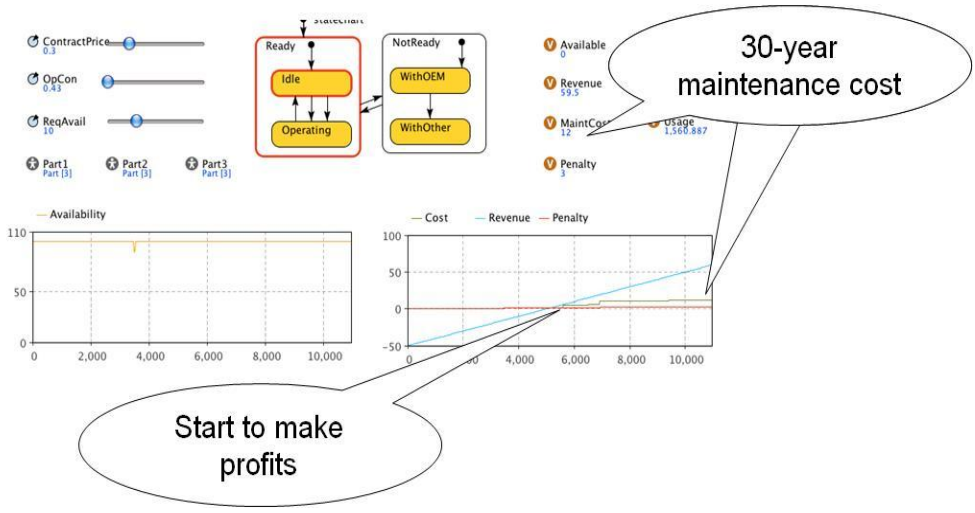
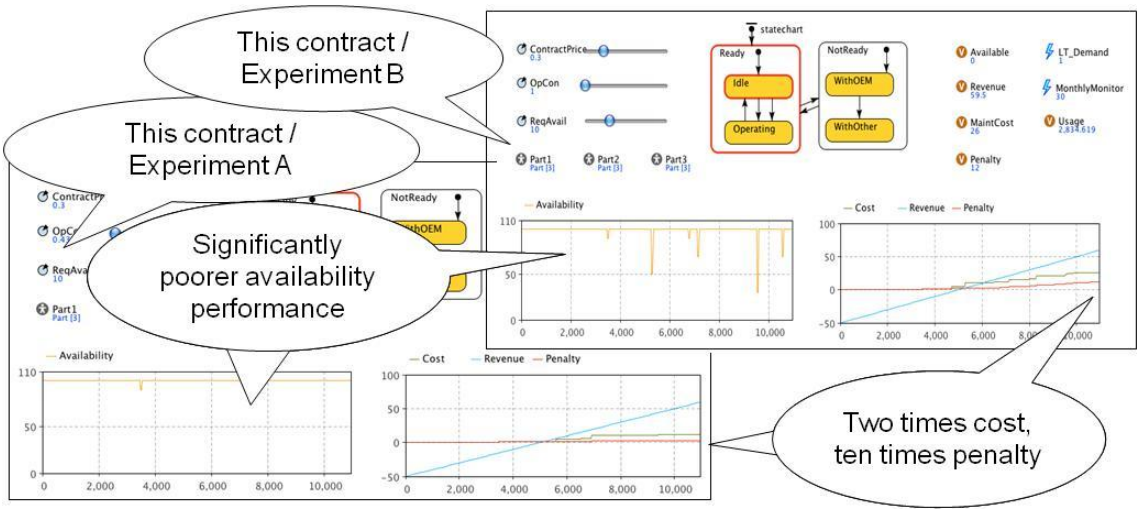


Figure 7-17: Demonstration 1 - ShipCo model

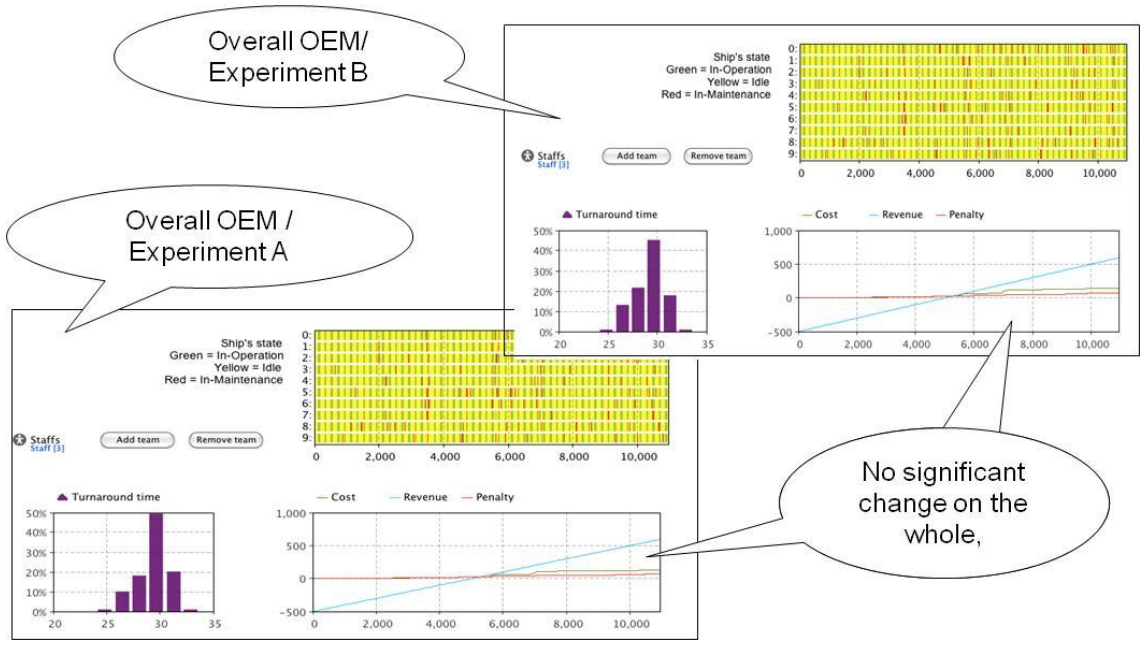
The figure describes the accumulated cost of $12 million at the end of the contract. The OEM can consider this information together with the ship production cost and desirable cash flow pattern for pricing the contract.

In the second demonstration, the OEM can evaluate the impacts if the customer renegotiates the contract from the first experiment to operate the ship 50% of the time in the UK and 50% outside the UK at year 11.

To do this, the model was executed until around 4000 time units. Then, the model was paused to move the *OpCon* slider in the ship model to 1 (i.e. 50/50). Having done this, the model was continued until the end of the simulation. The result (referred to as Experiment B) was contrasted against the first demonstration (denoted as Experiment A), as presented in Figure 7-18.



(a) Impact on contract performance



(b) Overall impact on OEM

Figure 7-18: Demonstration 2 - ShipCo model

The outputs reveal that the change can cause this contract a significantly poorer availability performance, a double cost, and a ten-fold increase in penalty charge. However, it would not significantly affect the OEM's financial status. Therefore, the OEM may not need to be concerned about this risk.

In the last demonstration, the OEM considers giving a discount of $0.1 million per month, per ship to the contracted customer if the ships will also be used by other short-term customers at the rate of 5 ships per day.

To set up this experiment, the *ST_Demand* slider in the main model was set to 5, and the *ContractPrice* slider on all ship models were set to 0.2 (the default setting is at $0.3 million). The result was compared against the first situation (i.e. Experiment A) presented in Figure 7-19.

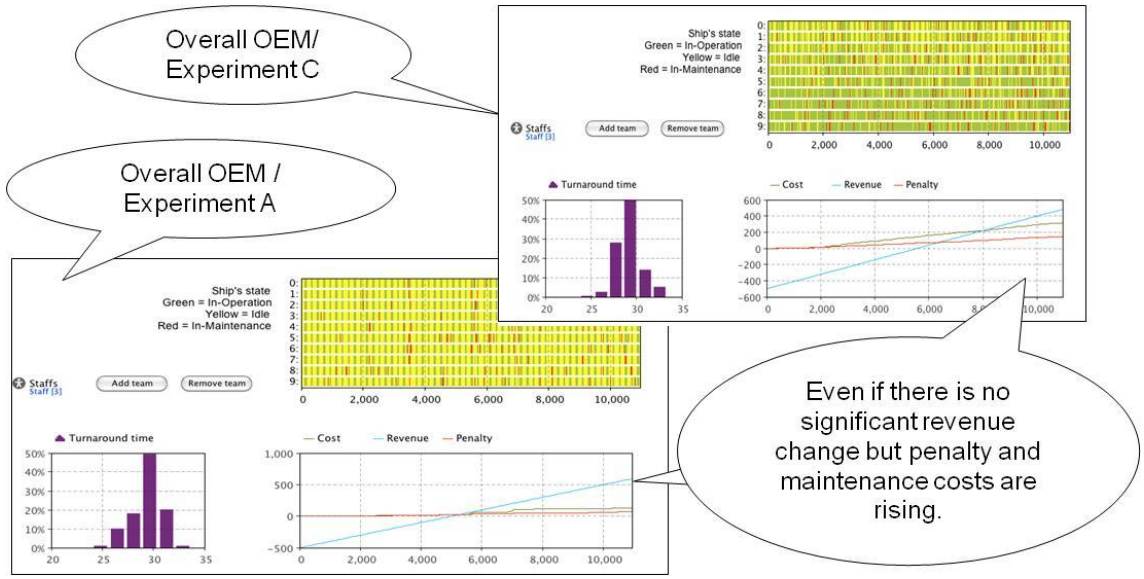


Figure 7-19: Demonstration 3 - ShipCo model

The outputs reveal that the OEM may incur more penalty and maintenance costs even if there is no significant revenue change. Thus, this discount should not be offered.

Overall, the demonstration illustrated the model's capability in estimating cost based on usage requirement and subsystem information, enabling contract renegotiation, and provisioning a marketing strategy. In addition to these examples, users can adjust other inputs, which cover the number of maintenance teams, penalty cost, activity's cycle times, number of contracted ships, a part's random failures, agreed available days, staff's adaptive capability, chance of opportunity fixing, number of major components, and all uncertainties subject to these inputs.

7.1.3 Case III: TrainCo

Data collection process

TrainCo manufactures trains and provides a wide range of services such as maintenance and repair, daily checking, spares and technical supports, reconditioning, fleet management, and cleaning. The company has been in the service business for almost 20 years. Generally, the focus of services is maintenance and repairs.

The validation followed an iterative process with a service engineer. The process involved an introduction of this research, a face-to-face semi-structured interview, presentation of sample models, and follow-up emails for feedback. Again, the follow-

up email includes a set of slides to introduce the user how to use the model for actual decision making.

In the train company, service contracts evolve around maintenance service. Engineers follow an instruction to perform services and each task has a standard time. There are several types of contracts offered by the OEM, depending on the scope of work. However, there is a standard format regulated and applied to all train service contracts.

Prior to making a contract, the train operating schedule is given by the customer, and the OEM will propose the number of trains that should be leased to comply with the given schedule, including spare trains. Availability is monitored every morning if the agreed number is achieved. For instance, the OEM may suggest leasing 10 trains in total, in which 8 trains are required to operate every morning and 2 trains can be on standby. In this case, the availability is 100% if there are 8 trains available for operations in the morning. If less than 8 trains are available or the schedule is delayed during the day, the OEM is penalised.

The OEM can estimate when each train should be retrieved for maintenance based on experience, and sometimes outsource services to meet the demands. However, if the penalty becomes too high, more staff are recruited. In which case, the customer can request to leave the contract. However, there is no policy to lay-off staff or seek for more contracts when staff utilisation is low.

Validation outcomes

Step 1: Formulate the basic model

Unlike EngineCo and ShipCo, TrainCo provides contracts on a fleet basis, monitors performance monthly and has several customers. These characteristics imply that the basic construct must be modified, which can be done in two ways. The first option would be to represent one customer as one Asset agent. The second option, which was adopted in this thesis, was to replace Asset agents with Customer agents in the main model, hence, moving the Asset agents to the Customer model.

Consequently, the following actions were made at this step:

1. The information Java object was created as defined in the basic construct.

2. The main model was created which consists of an OEM agent and Customer agents (there are five customers in this model) in the PSS environment.

3. Adjustable attributes were created for each customer, which include *ReqAvail* and *MonthlyFee*. By doing so, the agreed available trains and the contract's monthly price can be customised to each contract and modified by users at any time. Similarly,

the variables *Availability, Penalty,* and *Revenue* were created to allow monthly monitoring of each contract.

4. The events *DailyMonitor* and *MonthlyMonitor* were placed in the main model to monitor the number of available trains in the morning, penalty, and monthly contract performance (indicated by *Availability, Penalty, and Revenue*).

5. The time plots of availability, profit, demands for maintenance, and a histogram of actual recovery period were placed in the OEM model for the benefit of performance analysis on the main model.

6. The Asset agents were created inside a Customer agent and the *Demand* event was moved from asset model to customer model.

7. The OEM agent was defined based on the basic construct. However, MTTR was created as an adjustable input and moved to the main model so that TrainCo could conveniently apply all values directly to the main model.

8. The Asset agents were defined according to the basic construct with the following modifications. Firstly, *OpCon* is absent in the model as different operating conditions do not significantly affect contract performances in this case. Similarly, *MissedHrs* is neglected as the penalty is directly tied up with the number of available trains every morning (recorded by *DailyMonitor*) and the accumulated delays in a month (*MonthlyPenalty)*. Secondly, *Service* is triggered by a rate function of *TimeBetweeServices* rather than cyclic timer as TrainCo estimates the maintenance cycle as a rate, for example, once every three month. Thirdly, the transition inside the *NotReady* state was removed as TrainCo's customers are not interested in each train's downtime as long as there are available trains for operations as agreed.

This step resulting in the model in Figure 7-20

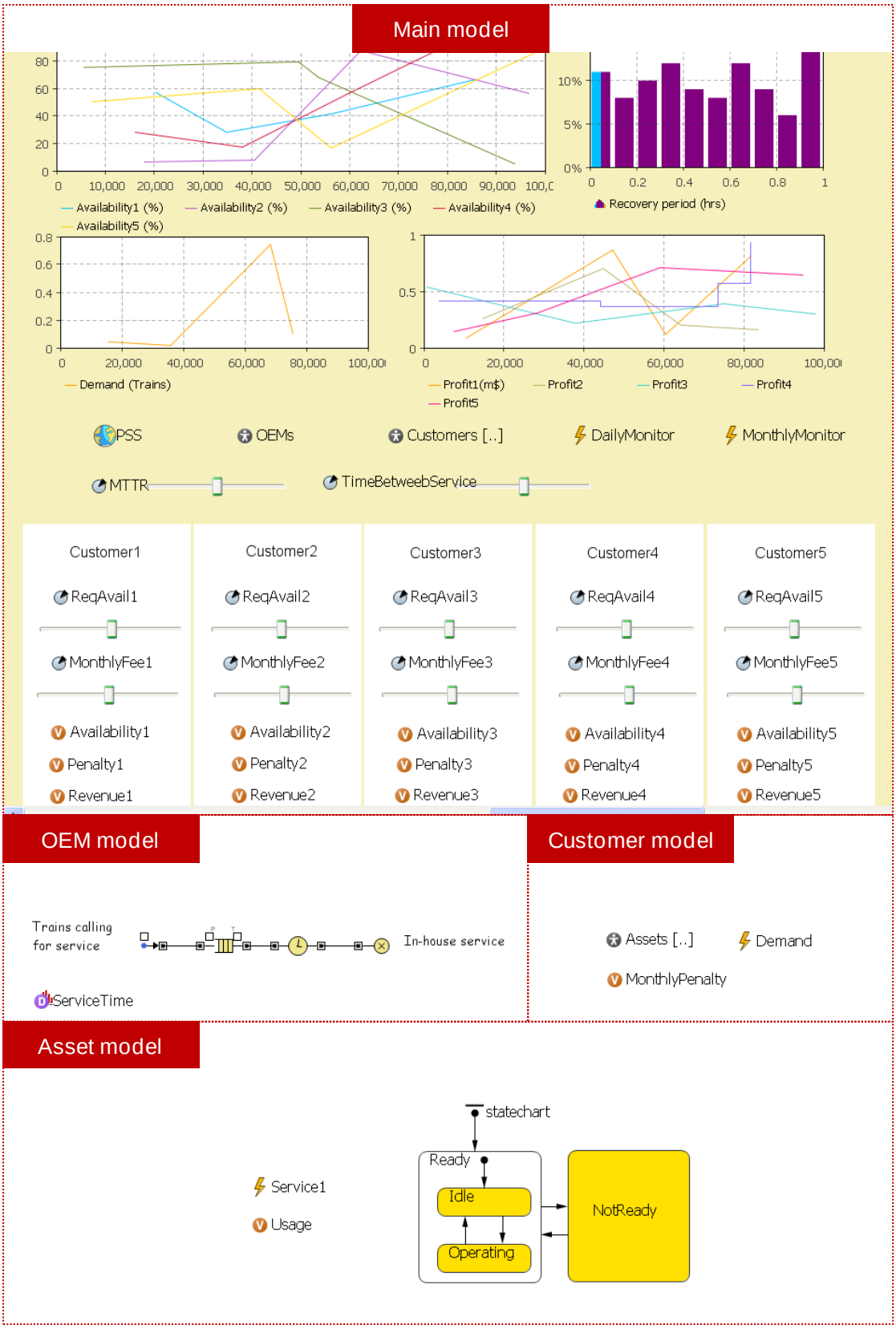


Figure 7-20: Step 1 – TrainCo model

Step 2: Apply service decision making construct

In terms of the service structure variance, TrainCo's staff follow instructions and standard time in performing services and are not allowed to deviate from this. Thus, this case exhibits A0 variant. Consequently, there was no modification at this step.

Step 3: Apply subsystem construct

Even if trains have heterogeneous subsystems, TrainCo can confidently estimate the interval that each train requires between two maintenance services. This means a train's behaviour can be predicted at the aggregate level. Therefore, it shows B0 characteristic at which no further action was necessary at this step.

Step 4: Apply work breakdown construct

As TrainCo's customers are only interested in available trains for operations, it is not necessary to monitor a train's status during maintenances. This means TrainCo exhibits C0 variant, thus, no further action was required at this step.

Step 5: Apply contract creation construct

TrainCo has no strict strategy when the company should seek for a new contract. This can be described as D2 variant, thus, an interactive button was added to each customer in the main model to allow manual contract creation by the user. This resulted in the main model in Figure 7-21.

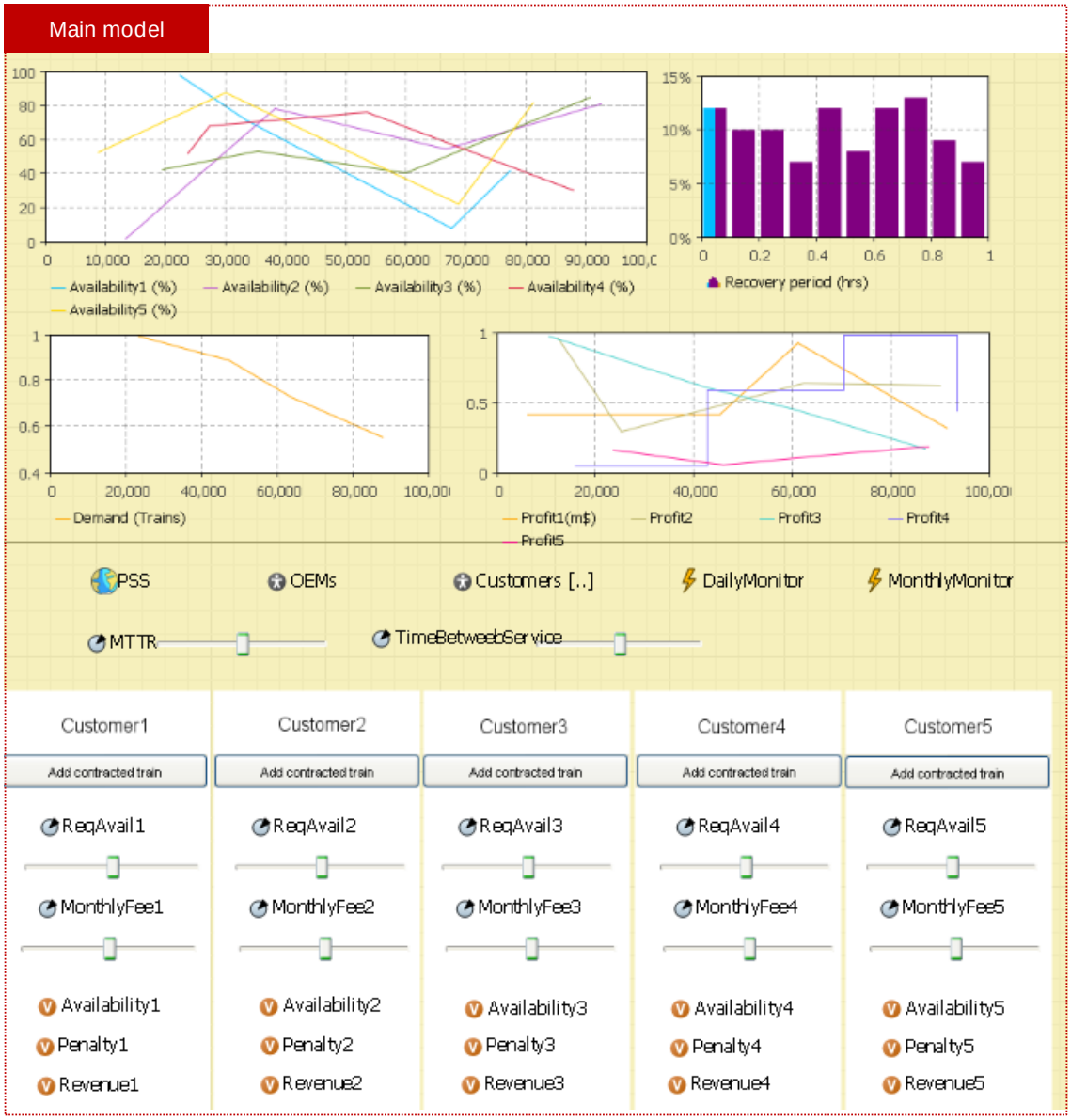


Figure 7-21: Step 4 – TrainCo model

Step 6: Apply capacity adjustment construct

TrainCo does not pursue a lay-off policy when the utilisation is low. On the contrary, the company recruits more staff when the incurred penalty becomes too high. Thus, this case demonstrates E2 variant in which its policy is based on the level of penalty. Nonetheless, the observation during the interview revealed that capacity adjustment is not an issue for TrainCo. Accordingly, this characteristic was absent from the model and no action took place at this step.

Step 7: Apply contract termination construct

Early termination of service contracts from customers happened when they were unsatisfied with the contract performances. Accordingly, the following actions were made at this stage:

1. Followed F1 construct to modify the OEM model and the *Quit* Java object.

2. As the cause of early termination was not explicitly stated during the interview, this model captures this situation via manual command by users. Therefore, two buttons were added in the main model to perform two types of terminations; *Remove a contracted train* invokes an immediate removal of one train in the fleet and *Leave contract* initiates a command to deactivate the whole fleet.

As a result, this step led to the model in Figure 7-22

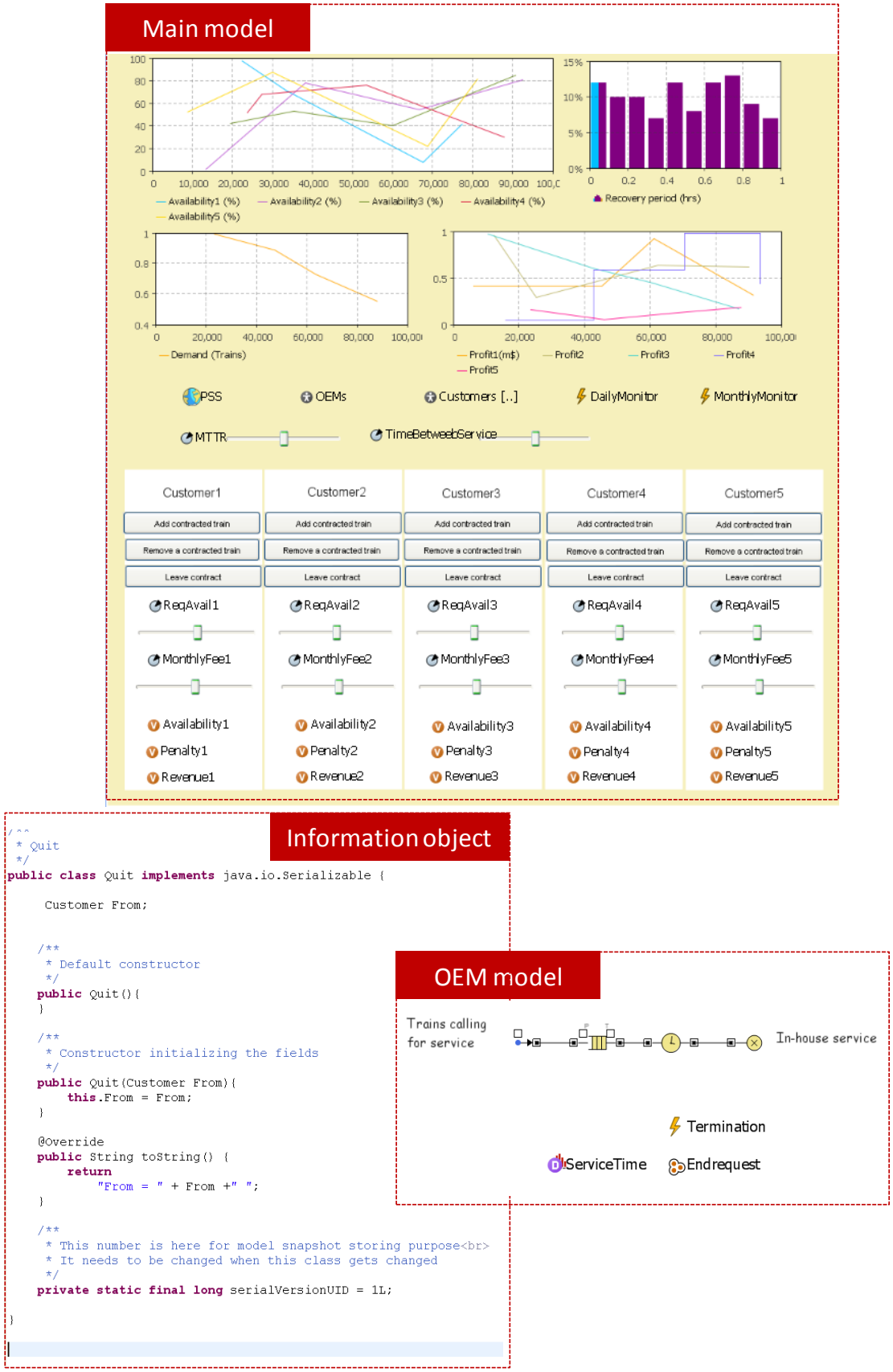


Figure 7-22: Step 7 – TrainCo model

Step 8: Apply relationship construct

TrainCo can subcontract service operations to other suppliers when there is excessive demand. Besides, this case has a basic service decision making structure, therefore, the case represents G1 variant. The following actions were made at this step:

1. The Asset agent's state chart and transitions were amended to G1 construct in Figure 6-20. However, the transition inside *NotReady* state was neglected from the model as the penalty is not considered from an exceeded turnaround time.

2. The OEM agent was modified as described in G1 construct. Nonetheless, the maintenance demand graph has already been presented in the main model. As a result, the graph was removed from the OEM model.

The resulting model is shown in Figure 7-23. The model illustrates an OEM which offers service contracts to 5 customers. Each customer leases 5 trains and signs a maintenance contract with the OEM to ensure availability and on-time operations. In terms of operations, the OEM currently has 5 servicing staff. If there are more trains waiting to be serviced than available staff, the OEM subcontracts the excessive demands. In this default setting, the OEM estimates that each train should require service every 1000 operations. Yet, users can interactively adjust the required availability level, contract's monthly fee, MTTR, time between each service of a train, and the number of contracted trains. Besides, users can terminate any contract within the model execution. The outputs of this model are availability performance, recovery duration (turnaround time), profits, and the number of trains demanded for services.

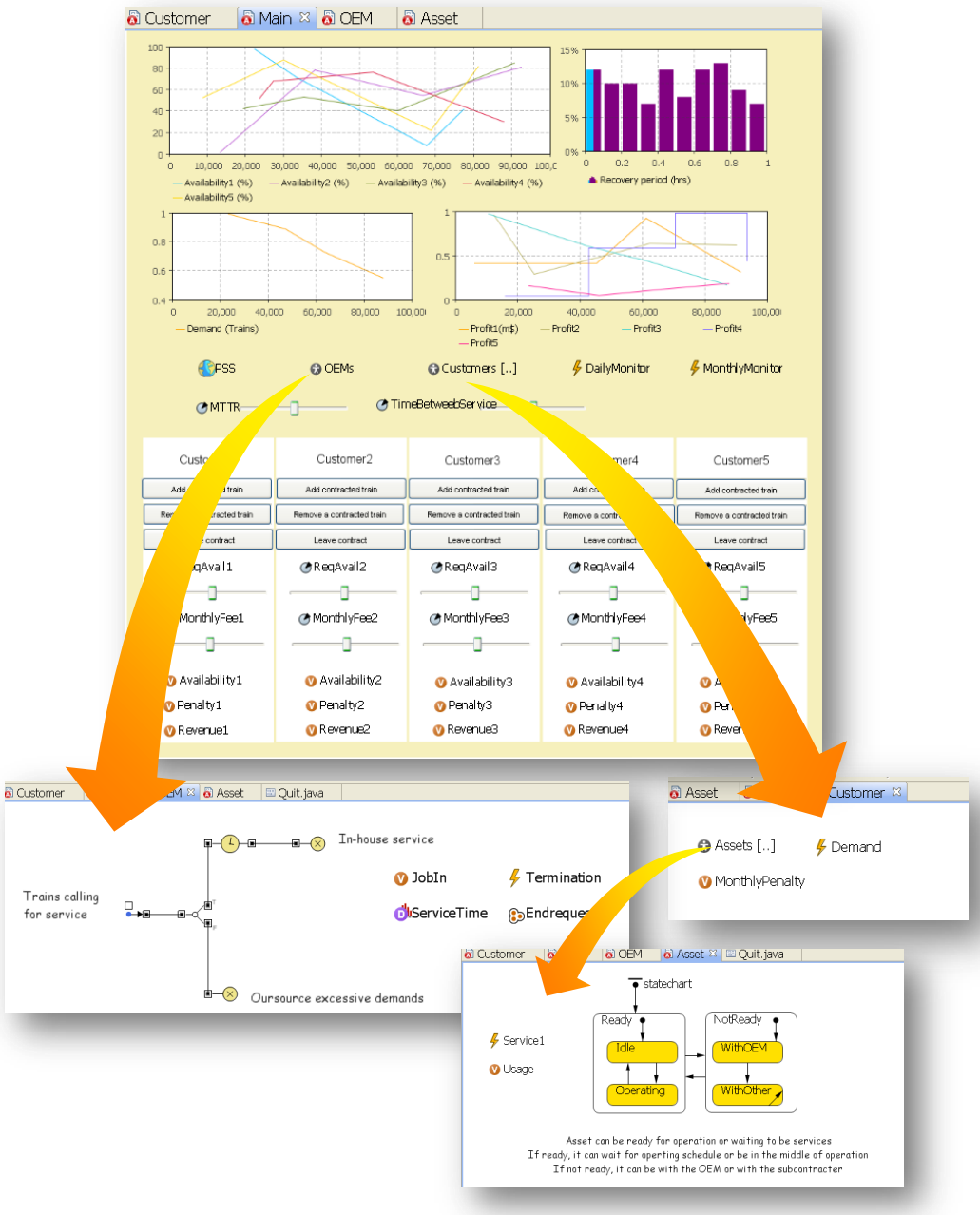


Figure 7-23: TrainCo model

On the whole, TrainCo's model adopts most elements defined in the constructs. However, some of these elements were transferred between sub-models. Overall, approximately one third of the model development time could be shortened using the constructs compared to building the model from scratch. This was also based on the experiment conducted by the author. The modifications can be concluded as follows:

- Some elements were added to the model. Firstly, the fleet contracting scenario led to an addition of the *Customer* agents. Secondly, *DailyMonitor* responds to

the daily availability checking. Lastly, the monthly payments and the review of contract performances called for *MonthlyMonitor, Availability, Revenue, MonthlyPenalty,* and subsequent graphs.

- Some elements in the constructs were moved between the agent models as a result of the presence of Customer agents (*Demand, Asset agents*), and for user convenience in investigating outputs (*Recovery* histogram, *MTTR, ReqAvail*).

- Some characteristics are not major issues for TrainCo, therefore, the associated elements were discarded. These included *OpCon, MissedHrs,* and those associated with the capacity policy variance.

- The way in which the company estimates input affects the method inside the constructs, thus, the model adopts some modifications to suit the user's familiarity. For example, TrainCo generally describes the maintenance's frequency in the form of rate, therefore, the *Service* event is triggered by rate rather than cyclic timer as defined in the constructs.

- Contract termination is captured using a manual input because the trigger was not provided explicitly during the data collection.

Practical implications

Upon the model completion, the model was sent to TrainCo along with instruction on how the model can be used to assist decision making for the company. The company had four weeks to experiment with the model. Three example situations were presented to illustrate the use of the model to contribute in decision making.

Firstly, the situation deals with prediction of contract performance, profits, and losses based on the proposed number of available trains, reliability of services, monthly fee and penalty. In this example, one customer wants to lease 7 trains and for all to be available in the morning, 100% on-time, at a $7m monthly fee, and $1m penalty, whereas the other customers leases and contracts 5 trains at a $5m monthly fee (at default setting).

To set up the experiment, the *Add* button of *Customer1* on the main model was clicked to have 7 leased trains, the *ReqAvail*'s slider was moved to 7 contracted trains, and the *MonthlyFee*'s slider was set to $7m. The model was run until the end of the simulation time.

Figure 7-24: Demonstration 1 - TrainCo model

The overall output (Figure 7-24) demonstrates that the OEM would have one train (out of 27 trains in total) demanded for maintenance, and 70% of the trains demanded for maintenance services would be recovered within 30 hours. Comparing between the two contracts, the 7-train-contract would generate profit more than twice as much as in the other contracts and availability performance is generally better. This example illustrated the capability of the model in customising contract and predicting performance.

The second example aims to expose the capability of the model in capturing a critical dynamic behaviour of PSS – an early termination of contract. The demonstration investigated the result from the customer's sudden termination of a contract at the second year.

To capture this phenomenon, the *Leave contract* button of *Customer1* was clicked when the simulation time was around 15000. The output was compared with the first experiment, as shown in Figure 7-25.

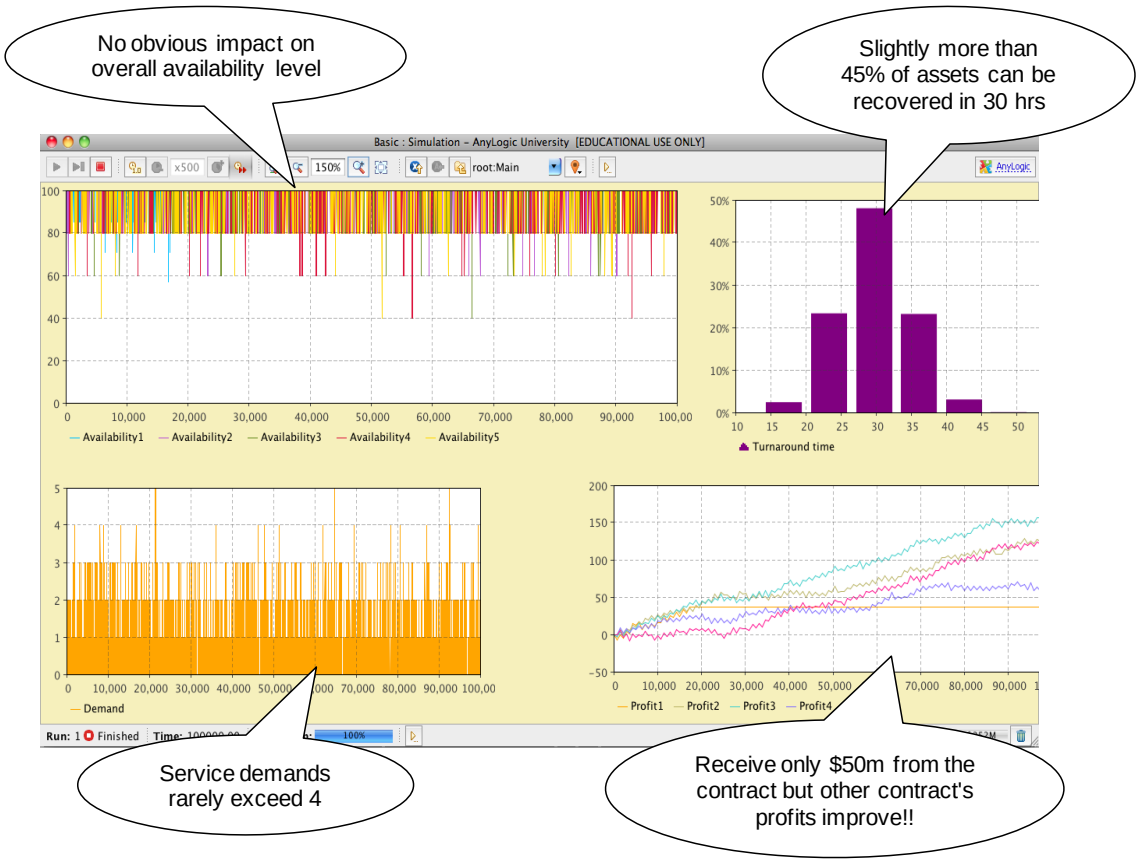


Figure 7-25: Demonstration 2 - TrainCo model

According to Figure 7-25, the result reveals no significant impact on the OEM even if the profit from this contract would be four times less than the first experiment. This is because the profits from other contracts would be improved between 1-2 times than the first experiment, thus these profits ultimately cancel out the effect from the terminated contract.

The third illustration also deals with a dynamic behaviour of PSS. It aimed to address the model's capability in capturing the continuity of making new contracts. In this example, the OEM investigates whether they can cope with the workload if 10 trains were added every year at the additional $1m monthly fee.

To set up this experiment, the model was paused every 9000 time units. Then, the *Add contract* button was clicked twice for every customer and the *MonthlyFee* and *ReqAvail* sliders were moved to 7 (because the default values are 5).

After the execution, the result (Figure 7-26) reveals that the availability and recovery performances were improved for all contracts. Approximately 75% of the trains demanded for maintenance services would be recovered within 30 hours. Profits

would be continuously rose for all contracts. Similarly, the demands for maintenance services are increased.

Figure 7-26: Demonstration 3 - TrainCo model

To summarise, the potential of the model in predicting contract performances, customising a contract based on availability requirements, and encapsulating major dynamic behaviour in PSS (contract termination and creation) during contract delivery phase were highlighted to TrainCo.

7.1.4 Discussion of case study validation

The detailed discussion on applicability and practicality of the constructs were already covered in each case. Next, the efficiency and effectiveness of the constructs at a high level are discussed. The efficiency was defined in Chapter 3 as the capability in shortening model development time, whilst the effectiveness is measured against the

applicability to the real world, the practicality in aiding decision making, and the feasibility in developing a simulation model.

In terms of efficiency, the constructs can shorten model development time between 30%-75% in the three cases. However, model development time is generally be influenced by understandings of the system and modelling experiences of modellers. Therefore, the efficiency of the constructs was also further evaluated from user viewpoints.

As for comparison of the applicability, the EngineCo model applies the elements and methods inside the constructs to the greater extent, followed by ShipCo and TrainCo. This was because EnginCo contracted on an individual basis whereas ShipCo and TrainCo contract on the entire fleet. Besides, ShipCo provides a leasing contract with only one long-term customer, whilst TrainCo has several customers. The level of applicability to each case is also affected by contract requirements. EngineCo guarantees on turnaround time, ShipCo contracts on accumulated available days in a month, and TrainCo is responsible for both daily availability and delays. Therefore, the number of model elements to monitor these requirements increases between the EngineCo and TrainCo models. However, at least approximately half of the elements and methods could be reused in all cases. Regarding the variances from the basic constructs, the three cases share the same work breakdown, capacity adjustment and contract creation variants. To enhance applicability and reduce amendments, these aspects were handled in the final constructs.

The practicality of the models has been evaluated in the demonstration of each case. The feedback validates the contribution of the models in practice. The feasibility can be evaluated from the fact that all models could be developed within a few days by using the constructs. Similar to the efficiency evaluation, the feasibility was also highlighted by the use of a third party in the next section.

7.2 User validation

This evaluation focused on direct users of the constructs. To obtain insights into the feasibility of the constructs, close observation of model developments was required. Additionally, the validation was aimed at generalising the feedback. Accordingly, participants were selected from different levels of simulation background but have involved in the PSS research. The first participant has basic knowledge of simulation techniques, the second participant is an expert in other simulation techniques but relatively new for ABS, and the last participant is an expert in all simulation techniques.

Questionnaires were given prior to and after piloting sessions (see Appendix J). The pre-test questionnaire investigates the participant's background in PSS simulation

modelling whereas the post-test questionnaire aims to evaluate feasibility, practicality, applicability, and efficiency of the constructs. The audit trail of the piloting sessions is provided in Appendix K. Data triangulation was performed by using both direct observations and feedback from the users.

7.2.1 Simulation learner

The first participant is a practitioner in an aircraft company moving towards PSS offerings. This participant has been involved in PSS research for four months and has experience in DES for one month. Therefore, the software package and some applied Java commands were first illustrated. The participant was asked to follow the methodology and apply the constructs to his company, with some help from the author. The whole process, including model completion, took less than two days.

The following results could be drawn from observations:

1. The participant had limited understanding of the case variants and their connection to model elements.

2. The participant had difficulty in understanding some programming languages such as string, double, object. As a result, the author needed to assist in handling technical errors.

3. The participant performed very well in adopting the given Java commands to different situations, despite the unfamiliarity with Java language.

4. The aircraft context exhibits both B1 and B2, thus, some modifications were required. The participant struggled to complete this stage and required the author's help. To cope with this variation, the delay transition inside Part agent was enabled to all connected agents rather than all agents, and a start up code was inserted inside the model's general property to connect subsystems together that can be influenced by one another.

5. The difference between some software elements required further clarification, for example, source and enter, and sink and exit.

Besides the observations, the participant provided the feedback that the constructs were well-presented, reasonably-easy to modify, reasonably-easy to implement, could help the participant to develop the model faster, and had a very good coverage of PSS. The participant showed his interest to use the model in his company and would recommend the constructs for future applications. The areas for improvements were suggested to simplify the description and give clear examples of the case-dependent characteristics.

7.2.2 DES expert

The second participant is a researcher in the area of simulation of prognostic technology for PSS application. He has 3-years experience in PSS and DES. In this validation, there was no introduction of the software. Instead, a document and a brief explanation on the research and the constructs were given. The document comprises a description of the TrainCo case study, instruction on how to use the constructs, examples of adopted Java commands, and the constructs. Based on this document, the participant was asked to develop the model.

This validation was completed in two meetings which took four hours excluding some additional hours that the participant spent with the model himself. During this time, the participant required minor assistance. The observations revealed the following points:

1. The participant still required an explanation of what each agent describes.

2. The participant was unsure how to complete the model and modify the commands.

3. Assistance was needed to cope with some error messages.

4. The participant interpreted the scenario of the case differently from the real scenario.

Additionally, the participant provided the following feedback:

1. The presentation of constructs needed further explanation, for example, the difference between dataset and histogram dataset.

2. The construct could have included contract termination by OEMs.

3. The constructs were very effective in capturing PSS characteristics, very efficient in helping rapid simulation model development, and could help to perform the task very easily.

4. The constructs were reasonably well-presented. The implementation of constructs on a software tool was recommended to enhance more understanding.

5. The constructs were easy to modify, yet, could have been easier if all Java code could be inserted in the model without amendments.

Additionally, the participant saw the benefits of the constructs' presentation that is independent from any software packages, but some of the UML notations were also suggested to use along with the constructs.

In conclusion, the participant stated that the constructs could provide significant contributions in PSS contract modelling, and can be used as modular structures that help users develop the model more easily. They can represent a meta-modelling language which guides users to build agent models and to understand PSS contracts with minimum effort in programming.

7.2.3 Simulation expert

This participant is a consultant well-known in the area of simulation in strategic decision making, and has a 5-year involvement in PSS research. He has extensive experience in using several software packages based on spreadsheet, SD and DES with speciality in ABS. During the validation process, a document which contains a description of the ShipCo case study, instruction on how to use the constructs, Java commands adopted in the constructs, and the constructs, was given to the participant. It was aimed that the participant could complete the model without any help. As a result, the model was completed in approximately one day.

After the completion, the feedback was obtained as follows:

1. The constructs were very easy to use but would be easier for general users by presenting them as a drag and drop modelling solution or a building block approach in which these agents are pre-defined and therefore become configurable through simple modification or input data.

2. The constructs could help to develop the model more quickly. The participant described that it is a very good step towards configurable modelling for contract negotiation and asset optimisation in a PSS environment as it can increase visibility in the negotiation processes. He highlighted that the constructs can be commercially delivered as 'simulation as a service' online to enable further customisation and comparison between different potential options.

3. The constructs could capture PSS characteristics reasonably well. It was also suggested to implement cost modelling in the constructs.

4. The constructs could have been presented more clearly by systematically explaining them agent by agent. A balance scorecard may be adopted to enhance clarity of performance indicators.

5. The modifications to the model were not relatively simple and it could be too complex for users who are not familiar with the software package. Recommendations were made on implementing more user interactive inputs and scenario configuration. The input screen could have been made separate from the model.

On the whole, the participant was impressed by the generality and capability of the constructs in capturing major aspects of service contracts. He also declared the merits

and novelty of the constructs and was keen to follow and recommend it to others in the future.

7.2.4 Discussion of user validation

The validation revealed a wide range of opinion, which was most likely caused by their different experiences in developing ABS models. Although they have all involved in PSS research, all participants had never come across any existing methodology which enables comparison across PSS offering alternatives using simulation. Moreover, all participants acknowledged the potentials of the constructs and would recommend them for future use.

In terms of understanding of the constructs, this depended highly on the simulation background of each user. However, the author attempted to enhance the understanding later in the final constructs by providing an overview of the construct and their implementation in a software package. The implementation identifies both screenshots and programming code agent by agent. This should also improve confidence in completing the models.

Regarding the selection of the variants, the user's interpretation could be influenced by the given explanation to a large extent. However, it should be noted that the direct users of the constructs are the OEMs who also have simulation modelling background. Therefore, the issue would be resolved once the description of the constructs becomes clearer.

The suggestion to include contract termination by OEMs could be implemented in the construct, and is detailed in the next section. Nevertheless, the cost aspect was too time-consuming to be included in the scope of this research. Similarly, the capability to implement the constructs as a drag and drop modelling solution or a configurable building block / input screen can be explored in future work.

Overall, despite the fact that the constructs may still be complicated for some users, this validation revealed that no participant took longer than two days to develop the models. This implies a high feasibility of developing a model from the constructs. The feedback from all users also supports the efficiency of the constructs in rapid model development. The simulation learner also performed very well in adopting Java commands without previous knowledge of the Java language. And in fact, the complexity indicates the extensive efforts required in developing service contract models without using the constructs, thus, it highlights the significance of this research.

7.3 Refinements of the constructs

This section deals with the amendments of the constructs after being evaluated by both case study validation and user validation.

The evaluation suggested two major areas of improvements; the presentation and the elements/methods inside constructs. The improvement of the presentation aims to provide better understanding of the developed constructs, especially when further customisation is required. The elements and methods were amended to improve the ease of use.

From the presentation viewpoint, the constructs were amended to provide model elements and higher level modelling methods, so that users can understand the 'big picture' of the modelling mechanism inside the constructs. In addition to the constructs, a user manual, and examples of detailed programming are included to support the use.

In terms of elements and methods, major changes were made as follows:

- The *basic construct* was amended to become the *shared construct*. The basic construct enables a user to complete a model. Further modifications are made by changing code as well as adding and removing elements (or code). This caused confusion to some users, and therefore, the shared construct was developed so that users can only add elements and methods for further case customisation.

- For generalisation, user interactive control was provided to adjust capacity, make new contracts, and terminate contracts. Short-term demands are always included in the constructs. Users can disable these functions if they are absent in the case. A monthly fee, penalty, service cost, and MTTR are adjustable by control sliders. Customer agents are presented in the shared construct to allow easy modelling of a fleet scenario.

- Subcontract options were removed as capacity becomes adjustable during execution.

- Contracting unit (fleet versus individual), value parameters (availability-based versus time-based), and payment mode were added into the case-dependent characteristics.

- B2 variant is removed as the influence between Subsystem agents is difficult to define in practice.

7.4 Chapter summary

This chapter detailed the evaluation process on the constructs developed in the previous chapter. The evaluation was made through three case studies and tested externally by three users. Feedback was taken from the three case interviewees, the three users, and the author's observations. Overall, all practitioners and users clearly saw the potential and benefits from using the models, and were keen to apply the constructs in practice. However, for some users, the constructs were not easy to implement and modify. This led to improvements to the final constructs presented in the next chapter.

8 Presentation of the final constructs

This chapter describes the final constructs. Section 8.1 provides the overview of the constructs and instruction on how to use them. Section 8.2 describes the shared constructs and Section 8.3 deals with the case-dependent constructs. These constructs can be implemented in any software package that supports a hybrid ABS-DES technique. Finally, Section 8.5 summarises this chapter.

8.1 Overview of the constructs

The final constructs follow the same structure as the primary constructs. The constructs consist of two parts: the shared construct provides common modelling elements across service contracts and the case-dependent constructs contain elements that can be different across cases. These elements relate to service decision making, subsystem, work breakdown, and contractual mode. A construct is given in correspondence with each variant. The constructs can be summarised into Figure 8-1.

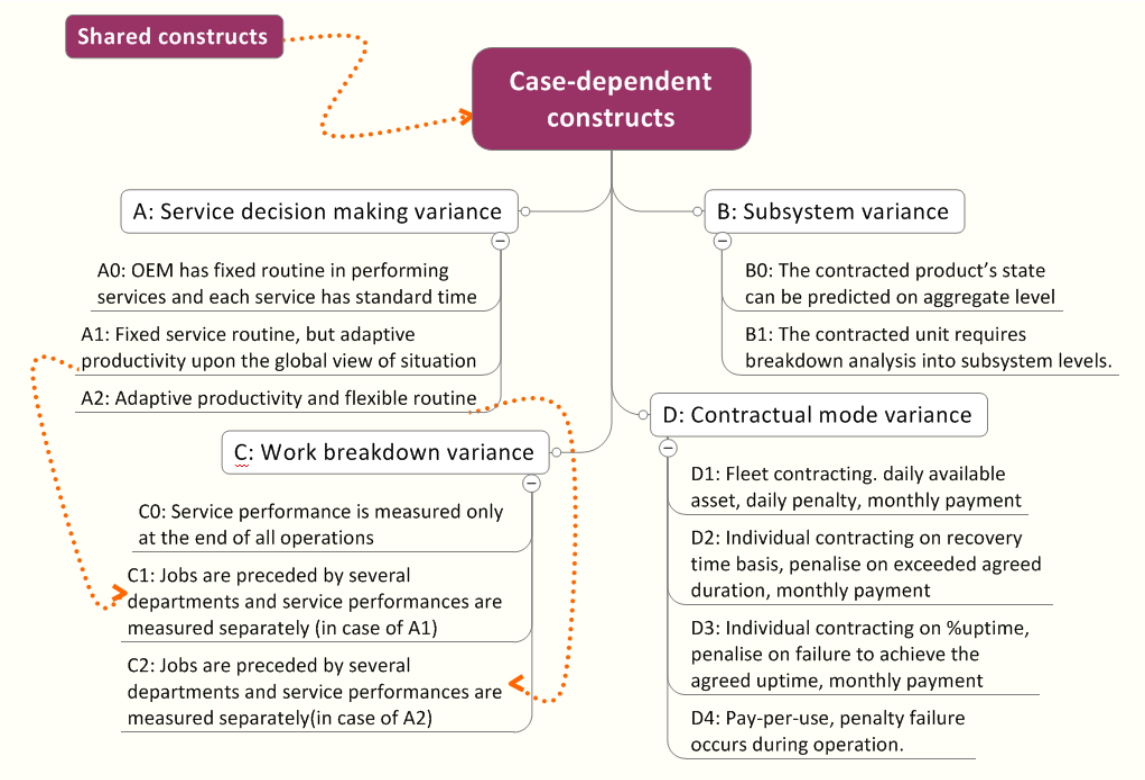


Figure 8-1: Overview of the constructs

Figure 8-2 illustrates how to use these constructs to build a service contract model.

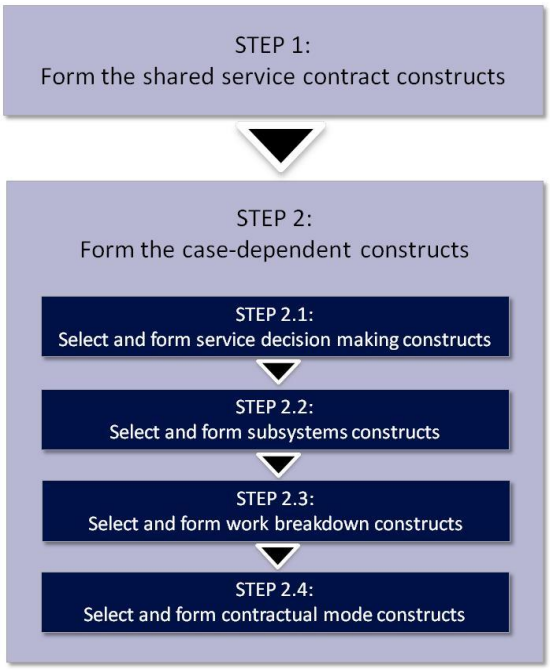


Figure 8-2: The steps in building a model using the constructs

1. The first stage is to develop *a shared service contract model* from *the shared service contract construct*.

2. The second stage is to customise the model using *the case-dependent constructs*. At this stage, the characteristics of the case are mapped with the characteristics identified by the case-dependent constructs.

Users are required to complete each variance (step 2.1 – step 2.4) to get to the final model. The variants within each characteristic variance are summarised in Figure 8-1.

The next section describes the shared construct.

8.2 The shared construct

The shared PSS elements are associated with asset health and life cycle, OEM service process, in-service asset information, service efficiency measures, and usage unit. To account for these PSS elements, the shared construct consists of three layers and encapsulates five fundamental model components.

The first layer contains the following components: *OEM agent* and *Customer agents* in a *PSS environment*. The second layer details the OEM service process inside the OEM

agent and encapsulates *Asset agents* inside the Customer agent. The third layer presents the asset's state inside Asset agent. The last element is *a Java object* which represents the asset information passed to the OEM prior to servicing. This information object refers to the communication method in the construct which depicts an OEM-customer relationship in PSS business.

The first layer is represented by **the main model** (Figure 8-3) which describes an OEM agent who signs PSS contracts with Customer agents. The OEM may also use the assets for short-term demands (*ST_Demand*) if they are not used by contracted customers. This mechanism is governed by *SearchFree.*

The **OEM agent** describes list of activities; an asset enters the OEM system, waits for service, is serviced if there is available staff, and returned to the customer once finished. The OEM always records the number of assets in the system (*JobIn)* and the time each asset stays in the system (*ServiceTime*). Users can adjust *MTTR* and the number of staff (*Capacity*) at anytime.

The **Customer agent** contains contracted assets. Customers may renegotiate to have more or fewer assets during the contract delivery phase.

The **Asset agent** shows the asset's state in the life cycle. It can be ready for operation or not. If it is ready, it can be in operation or waiting for an operation. The asset's *usage* is always recorded. Customers may renegotiate to change the agreed operating condition (*OpCon*) of an asset during the contract delivery phase.

The **information object** stores asset information and enables the OEM to record the time it enters the service process.

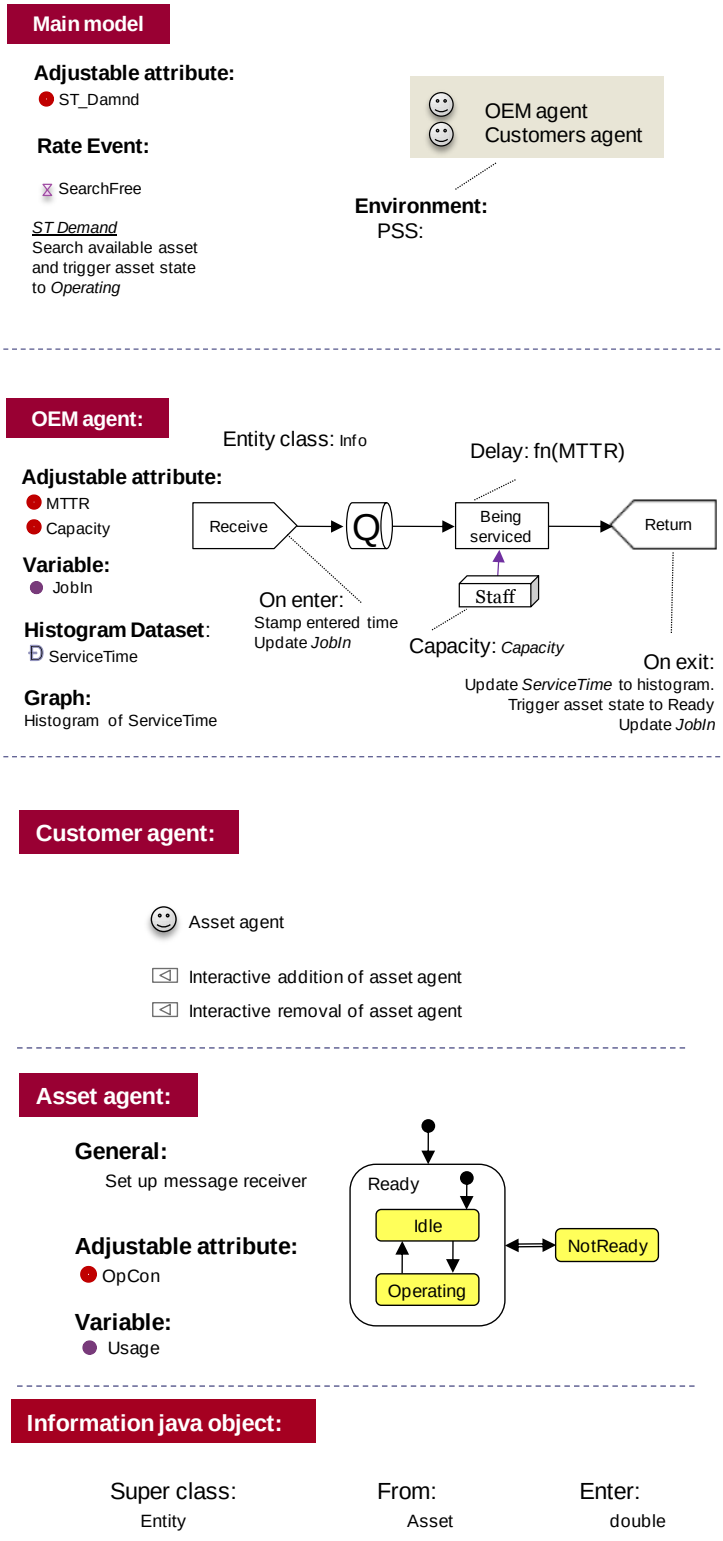


Figure 8-3: The shared service contract modelling construct

8.3 The case-dependent constructs

The case-dependent constructs result from different case characteristics dictated by the following PSS elements:

- Service decision making constructs
- Subsystem constructs
- Work breakdown constructs
- Contractual mode constructs

The variants within these variances are summarised in Table 8-1.

Table 8-1: Summary of the case-dependent constructs

Characteristic variance	Characteristic variants	Construct variances (from the shared constructs)
Service decision making	A0: OEM has fixed routine in performing services and each service has standard time	Input distribution is encapsulated in the OEM's delay element and communication protocol is establish between asset and OEM agents
	A1: Fixed service routine, but adaptive productivity upon the global view of situation	There are two levels of inputs in the OEM's delay element. Communication protocol is establish between asset and OEM agents
	A2: Adaptive productivity and flexible routine	Staff are created as another agent under the central OEM. Communication protocol is establish among Asset, OEM , and Staff agents
Subsystem	B0: The contracted product's state can be predicted on an aggregate level	Servicing schedule is monitored periodically via a cyclic timer
	B1: The contracted unit requires breakdown analysis into subsystem levels.	Subsystems are created as another agent under the Asset agents
Work breakdown	C0: Service performance is measured only at the end of all operations	Same as the shared construct
	C1: Jobs are preceded by several departments and service performances are measured separately (A1)	Jobs are created as a separate agent issued by the Asset agents to track different performance requirements
	C2: Jobs are preceded by several department and service performances are measured separately (A2)	Based on A2 structure, Jobs are created as a separate agent issued by the Asset agents to track different performance requirements

Characteristic variance	Characteristic variants	Construct variances (from the shared constructs)
Contractual mode	D1: Fleet contracting Daily available asset Daily penalty Monthly payment	Contract requirements, performance measure, risk and reward are captured. Three cyclic timers represent daily monitoring, monthly monitoring, and asset operating activities. Transitions captures asset's operating state.
	D2: Individual contracting on recovery time basis Penalised on exceeded agreed duration Monthly payment	Contract requirements, risk and reward are captured. Two cyclic timers represent monthly monitoring and asset operating activities. Transitions captures asset's operating state and penalty mechanism.
	D3: Individual contracting on percentage uptime or available days Penalise on failure to achieve the required level Monthly payment	Contract requirements, performance measure, risk and reward are captured. Two cyclic timers represent monthly monitoring, and asset operating activities. Transitions captures asset's operating state and % uptime of the asset.
	D4: Pay-per-use Penalise if failure occurs during operation	Contract requirements, risk and reward are captured. Cyclic timer represents asset operations. Transitions capture penalty mechanism and asset's operating state.

Unless stated and highlighted in black, the construct is identical to the shared construct. Users can also further detail this final model to suit their cases.

8.3.1 Service decision making structure

A0: OEM has fixed routine in performing services and each service has a standard time.

For example, overhaul services always cover 4 successive stages and each stage always takes around 1, 3, 20, and 4 days respectively.

In this case, the transitions and communication are defined within the Asset agent (Figure 8-4) to allow service interactions between OEM and Asset agents.

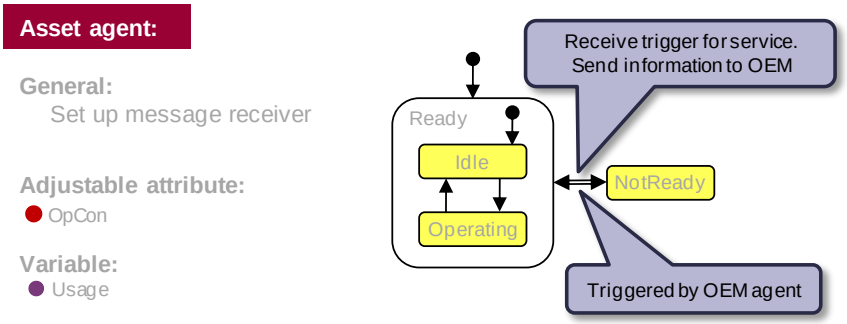


Figure 8-4: A0 construct

A1: Fixed service routine, but adaptive productivity upon the global view of the situation

For example, disassembling a wheel always includes four executive stages, and staff generally take 10 hours to disassemble wheels for a car of a train, yet, they can take only 5 hours if there are other trains waiting to be disassembled.

Minor modification to the shared OEM construct can represent this variant (Figure 8-5). The cycle time is self-adjusted upon the overall workload. This implies that the staff are aware of the queuing jobs in the system at the beginning of each operation, and adopt their productivities accordingly. Here, the parameter *Adaptive* depicts the baseline of the number of jobs the staff often perceive as a high workload. Within the Asset agent, the transitions and communication are defined to allow service interactions between OEM and Asset agents.

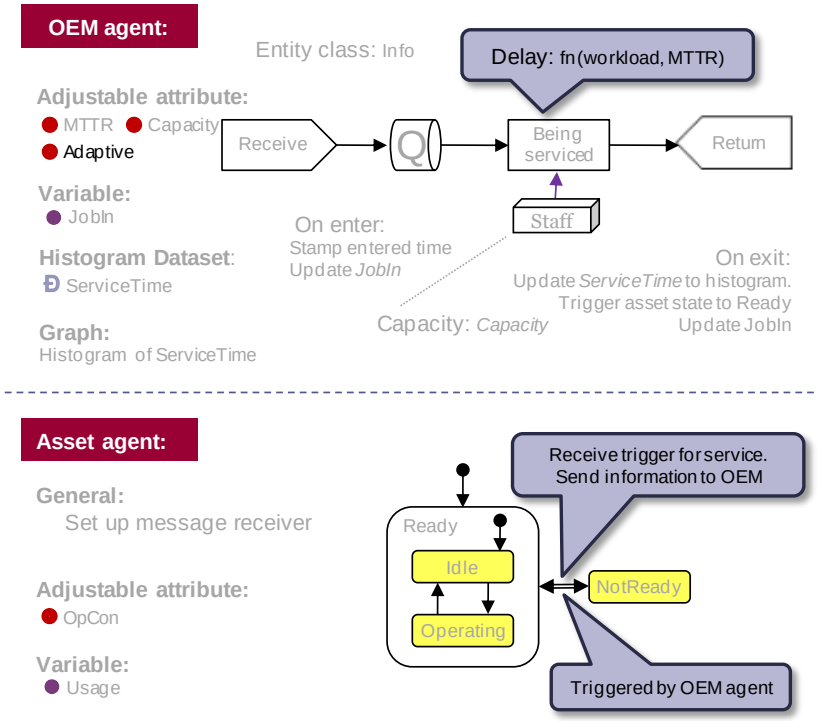


Figure 8-5: A1 construct

A2: Adaptive productivity and flexible routine

For instance, within a photocopier context, field-service staff can check a photocopier's condition or refill papers/ink first. There is no predefined rule what to perform first.

As the staff become more autonomous and decentralised in this case, they are created as agents embedded inside the OEM agent (Figure 8-6). Within the Asset agent, the transitions and communication are defined to allow service interactions between OEM and the Asset agents. Asset agent notifies the OEM agent for the service. The OEM agent registers the request in *AssetInQ,* and activates the *AssignJob* to allocate the job to field staff based on the staff's workload. The staff have flexibility to select a sequence of tasks. Once completed, the asset is approved for operational-ready.

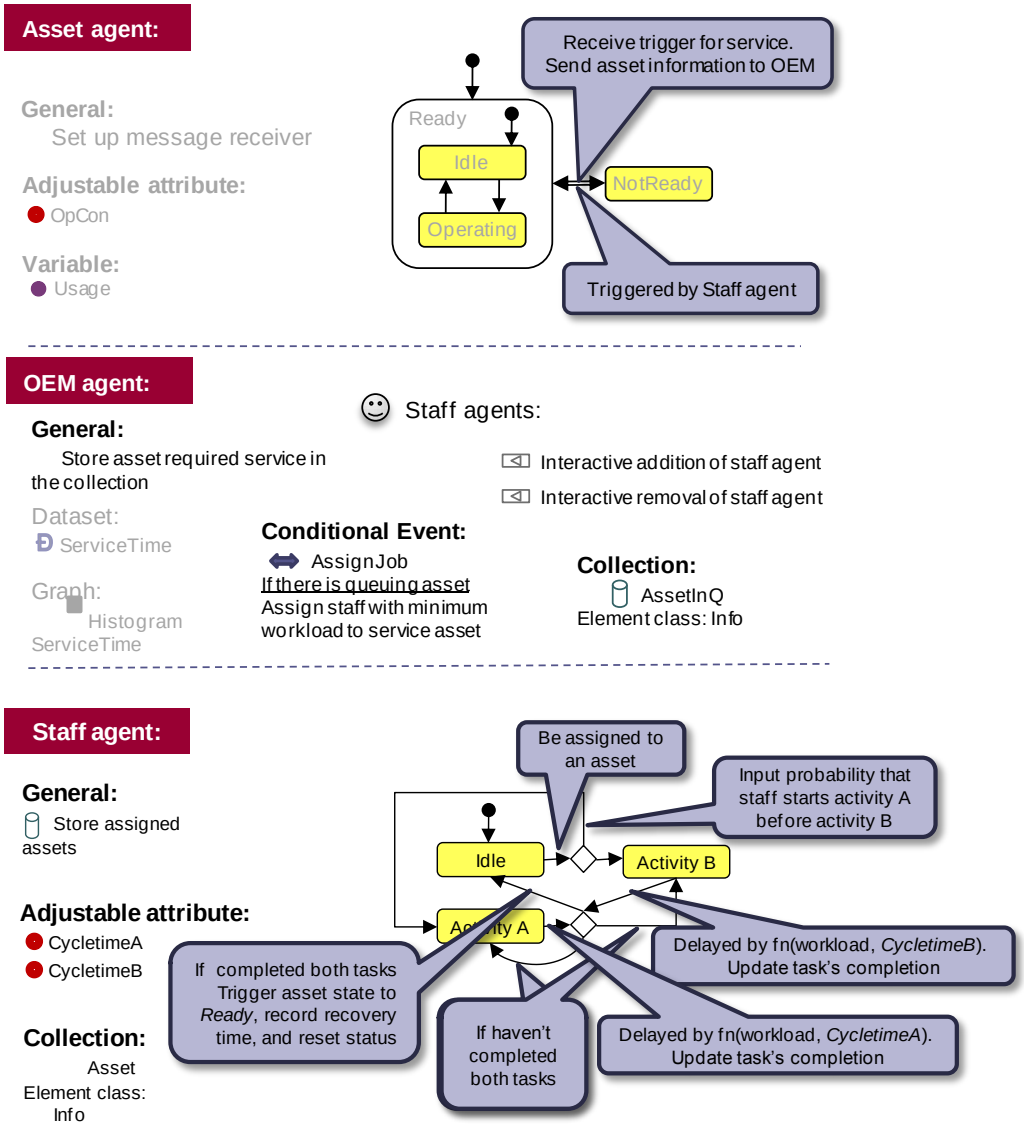


Figure 8-6: A2 construct

8.3.2 Subsystems

B0: The contracted product's state can be predicted on an aggregate level

For instance, it can be estimated that the assets will require service once every three months. In B0 construct (Figure 8-7), servicing schedule is monitored periodically via a cyclic timer and triggers asset state. The asset usage is updated after an operation.

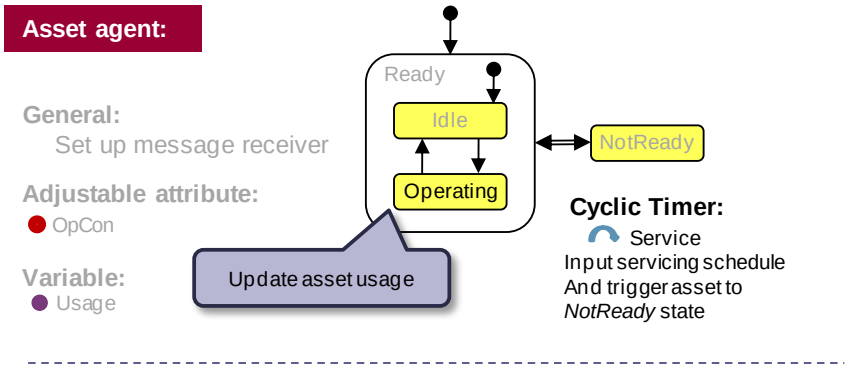


Figure 8-7: B0 construct

B1: The contracted unit requires breakdown analysis at a subsystem level

For example in an aircraft, the fuselage needs maintenance every 200 flying cycles, while engines require maintenance every 5000 flying cycle. Therefore, the OEM cannot estimate when the aircraft needs servicing on a fixed interval.

In B1 construct (Figure 8-8), the servicing schedule is monitored periodically via a cyclic timer and triggers the asset state. The asset's state depends on its subsystem behaviour. Therefore, these subsystems can be defined as the agents encompassed within the Asset agent. Each subsystem can encounter different degradation rates from other subsystems and adjust itself differently on various operating conditions. Once servicing is performed, whether degraded subsystems can be replaced or not, depends on the variable *ChangeLikelyhood*. This variable is driven by the remaining useful life which is a function of the past operating conditions and the estimated life.

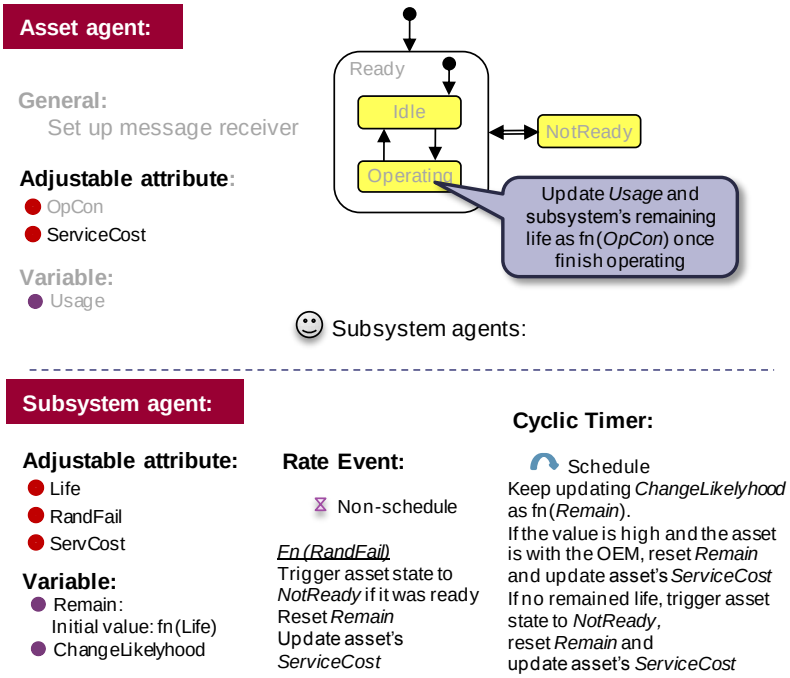


Figure 8-8: B1 construct

8.3.3 Work breakdown

C0: Service performance is measured only at the end of all operations

For instance, a contract guarantees up to 40-day turnaround time. No further customisation from the shared construct is required in this case.

C1 Jobs are preceded by several departments and service performances are measured separately (in case of A1 variant)

For instance, a contract guarantees that the OEM will respond to the call within 1 hour and recover the asset within 4 hours, and the OEM exhibits A1 decision making variant.

A Request agent is encompassed within the Asset agent in C1 construct (Figure 8-9). The information object is amended to be issued by the Request agent upon the change of the asset's state.

Another delay element is added to represent the additional department within the OEM agent. Each department completes each task in the Request agent. After all services are performed, the Request agent signals the Asset agent and destroys itself.

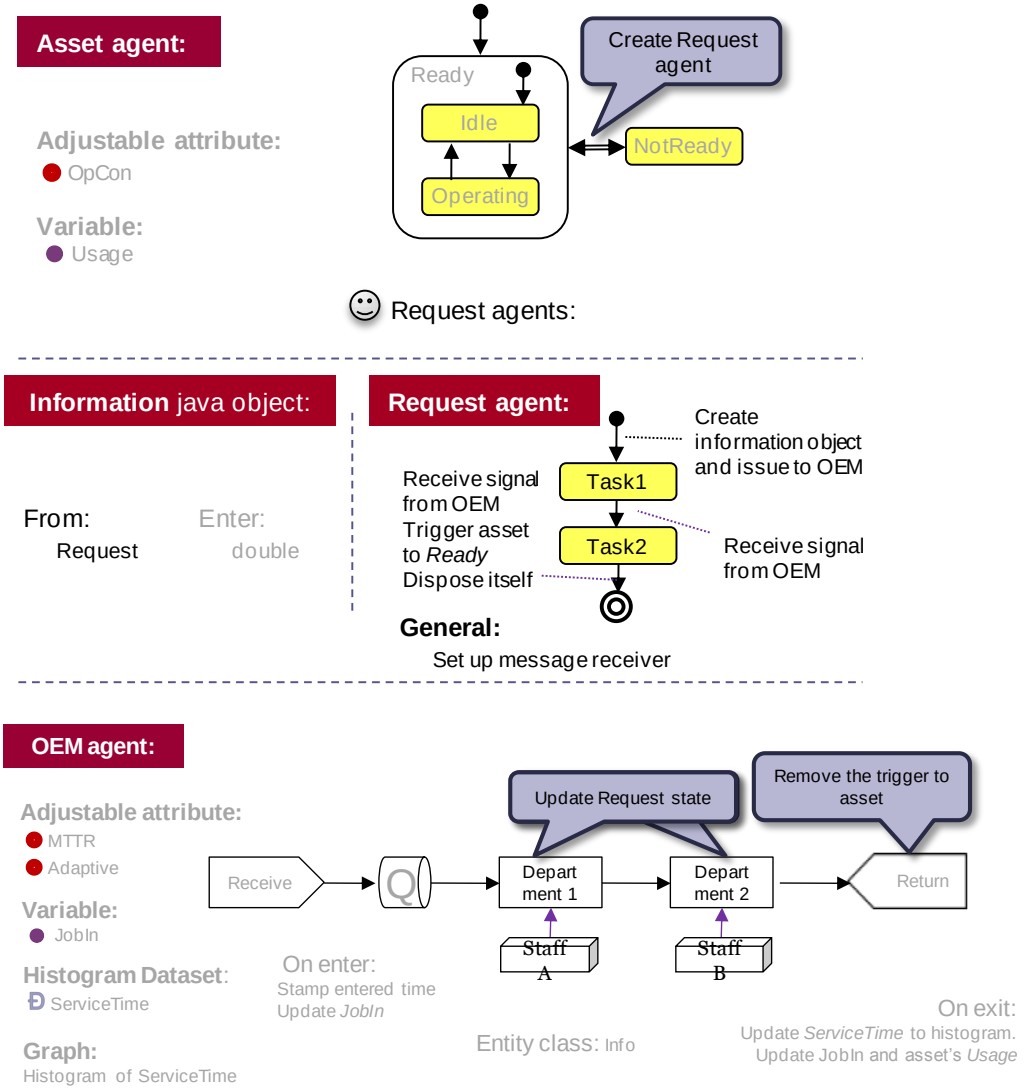


Figure 8-9: C1 construct

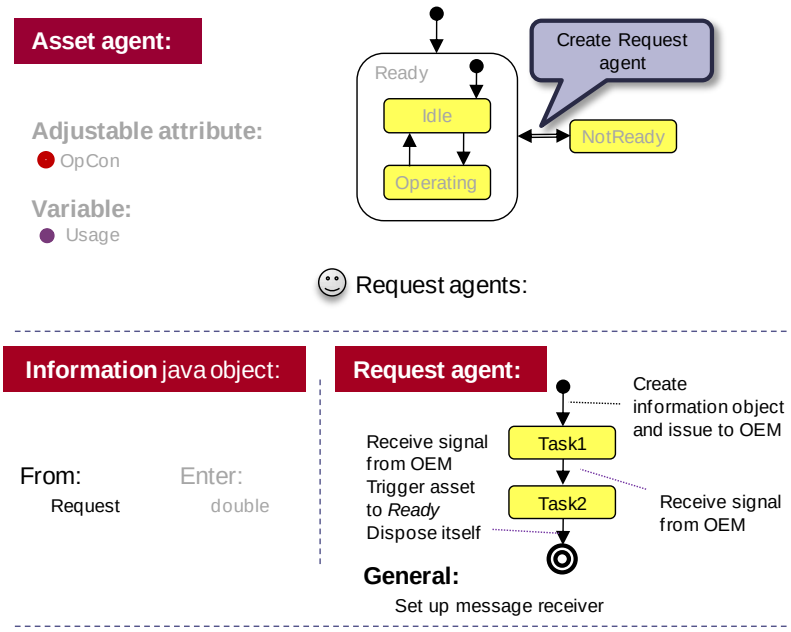
C2 Jobs are preceded by several departments and service performances are measured separately (in case of A2 variant)

For instance, the contract guarantees to respond to the call within 1 hour and to recover the asset within 4 hours, but the OEM exhibits A2 decision making variant.

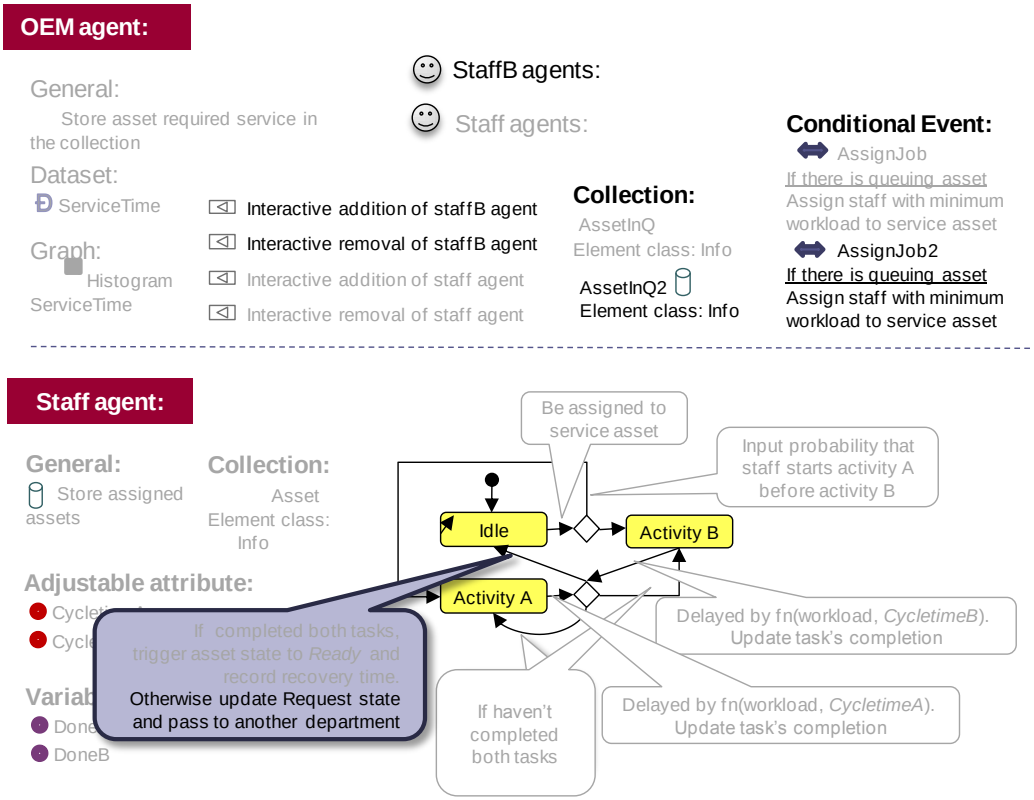
A Request agent is encapsulated within the Asset agent in C2 construct (Figure 8-10). The information object is amended to be issued by the Request agent upon the change of the asset's state. After the object is sent to the OEM agent, the agent registers the request in *AssetInQ,* and activates *AssignJob* to allocate the job to field staff based on

the staff's workload. The staff have flexibility to select a sequence of tasks to perform the service.

StaffB agents are created to represent the additional department within the OEM agent. Each department completes each task in the Request agent. After the service is performed by both departments, the Staff agent signals the Request agent to further update the Asset agent's state and to dispose off itself. If the service has not been completed by both departments, the Staff agent signals the Request agent to update its state.



(a) Asset, Request, and Information object constructs



(b) OEM and Staff constructs

Figure 8-10: C2 construct

8.3.4 Contractual mode

D1: Fleet contracting, daily available assets, daily penalty, and monthly payment

In this category, the service contract is made on the entire fleet of assets, available assets and charges are recorded daily, but the payment is made monthly.

D1 construct is shown in Figure 8-11. Within the Customer agent, the variables depict contract performance (*Availability*), reward (*Revenue*), and risk (*Penalty*). These variables result from the three attributes, specified in the contracts. These attributes are adjustable during model execution to represent contract renegotiation. The three cyclic timers control the daily monitoring of available assets (hence, penalty), the monthly contract payment, and the asset operating schedule. The transitions inside the Asset agent correspond to the operating schedule.

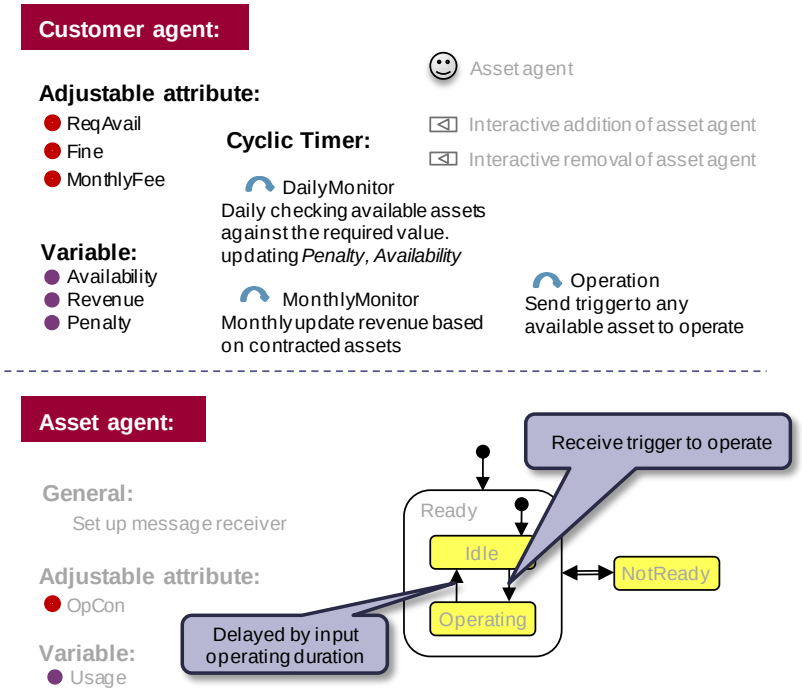


Figure 8-11: D1 construct

D2: Individual contracting on recovery time basis, charges for exceeded agreed duration, and monthly payment

In this case, the service contract is made on individual assets with monthly transactions, guarantees recovery period, and incurs charges for failure to recover the asset within the period.

D2 construct is illustrated in Figure 8-12. Within the Asset agent, the variables depict reward (*Revenue*), and risk (*Penalty*). These variables result from the three attributes specified in the contracts (guaranteed recovery period, contract price, charge). These attributes are adjustable during model execution to represent contract renegotiation. The two cyclic timers control monthly contract payment and asset operating schedule. The transitions inside the *Ready* state correspond to the operating schedule, whilst the transition inside the *NotReady* state updates the penalty if the guaranteed period is exceeded.

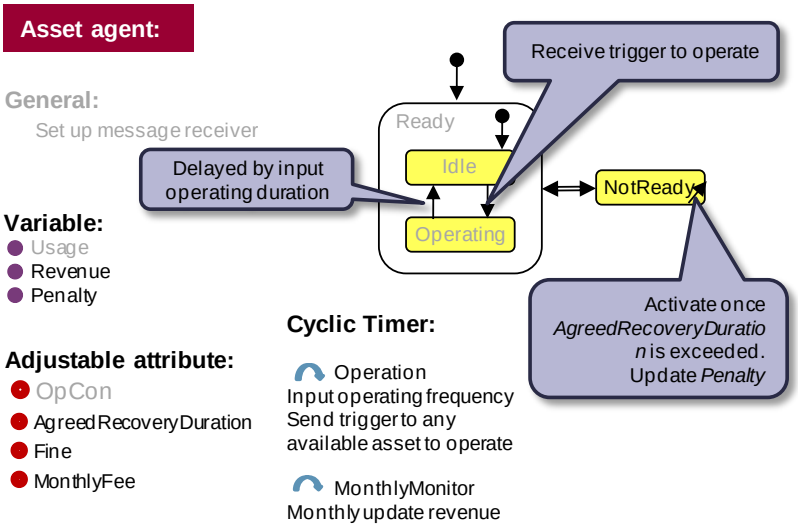


Figure 8-12: D2 construct

D3: Individual contracting percentage uptime or available period basis, charges for failure to achieve the level, and monthly payment

In this case, the service contract is made on individual assets with monthly transactions, guarantees available period or percentage uptime, and incurs charges for failure to achieve the level.

D3 construct is illustrated in Figure 8-13. Within the Asset agent, the variables depict contract performance (*UpTime*), reward (*Revenue*), and risk (*Penalty*). These variables result from the three attributes specified in the contracts (guaranteed uptime, contract price, charge). These attributes are adjustable during model execution to represent contract renegotiation. The two cyclic timers control monthly contract performance (including payment) and asset operating schedule. The transitions inside the *Ready* state correspond to the operating schedule and uptime monitoring.

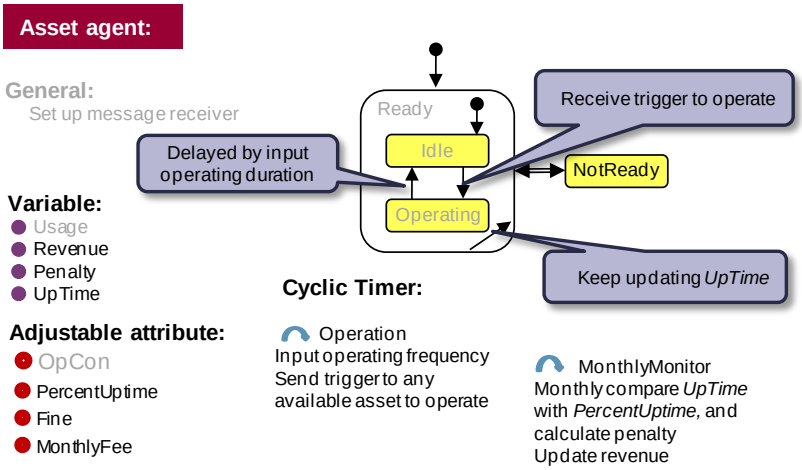


Figure 8-13: D3 construct

D4: Pay-per-use and charges for failure occurs during operation.

In this case, the customer pays only when the asset is operating and the OEM is charged if the asset fails during the mission.

D4 construct is illustrated in Figure 8-14. Within the Asset agent, the variables depict reward (*Revenue*) and risk (*Penalty*). These variables result from the two attributes specified in the contracts (price-per-use, charge). These attributes are adjustable during model execution to represent contract renegotiation. The cyclic timer controls the asset operating schedule. The OEM is charged in advance of the operation and receives it back after the operation to penalise only if the asset's failure occurs during an operation.

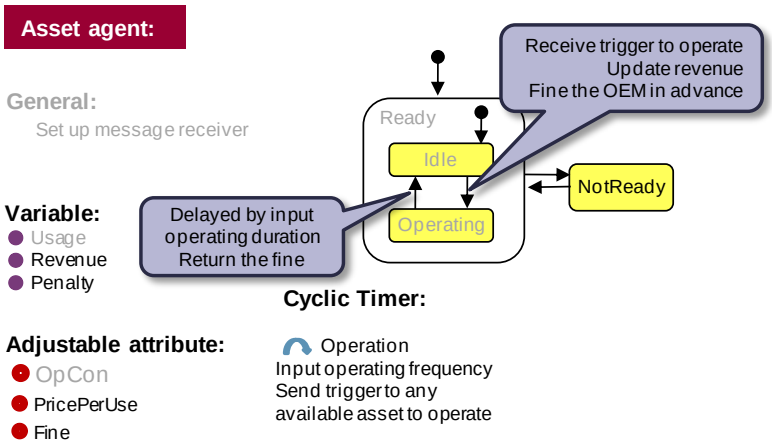


Figure 8-14: D4 construct

All constructs are independent from software tools. In other words, modellers can apply the constructs on any software packages that support ABS and DES techniques. The demonstration how to implement these constructs in a software package is provided in Appendix L.

8.4 Chapter summary

This chapter presents the final modelling constructs that can be used to provide effective and efficient simulation model developments. The constructs comprise two parts: the shared service contract construct and the case-dependent constructs. The shared construct incorporates common PSS offering model elements whereas the case-dependent constructs capture the case-dependent elements. The case-dependent elements are caused by the variants in service decision making, subsystems, work breakdown, and contractual mode. The next chapter discusses the achievements of this research.

9 Discussion

The previous chapter detailed the final modelling constructs, which completed the last objective and the aim of this research. This chapter discusses research findings in Section 9.1, strengths of this research in Section 9.2, limitations of this research in Section 9.3, and emergent literature in Section 9.4. This chapter is then summarised in Section 9.5.

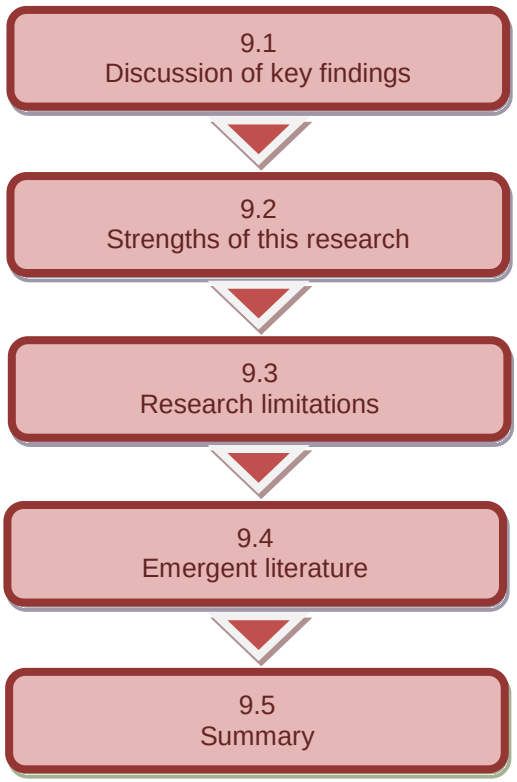


Figure 9-1: Chapter 9 outline

9.1 Discussion of research findings

The findings obtained during the research described in this thesis are discussed below.

9.1.1 Impacts from different contracts on simulation modelling approach

PSS has been an ongoing interest for the research community. Before the case studies were conducted with the companies, the ideas and developments of models in this thesis were led by reported cases from literature and various discussions with other researchers within the PSS community. There were two extremes of opinions from the community. At one end, PSS is perceived as having a large variety and a wide range of contracts. This range is seen so wide that at the beginning of this PhD research programme, the community doubted whether the aim of this research can be

achieved. At the other end, another group of researchers viewed PSS contracts as a standard format which has the agreed availability, reliability, and supportability levels. However the body of literature cannot provide sufficient evidence to justify these arguments.

Along this line, it is always believed that although there might be a wide range of contracts, some of them would share a common format. Therefore, some differences among these contracts may not affect the modelling approach. This means it is possible that two different contracts can be designed using the same model. This research has always attempted to support this hypothesis.

Having discussed with the practitioners involved during the evaluation process, all cases have their own formats which apply for all contracts in the company. The parameters are consistent between contracts, but the values of these parameters may change across contracts. Similarly, the scope of services can be renegotiated. This implies that, for each company, a model can be built and reused for all contracts. Among these cases, one company opens for new contracting ideas. In other words, there has been one format, but other formats can be implemented if it proves beneficial.

Based on the validation of the primary constructs, the hypothesis was supported from the fact that many model elements have been repeatedly adopted across cases. Particularly in the final constructs, the case-dependent variants are even further decreased from the primary constructs. Nonetheless, it has to be noted here that to a large extent the constructs were formulated from a product perspective. It may need additional modifications in order for them to be applicable to service-centric PSS context.

9.1.2 Simulation as a tool for PSS design

Having presented the models, researchers in the community were mostly impressed by what the models were capable of. Very positive feedback was often received in terms of the model's capability in visualising contract performance in real time and the interactive functionality that allows users to interact with the models. In other words, users were excited by seeing how the contract performances were affected after their decisions to change the particular inputs. However, there were some doubts about the underlying logic inside the models.

From the experience throughout the research, a simulation model of service contracts could take several months to develop and verify without the constructs, in contrast to a matter of days using the constructs. Therefore, developing the models from scratch depends greatly on the modeller's experience and his understanding of the problem, especially, when detailed modelling is required.

Besides, it was found that an important factor in ensuring a reliable model is the understanding of some underlying mechanisms inside the software package. For instance, two replications of the same experiment were conducted with identical inputs at different days with the same random number stream. It was noticed that the outputs of the first asset and the last asset in the array were interchanged. At that time it was understood to be an effect from multi-thread execution, as stated by Yu (2008) that a multi-thread feature is often incorporated in an agent paradigm. However, the software vendor clarified later that the package runs in single-thread mode. Therefore, the interchanged outputs were a result of the next-event time handling method when two events take place simultaneously. In that experiment, the two assets had identical inputs, thus, they triggered events at the same time. This means that an experiment may not produce an identical debugging sequence. It has to be noted that this time handling method is typical in DES and ABS and independent from software package. Consequently, it is important to check how the applied software engine handles this effect in order to avoid misbehaviour.

For these reasons, simulation may not be the most appropriate tool considering the effort and time spent to model complex systems if the modeller has no guideline, inexperience, or lack of understanding in the system.

Nonetheless, simulation has been proven effective in enhancing an understanding of the system and the topic. Having completed several PSS offering models, the contract success-failure mechanism was clearer, the interview questions were appropriately structured, and the unexpected scenarios during the interviews were better understood.

Three fundamental simulation modelling techniques have been analysed in this thesis: SD, DES, and ABS. Section 4.1 already introduced their capability and drawbacks from other contexts. In the context of product-centric PSS, they are summarised as shown in Table 9-1. In the Table, DES-ABS is considered from the final constructs.

Table 9-1: Summary of capability and drawbacks of simulation techniques within the PSS context

	SD	DES	SD-ABS	DES-ABS
Capability	Simple to form. Clearly shows influences and relationships between parameters. Can include more factors to be investigated than other methods. Easy to communicate the model.	Can capture customers-OEM relationship as a result from product performance. Can clearly present 'value' parameters and service efficiency.	Can clearly present 'value' parameter, service efficiency measure, and input's uncertainties. Can expose individual assets, their life cycles and their active nature. Enable assets to remain in the model as well as being disposed at the end of their life. Enable effects of dynamic behaviour such as contract renegotiation to be explored.	Can clearly present 'value' parameter, service efficiency measure, and input's uncertainties. Can expose individual assets, their life cycle and their active nature. Enable assets to remain in the model as well as being disposed at the end of its life. Enable effects of dynamic behaviour such as contract renegotiation to be explored. Can embed priority rule to assets waiting in the queues.
Drawbacks	Could not illustrate decentralised decision making and stochastic nature of in-service activities. Could capture 'value' on high level. Asset life cycle could not be visualised.	Not natural to present hierarchy of decision making. Could not show asset autonomy. Asset life cycle and usage information could not be exposed. Required a lot of analogy.	Could not incorporate queuing rule. Unnatural to present asset's life cycle in the OEM model.	May result in an overly-detailed model.

9.1.3 Agent-oriented approach in the PSS modelling context

This research finding is linked with the use of agent-oriented approach in a new context. An agent is capable of being identifiable, situated, goal-directed, autonomous, and flexible. These functionalities are encompassed in the final constructs as follows.

By being identifiable, agent behaviour was designed inside the OEM agent, Customer agents, and Asset agents. Thus, it enabled fast and easy replication of agents. Only their attributes/parameters (e.g. contract price) needed to be defined. Additionally, it allowed the entire system to be broken down to agent by agent, which encouraged the modellers to model and verify the whole system bit by bit.

By being situated, the influences and communication between agents can be established. An example of the influences is the impact from Subsystem agents' condition on the Asset agent's health. The flexibility functionality triggers several events in the constructs, for instance, Staff agents can adapt their productivities depending on their workloads. Finally, the goal-directed and autonomous functionalities prevent inappropriate interactive commands by users. For instance, once a user interactively removes a staff during model execution, a decision rule can be programmed to activate only if the staff is idle.

Nonetheless, as the entire system can be modelled by breaking it down to pieces, the modellers may lose the 'big picture', and hence, include too many details in one agent. As a result, an ABS model can be overly-complicated. Therefore during modelling process, it is recommended to initially assume that *"No agent is needed, and the model can be developed from SD or DES",* then find the argument to prove otherwise. This was applied to the modelling approach prior to the formation of the constructs, which proved to significantly reduce complexity. ABS should be used only when the traditional techniques: 1) are insufficient, 2) can be considered unnatural, and 3) can be created only through extensive coding. The hypothesis can be applied from the main model down to the lower layer.

The speed of model execution can also be easily influenced in an ABS model. For instance, it can be executed substantially slowly by modelling agent behaviour in the lowest hierarchy using continuous modelling elements such as flow variables. Besides, it can be significantly slower as agents in the model increase and it is run in single-thread mode. This is because an agent is executed in series and in a very small time step.

9.1.4 The constructs as an aid for decision making

In the existing PSS literature (e.g. Baines et al., 2007; Tukker, 2004; Mont, 2002), product element and service element are often considered as a combined package. However, both TrainCo and EngineCo separate these two elements. In TrainCo, the trains are normally sold to ROSCO, and the operators lease these trains from ROSCO

but sign service contracts from TrainCo. Moreover, in EngineCo, contracted engines may not be manufactured by EngineCo. This finding indicates that the school of thought between the literature and practice are not quite match.

Another sign of mismatch between PSS literature and practice comes from the criticality in delivering the agreed performance. The pre-defined penalty and the fact that the contracts commit the OEMs for long period of time led to the perception that it is extremely important to deliver the agreed performance in PSS. However, the results from interviews during SD model development and validation indicate that this issue is important but not critical, particularly with EngineCo and ShipCo. In the EngineCo case, the OEM could negotiate to pay no penalty with the customer. In case of ShipCo, the company could even negotiate for additional payments with the customer under some unexpected circumstances.

Whilst these findings revealed some misperceptions between PSS theory and practice, the constructs can incorporate both theoretical concept and closely capture practical situations, as can be seen from Table 9-2. In the table, the plus signs indicate strengths or included issues, while the minus signs depict weaknesses or uncovered issues.

Table 9-2: Benchmarking between the literature, the constructs, and current practice.

Criteria	Literature	Constructs	EngineCo	ShipCo	TrainCo
Generalise to all PSS	Out of scope				
Incorporate some analytical techniques					
Extend life cycle perspective from product selling	+	+	+	+	+
Address value parameter explicitly	+	+	+	+	+
Highlight interactions between parties in supply chain and customer.	Out of scope				
Include both economic and environmental measures					
Demonstrate the link between asset transformation and service support	+	+	-	+	+
Present service efficiency measures	-	+	+	+	+
Capture link between product performance and customer-manufacturer relationship	-	+	+	+	+
Illustrate redesigns from customer involvements	-	+	-	-	+

Criteria	Literature	Constructs	EngineCo	ShipCo	TrainCo
Demonstrate decentralise decision making	-	+	-	+	-
Represent cultural mind frame, social habits, and influence between customers	Out of scope				
Capture effect of technology on company's capability	-	-	-	-	-
Incorporate government influence	Out of scope				

This table shows the contribution of the constructs in four areas (No.8 – No.13) and the direction for future research in three areas (No.12 – No.14). It can also be seen that PSS modelling literature is lagging behind the industrial implementation, whilst the constructs can capture all issues that appeared in the case industry. Therefore, from a theoretical point of view, the constructs can be a potential solution that aids decision making.

This research produced two versions of the constructs; the primary constructs which cover more scenarios and the final constructs that are easier to implement.

The primary constructs include more variants and amounts of decision logic to encompass more scenarios. They were also intended to enable model verification at the end of each step, and therefore, each step could be executed by itself. This intention, however, resulted in more modifications once all the steps were integrated. Furthermore, the constructs are independent from any software package. Therefore, exact code was not provided in this version of the constructs. However, this attempt reduced the confidence in completing the model of a user.

For these reasons, the amounts of decision logic and variants were reduced and user interactive features were added in the final constructs. The *basic construct* was modified to become the *shared construct* so that the customisation from the shared construct can be mostly made by adding elements rather than modifying them. Moreover, the constructs were implemented as an illustration in a software package. Nevertheless, this research neither aims to adopt the best modelling methods, nor provides a tutorial on using the software package. Therefore both versions exclude general basic functions such as agent creation.

Accordingly, the final version is easier to use, nonetheless, covers less functionality in analysing problems than the primary version. However, both versions can be effectively used to understand contracting issues and the correlation between contracting, operational planning, and financial issues.

9.2 Strengths of the research

This research can be evaluated from several viewpoints; research process, the contribution to knowledge, and the practical functionality.

9.2.1 The research process

This research **has systematically developed** and **contains explicit evidence** to support a conclusion for each objective of this thesis. In Chapter 2, a literature review was conducted to examine successful cases and existing modelling techniques, which led to the identification of gaps of knowledge. The review **was evaluated and refined by a wider community** in a journal publication. In Chapter 3, the research aim and objectives were developed based on the gap analysis. This led to the logical construction of a research methodology. Chapter 4 provided a detailed evaluation of the potential modelling techniques from literature and actual model developments. The analysis from this stage was primarily made against the gaps in knowledge identified in Chapter 2 which built up the arguments as to why a hybrid ABS-DES was considered the most appropriate technique to underline the core structure of the modelling constructs. The evidence in this stage, which includes a literature review of the relevant modelling techniques, the resulting models, and their analysis, were presented as conference and journal papers and to other simulation experts and practitioners. In Chapter 5, the hybrid technique was applied to different case studies to refine and generalise the modelling approach. The analysis and lessons learnt from this stage led to the modelling approach and underlying modelling methods inside the constructs. The detailed models were verified with a simulation expert and are documented in the Appendices (E, F, G, H). The constructs presented in Chapter 6 were validated using case studies and external users in Chapter 7. Based on the validation result, the final constructs were developed. Throughout this research, the research methods for carrying out simulation study have always been followed to ensure model functionality.

Besides the reviews by the wider community, this research attempted to ensure quality of results and empirical study by **selecting relevant case studies**, and by **following case study protocol**. The cases were chosen from different sectors under a product-centric PSS context so that generality of results could be enhanced. The interviewees are all directly involved in offering service contracts. Seven different cases have been covered in this thesis, including four cases in Chapter 5. The case study methodology was selected to enable insight into the contracting decisions and the modelling approach. The case study protocol ensured completeness and relevance of the data. **All interviews were conducted jointly with another researcher** to reassure consistency of information.

To evaluate reliability and repeatability, the constructs **were validated externally by other modellers** with different simulation experiences. The author closely observed the development process to collect implementation data. Additionally, the modellers

provided feedback after developing models from the constructs. This process also aimed to remove bias while evaluating the usability of the constructs. The usefulness of the constructs was also **evaluated by practitioners** who were involved in the validation. Moreover, the concepts and the models applied in this research have been **continuously discussed with other researchers in the PSS community** to verify the underlying assumptions and develop wider understanding of the topic.

Lastly, throughout this research, the **literature survey and assessment of the developed models against the gaps in knowledge have been conducted iteratively** after the end of each step. The intention was to continuously evaluate the achievement and novelty of the constructs.

9.2.2 Contribution to knowledge

A number of contributions to knowledge have been produced in this research, summarised in Table 9-3. The main contribution is *a new way of rapid development of simulation models of service contracts (using modelling constructs), which can assist modellers to build and analyse service contracting implications in an effective and efficient manner*. In addition to that, this thesis also provides secondary contributions which relate to the use of simulation techniques in a new context, new ways of capturing the extended characteristic and dynamic behaviour of PSS beyond the traditional business, and rapid developing service contract simulation models. This research also brings together theoretical PSS research, operational planning and decision support tools.

Table 9-3: List of contributions

Contributions	Related gap in knowledge (Section 2.5)	Potential benefits	Related chapter
Identification of gaps in knowledge in PSS modelling	n/a	Address future researches in the potential areas. Provide lists of model parameters for further model customisation.	Chapter 2
Use of simulation in a new context	"There was no guideline on the usability and suitability of simulation techniques for a particular PSS problem."	Use to map an appropriate technique to a particular PSS problem.	Chapter 4
A new approach that brings together three research areas: theoretical PSS, operational planning, and decision support tools	"The majority of techniques in the literature produced results in the forms of guidelines, configurations, or specifications, which are appropriate for the high level design stage. Once designed, relationships in the system were mostly fixed." "…an evaluation could only be made separately between the hierarchies, thus the effects of unexpected behaviour from across levels may be discarded in the models."	Guide how to effectively use simulation for decision making. Provide a potential step towards a powerful decision support tool. Lower barrier in PSS adoption.	Chapters 4, 5, 7
A new way of rapid developing service contract simulation models	"The applicability of existing models in the literature was narrow and cannot be reused across cases."	Shorten model development time.	Chapters 6 and 8

Contributions	Related gap in knowledge (Section 2.5)	Potential benefits	Related chapter
A new way of capturing the characteristics beyond traditional business.	"Indicators of service efficiency, such as asset availability, asset operating times, and functional reliability, were not commonly found." "The changes of states of assets during the in-service phase, asset's relationships with manufacturer and customer, and the associated risks and penalty, are still limitedly exposed." "Decentralised decision making was not properly taken into account in any model." "Service response time was not examined as an output." "The relationships triggered by the product availability and performance, were not emphasised." "The degree of interactions between suppliers and manufacturers, market responsiveness and the influences between customers were also not explicitly included in any model."	Enable effective provision of a contract and avoid losses. Visualise risk and reward mechanism during the contract delivery phase.	Chapters 4, 5, 6, 8
A new way of capturing dynamic behaviour in the contract delivery phase	"The majority of the existing tools do not incorporate the time dependent variables." "The capability in encapsulating customer's autonomy in asset operations and in assessing risks from external events beyond the boundary of each agent is still limited."		

9.2.3 Practical implication

The constructs bring two levels of practical benefits. On a high level, the effectiveness of the constructs enables the OEMs to understand the risk-reward mechanism and implications of service contracting prior to making a contract. On the lower level, the effectiveness of the constructs allows the modellers to appropriately model service contracts, and the efficiency of the constructs assists the modellers to perform the task in a timely manner. Without the constructs, the modellers need to develop a model from scratch. The modelling elements can also be reused once created, which means a model can be created more quickly as the modellers get more familiar with the constructs.

In summary, the models showed the following benefits:

- A contract can be customised based on the asset usage and the price of the contract can be estimated rapidly. This means an OEM can be flexible to customer needs.

- A guaranteed contract performance can be estimated based on the OEM's current operational capability and the asset capability. This also enhances contract customisation.

- The profits from alternative contracts can be compared prior to making a contract. This capability is beneficial for an OEM to negotiate a contract with a customer.

- Impacts of any possible risks or any potential dynamic behaviour during the implementation phase (e.g. change of the asset usage, price changes) can be visualised before making a contract. This capability allows an OEM to take into account the impacts, and therefore, avoid losses from a contract.

- Alternative strategies can be compared so that an effective capability improvement strategy can be identified, for example, an asset life extension and staff recruitments. Consequently, unnecessary investments can be avoided.

- Performances of both individual level and the entire system can be monitored throughout the contract period simultaneously. This allows the effects between hierarchies to be captured.

- The profits and losses from a particular ratio of traditional business and contract business can be evaluated. Therefore, inappropriate marketing strategies can be prevented.

These capabilities were detailed in Chapter 7.

9.3 Limitations

This section identifies some limitations of this research, which can be associated with the research process and the constructs.

In terms of the research process, the first issue was related to **the number of case studies**. Even though several models have been built in this research, the constructs were primarily developed from four case studies and refined using another three case studies. The limited number of case studies was constrained by time which was devoted to handle other important aspects. Firstly, it was important to apply a simulation technique that could cope with the shifts in modelling. However~~t~~, literature that mapped different simulation techniques with PSS environment was lacking. Therefore, additional empirical studies were required before the final technique was selected (Chapter 4). Secondly, as ABS is generally new, a lot of manual code was unavoidable. This characteristic, in combination with the fact that PSS is complex, required a significant amount of time in developing, verifying, and validating the models without any guideline (Chapter 5). Last, each validation required a series of interactions and commitment between the author and the company/user. As a result, the time to include more cases was limited. However, the author participated in PSS events and discussed the scenarios with other researchers in the PSS community to gain wider understanding of the topic.

The second issue is related to **bias**. The initial source of bias could arise from the fact that the author is familiar with the research area and the constructs. As a result, model modification could be performed in a timely manner with less extensive effort. Nonetheless, this bias was minimised by involving external participants to evaluate the usability of the constructs. The second bias could be caused by the participant's experience in developing ABS models. The participants might compare the ease of use of the constructs with their familiar software packages developed from other commonly-used simulation techniques (such as DES). As those techniques have been matured, their software packages tend to be well-developed and user-friendly. Consequently, participants may expect the constructs to be as easy as those software packages. Moreover, their experience and understanding of the topic potentially influenced the modifications. To enable a fair comparison, implementation of the constructs as a commercial-off-the-shelf tool would be necessary. However, in this research, this bias was handled by involving an expert in ABS and PSS areas in the validation. The last bias could be a result from the limited number of participants involved in the user validation. Nonetheless, the author aimed to closely observe their difficulties during the modelling process, and to ensure that their feedback was insightful and valid. This required the participants to actually follow the constructs and develop the model, which cannot be ensured by using surveys. This process is too time-consuming to involve many users.

There are two limitations regarding the constructs. Firstly, they still require **modification** to suit each case to some extent. This is because different companies may be interested in different model parameters to experiment with. In fact, the constructs could have been designed to cover as many analysis elements as possible. Nevertheless, it means that there would be too many unnecessary elements for some companies. Therefore, the constructs include only commonly-used elements and further modifications are left to the users. Secondly, the constructs aim to provide a guideline to users, not the best modelling methods. Thus, they need some levels of modelling **experience** to apply programming code. Additionally, as PSS simulation modelling literature is still in an early phase, this research focussed on building modelling capability from the fundamental stage. Therefore, the constructs were developed for learning rather than providing solution. For this reason, output validation was excluded in this research.

9.4 Emergent literature

PSS offering design was an ongoing interest in literature during the undertaking of this research. In high level design, attempts have been made to conceptualise PSS using reference model and modelling language (Becker, 2010), and UML (Lin, 2010). Similarly, ServiceCAD (described in the literature review chapter) has been continuously developed. At a detailed level, Monte Carlo was adopted to simulate life cycle cost of machine tools (Lanza et al., 2011). DES was used to enable comparison across different product-service business models (Kuo, 2011; Ball et al., 2010). Also, a hybrid SD-ABS was proposed to estimate costs whilst including risks and uncertainties in a PSS supply chain (Erkoyuncu, 2011).

Nevertheless, these publications do not highlight active nature of in-service assets and their life cycle as well as the possibility of renegotiation and early termination of contracts.

9.5 Chapter summary

This chapter discussed key findings, highlighted strengths of this research in terms of research process, contributions to knowledge, and contributions to practice. Limitations were also addressed. The chapter ended the discussion by identifying related literature that emerged during the period of research. This reveals an ongoing interest in the area, yet, it does not impact on the contributions of this research. The next chapter concludes this thesis.

10 Conclusions

The previous chapter already discussed the research findings obtained during the undertaking of this research. This chapter concludes the outcomes from this research in correspondence with the research objectives (Section 10.1) and provides direction for future research (Section 10.2). Finally, concluding remarks are stated in Section 10.3.

10.1 Summary of achievements against objectives

Ultimately, this research aims to propose the modelling constructs that enhances effective and efficient development of service contract simulation models. The constructs can enhance efficient development of simulation models as they can shorten modelling development time significantly. The constructs also enable effective development due to four reasons. Firstly, the characteristics and dynamic behaviour in PSS can be captured, which was lacking in the body of knowledge. Secondly, some degrees of case customisation are incorporated which enhances applicability of the constructs. Thirdly, the models developed from the constructs have been proved by practitioners and experts to be practical and meaningful. Finally, it is feasible to develop valid models based on the constructs.

The constructs are led by the achievements of each research objective as follows:

1. To identify a simulation technique that can potentially capture PSS characteristics and dynamic behaviour

A systematic process was conducted to ensure that this objective is achieved and the conclusion is valid.

The first stage reviewed existing modelling techniques in PSS. The review addresses PSS characteristics (Section 2.3.1), dynamic behaviour (Section 2.3.2), and existing modelling techniques in PSS (Section 2.4.1). The techniques were then evaluated in terms of capability in capturing the effects from dynamic behaviour in PSS (Section 2.5.2). As a consequence, three simulation techniques were primarily identified as potential techniques: SD, DES and ABS.

The second evaluation was presented in Chapter 4. In this chapter, the capabilities and drawbacks of these three techniques, obtained from the simulation literature, were first described (Section 4.1). To enhance the insight of analysis, actual model developments were conducted using these three techniques and their combinations (Section 4.2). The model functionalities were contrasted with 1) the strengths and weaknesses of PSS modelling literature analysed in Section 2.5, and 2) the shifts in modelling principle addressed during the model development, summarised in Section 4.2.5.

The outcome of this stage justifies the hybrid DES-ABS technique as an appropriate technique as it can potentially fill the gap of knowledge from both PSS and modelling viewpoints. From the PSS perspective, the hybrid technique could expose value parameters, service efficiency measures, decentralised decision making, extended life cycle from manufacturing, and OEM-customer relationship as influenced by product performance. In terms of modelling, the technique effectively represented input uncertainties, asset states, asset's heterogeneity, and their independency from the OEM. Besides, the hybrid model was proven to enable user interactive adjustment of several inputs during model execution. This functionality can illustrate contract renegotiation, operational changes, and market sensitivity.

The validity of achievement from this stage is enhanced due to two reasons:

- The primary investigation explored a broad scope and covered wide results, whereas the secondary investigation provided insights. Therefore, the conclusion is valid from both horizontal and vertical perspectives.

- The analysis was obtained from literature and practical viewpoints. Thus, the arguments are both conceptually and practically valid.

2. To determine an appropriate modelling approach that enables effective and efficient development of the models

To achieve this objective, the author applied the hybrid technique to various case studies in Section 5.1. These cases were led by the available information from literature, and selected from different sectors. The intention was to generalise the knowledge in modelling from different contracting scenarios. As the data obtained from the literature were not complete, the author validated the underlying assumption with PSS experts in most cases. However, it has to be pointed out that the focus of this stage is on the knowledge captured during the modelling. Accordingly, the external validations were mainly carried out by presentation to an expert in simulation within PSS context.

The results from model development led to the following major conclusions:

- It is too complex to design a construct which covers all theoretical PSS characteristics and dynamic behaviour.

- To optimise both feasibility and applicability of the constructs, common PSS elements and case-dependent elements should be managed separately.

- The common PSS elements relate to asset life cycle, OEM service process, in-service asset information, service efficiency measures, and usage-based analysis. The case-dependent elements involve contract modifications and

termination, contract creation, decentralised service decision making, asset structure, the track of job progress and an adaptive capacity.

- The case-dependent PSS elements resulted in various high-level model structures. However, the fundamental model elements in all cases comprise a Java information object, an OEM agent, Asset agents, and PSS environment.

3. To form primary modelling constructs based on the approach

This objective was achieved in Chapter 6. The findings in Chapter 5 were implemented in a simulation software package. The constructs consist of two parts: the basic service contract construct and the case-dependent constructs which enable the customisation of the basic model to suit each business case. The basic construct represents the common PSS elements and the case-dependent constructs capture their variances.

The basic construct consists of two layers and contains four fundamental model elements. The first layer contains an **OEM agent**, **Asset agents** and a **PSS environment**, whereas the second layer details the OEM service process inside the OEM agent and asset states inside the Asset agents. The state change inside the agents initiates the last element: a **Java object** which contains asset information and is passed between OEM and Asset agents.

The constructs were evaluated in terms of the ability to represent a PSS environment and the outcome reveals the effectiveness to model service contracts.

4. To evaluate and refine the primary constructs

To achieve this stage, the constructs were validated using three case studies under a product-centric PSS context and tested by three users. This step was presented in Chapter 7. To enhance generality, the cases were selected from different industries and the users were chosen from different simulation backgrounds. The outcome from both validation approaches proved applicability, practicality, feasibility, and capability in shortening model development time to a great extent. Additionally, the analysis highlighted opportunities to improve the constructs by making implementation explicit, simplifying the customisation, as well as reducing some variants.

5. To present the final constructs

As a result from the validation, the constructs were amended to have the following features:

- To enhance the use of the constructs for modellers, the overview and the user manual were first presented, and followed by the constructs. An example of implementation on a software package is provided in Appendix L.

- Similar to the primary constructs, the final constructs consist of two parts: the shared construct and the case-dependent constructs. However, the case-dependent constructs reduce the number of code modification made to the shared construct and are primary based on element additions for the ease of case customisation.

- The shared construct consists of three layers and contains five fundamental model elements. The first layer contains **OEM agent**, **Customer agents** and a **PSS environment**. The second layer details the OEM service process inside the OEM agent and **Asset agents** inside Customer agents. Finally, the third layer models asset's state within the in-service phase. The state change inside Asset agent initiates the last element: a **Java object** which contains asset information and is passed between OEM and Asset agents.

- The case-dependent constructs relate to variants within service decision making, subsystem, work breakdown, and contractual mode.

- The constructs are intended to guide model development independently from a software package, therefore, exact programming code is excluded. This enables experienced modellers to apply their familiar modelling methods. However, the example implementation in a software package should bring confidence to users in using the constructs and ensure repeatability of results.

In conclusion, this research completed all objectives and the aim. The next section identifies possible future directions of work based on this research.

10.2 Future research

Firstly, other techniques may be used along with the constructs to improve the understanding and clarify parameters prior to model development. For example, IDEF0 (Integration Definition for Function Modelling) can be applied to visualise all inputs and outputs of the contracting decision process, a balance scorecard can be adopted to systematically clarify and classify the contract's performance indicators, and a checklist can be used to map the case with the variants.

Secondly, feedback from the user validation and the case companies reveal opportunities in implementing visual interactive capability. To accomplish this, an input screen may be separated from the model via tools such as Microsoft Excel. A conceptual modelling tool such as BPMN can be linked after the input screen to illustrate relationships between actors in one page. Also, the shared construct can be made as a template provided as an option prior to model creation, and the elements

which indicate the characteristic variances can be encapsulated in a module that can be quickly created by drag and drop operations.

Thirdly, there are some weaknesses identified in the PSS modelling literature that were not covered in the constructs to avoid over-complicating them. These include the impact of monitoring technology and influences between parties in the supply chain. Similarly, as the constructs focus on the OEM's standpoint, the measures in the constructs are in terms of economic (cost, revenue) and operation (recovery time, queuing assets) whereas environmental measure (waste amount, material consumption) and financial measures (net present value) were not included. These aspects can be further investigated and possible to incorporate in the constructs. In fact, this research explored effects of technology and influences from stakeholders using SD. However, they were not presented in the constructs as SD elements can slow down model execution considerably and these aspects can be more complicated using other techniques.

Fourthly, this research focused on modelling capability. Further improvements can also be made in applying some analytical techniques such as cost analysis to maximise the precision of outputs, and optimisation techniques to automatically provide a solution to the OEMs.

Finally, the scope of this research applies to product-centric PSS. This scope can be expanded so that decision support tools can be made available for other types of PSS. These include the impact of cultural mind frames and social habits.

10.3 Concluding remarks

This chapter aimed to demonstrate that the research contained in this thesis has accomplished all the defined objectives in Chapter 3. It summarised the achievements related to the research objectives, and provided directions for future research. Overall, this thesis has illustrated that the hybrid Agent-Based Simulation and Discrete-Event Simulation is capable of capturing key PSS characteristics and dynamic behaviour that span beyond the traditional product selling businesses. With the modelling constructs delivered in this thesis, it is now possible to build a model by directly ***mapping*** case characteristics with the construct variants, ***dragging*** and ***dropping*** elements, and the associated code will be generated on-the-fly. This suggests a shifted modelling paradigm/mindset from ***building*** a model to ***assembling*** a model. Lastly, it is believed that the developed methodology can contribute in evaluating various PSS offerings prior to making a contract, in the real world. Also, the constructs can realistically be further implemented as a computer-based tool that eases the modelling tasks for OEMs.

References

Abramovici, M., Neubach, M., Schulze, M. and Spura, C., 2009. Metadata reference model for IPS2 lifecycle management. *In: Proceedings of the CIRP IPS2 conference 2009*, 1st-2nd April, Cranfield, UK.

Aircraft commerce, 2006. CFM56-3 maintenance analysis & budget. *In: Aircraft Commerce issue No.45 April/May.*

AIA, 2011. *Performance-Based Logistics Award* [online]. Available from: http://www.aia-dev.org/newsroom/awards/pbl_award/ [Accessed 24th June 2011].

Almeida, F.L., Miguel, P.A.C. and Silva, M.T.D., 2008. A literature review of servitisation: a preliminary analysis. *In: Proceedings of the POMS 19th annual conference,* 9th–12th May, California, USA.

Alonso-Rasgado, T., Thompson, G. and Elfström, B-O., 2004. The design of functional (total care) products. *Journal of Engineering Design*, 15(6): 515–540.

Anderson, M. and Tukker, A., 2005. Perspectives on radical changes to sustainable consumption and production. *In: Proceedings of SCORE conference*, Copenhagen.

Aurich, J.C, Fuchs, C. and Wagenknecht, C., 2006. Life cycle oriented design of technical product-service systems. *Journal of Cleaner Production*, 14(17): 1480–1494.

Azarenko, A., Roy, R., Shehab, E. and Tiwari, A., 2009. Technical product-service systems: some implications for the machine tool industry. *International Journal of Manufacturing Technology and Management*, 20(5): 700–722.

BAE Systems, 2006. *BAE Systems Awarded £947M Tornado Support Contract* [online]. Available from: http://www.baesystems.com/Newsroom/NewsReleases/2006/autoGen_10711123731.html [Accessed 26th June 2011].

BAE Systems, 2010. *Case Study: ATTAC (Availability Transformation: Tornado Aircraft Contract* [Online]. Available from: http://www.sanders.com/WorldwideLocations/UnitedKingdom/UKDefenceIndustrialStrategy/CaseStudies/autoGen_10721411422.html [Accessed 8th November 2010].

Baines, T.S., Lightfoot, H.W., Evans, S., Neely, A., Greenough, R., Peppard, J., Roy, R., Shehab, E., Braganza, A., Tiwari, A., Alcock, J.R., Angus, J.P., Bastl, M., Cousens, A.,

Irving, P., Johnson, M., Kingston, J., Lockett, H., Martinez, V., Michele, P., Tranfield, D., Walton, I.M. and Wilson, H., 2007. State-of-the-art in the product-service systems. *Journal of Engineering Manufacture*, 221(10): 1543–1552.

Baines, T.S., Lightfoot, H.W., Peppard, J., Shehab, E., Johnson, M., Tiwari, A. and Swink, M., 2009a. Towards an operations strategy for product-centric servitisation. *International Journal of Operations and Production Management*, 29(5): 494–519.

Baines, T.S., Lightfoot, H.W., Benedettini, O. and Kay, J.M., 2009b. The servitisation of manufacturing: a review of literature and reflection on future challenges. *Journal of Manufacturing Technology Management*, 20(5): 547–567.

Baines, T.S., Lightfoot, H.W. and Kay, J.M., 2009c. Servitised manufacture: practical challenges of delivering integrated products and services. *Proceedings of the IMechE Part B: Journal of Engineering Manufacture*, 223(9): 1207–1215.

Ball, P.D., Tiwari, A., Alabdulkarim, A., Cuthbert, R. and Thorne, A., 2010. Using discrete event simulation to investigate engineering product service strategies. *In: Proceedings of the 8th International Conference on Manufacturing Research ICMR*, 14th-16th September, Durham, UK.

Banks, J., Nelson, B.L. and Nicol, D., 2009. *Discrete-event system simulation*. Upper Saddle River, NJ: Pearson Prentice Hall.

Banks, J., 2007. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice* [online]. Wiley online library. Available from: http://onlinelibrary.wiley.com/doi/10.1002/9780470172445.fmatter/summary [Accessed 20th July 2011].

Bazargan, M. and McGrath, R.N., 2003. Discrete event simulation to improve aircraft availability and maintainability. *In: Reliability Maintainability Symposium*, Daytona Beach, FL, USA.

Becker, J., Beverungen, D.F. and Knackstedt, R., 2010. The challenge of conceptual modelling for product-service systems: status-quo and perspectives for reference models and modelling languages. *Information Systems e-Business Management*, 8(1): 33-66.

Bennett, B.S., 1995. *Simulation fundamentals.* Hertfordshire, UK: Prentice Hall International (UK) Ltd.

Berkowitz, D., Gupta, J.N., Simpson, J.T. and McWilliams, J.B., 2005. Defining and implementing Performance-Based Logistics in government. *Defense Acquisition Review Journal*, 11(3): 255-267.

Bey, N. and McAloone, T., 2006. From LCA to PSS –Making leaps towards sustainability by applying product/service-system thinking in product development. *In: Proceedings of the 13th CIRP International Conference on Life Cycle Engineering,* 571-576.

Bianchi, N.P., Evans, S., Revetria, R. and Tonelli, F., 2009. Influencing factors of successful transitions towards product-service systems: a simulation approach. *International Journal of Mathematics and Computers in Simulation*, 3(1): 30–43.

Bonabeau, E., 2002. Agent-based modeling: Methods and techniques for simulating human systems. *The National Academy of Sciences*, 99(7): 280–7287.

Borshchev, A. and Filippov, A., 2004. *From System Dynamics and Discrete Event to Practical Agent Based Modelling: Reasons, Techniques, Tools* [Online]. XJ Technologies and St. Petersburg Technical University, Russia. Available from: http://www.systemdynamics.org/conferences/2004/SDS-2004/PAPERS/ 381BORSH.pdf [Accessed 8th November 2009].

Brailsford, S.C., 2008. System dynamics: what's in it for healthcare simulation modelers. *In: Proceedings of the 2008 winter simulation conference*, 7th–10th December, Miami, Florida, 1478–1483.

Buxton, D., Farr, R. and McCarthy, B., 2006. The aero-engine value chain under future business environments: using agent-based simulation to understand dynamic behaviour. *In: Proceedings of the MITIP2006*, 11th–12th September, Budapest, Hungary.

Cantos, J.N., 2009. *An investigation of the service delivery systems at Man Truck and Bus UK Ltd.* Thesis (MSc). Cranfield University.

Cavrak I., Stranjak, A. and Zagar, M., 2009. SDLMAS: A Scenario Mode ling Framework for Multi-Agent Systems. *Journal of Universal Computer Science*, 15(4): 898-925.

Chahal, K. and Eldabi, T., 2008. System dynamics and discrete event simulation: a meta-comparison. *In:* C. Currie, K. Kotiadis, S. Robinson and S. Taylor, eds. *Proceedings of the Operational Research Society Simulation Workshop 2008 (SW08*). Birmingham, UK: Operational Research Society.

Chandraprakaikul, W., 2008. *Strategic positioning within global supply chains*. Thesis (EngD). Cranfield University.

Chung, C. A., 2004. *Simulation modelling handbook: a practical approach*. CRC press.

Creswell, J.W. and Plano Clark, V.L., 2007. *Designing and conducting mixed methods research*. CA, USA: Sage.

Creswell, J.W., 1998. *Qualitative inquiry and research design: Choosing among five traditions*. Thousand Oaks, CA: Sage.

Datta, P.P. and Roy, R., 2009. Cost modelling techniques for availability type service support contracts: a literature review and empirical study. *In: Proceedings of the 1st CIRP Industrial Product-Service Systems (IPS2) Conference*, 1st -2nd April, Cranfield, UK.

De Coster, R., 2008. Differences in forecasting approaches between product firms and product-service systems (PSS). *In: Proceedings of the 6th international conference on manufacturing research (ICMR08)*, 9th-10th September, Brunel, 539–547.

EMSA, 2006. *Enclosure 2:Part A- Vessel Service Contract* [online]. Available from: http://www.emsa.europa.eu/Docs/opr/emsa_op1_2006_enclosure2a_vessel_availability_contract.pdf [Accessed 27th June 2011].

Erkoyuncu, J.A., 2011. *Cost uncertainty management and modelling for industrial product-service systems*. Thesis (PhD). Cranfield University.

Evans, S., Partidario, P.J. and Lambert, J., 2007. Industrialization as a key element of sustainable product-service solutions. *International Journal of Production Research,* 45(18): 4225-46.

Fujimoto, J., Umeda, Y., Tamura, T., Tomiyama, T. and Kimura, F., 2003. Development of service-oriented products based on the inverse manufacturing concept. *Environmental Sciences & Technology*, 37(23): 5398–5406.

Gansler, J.S. and Lucyshyn, W., 2006. *Evaluation of Performance Based Logistics* [online]. Maryland University College Park Centre for Public Policy and Private Enterprise. Available from: http://handle.dtic.mil/100.2/ADA536805 [Accessed 27th June 2011].

Goulding, C., 2004. Grounded theory, ethnography and phenomenology - a comparative analysis of three qualitative strategies for marketing research. *European Journal of Marketing*, 39(3/4): 294-308.

Graedel, T.E., 1997. Designing the Ideal Green Product: LCA/SLCA in Reverse. *The International Journal of Life Cycle Assessment*, 2(1): 25–31.

Grimm, V., Revilla, E., Berger, U., Jeltsch, F., Mooij, W.M., Railsback, S.F., Thulke, H.H., Weiner, J., Wiegand, T. and DeAngelis, D.L., 2005. Pattern-oriented modelling of agent-based complex systems: lessons from ecology. *Science*, 310(5750): 987-991.

Guessoum, Z. and Briot, J.P., 1999. From active objects to autonomous agents. *IEEE Concurrency*, 7(3): 68-76.

Han, S., Park, M. and Pena-Mora, F., 2005. Comparative Study of Discrete-Event Simulation and System Dynamics for Construction Process Planning, *In: Construction Research Congress,* San Diego, USA.

Hara, T., Arai, T. and Shimomura, Y., 2006. A concept of service engineering: a modelling method and a tool for service design. *In: Proceedings of the international conference on service systems and service management*, 25th–27th October, Troyes, France, 13–18.

Hara, T., Arai, T., Shimomura, Y. and Sakao, T., 2009. Service CAD system to integrate product and human activity for total value. *CIRP Journal of Manufacturing Science and Technology*, 1(4): 262–271.

Harding, A. and Watts, P., 2000. The Northern Line train service contract. *Proceedings of the Institute of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, 214(1): 55-60.

IFS, 2010. *BAE Systems and IFS in PBL* [online]. Available from: www.ifsworld.com/.../BAE%20Systems%20and%20IFS%20in%20PBL/BAE%20Systems%20%20IFS%20in%20PBL.pdf [Accessed 27th June 2011].

Jacopino, A., 2007. Modelling R&M in Performance Based Contracts-When does risk equal reward?. *In: the proceedings of Reliability and Maintainability Symposium*.

Jennings, N.R., 2001. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4): 35-41.

Johnson, P. and Harris, D., 2002. Qualitative and Quantitative Issues in Research Design. *In:* D. Partington, ed. *Essential Skills for Management Research*. London: Sage, 99-116.

Kececioglu, D.B., 1995. *Reliability engineering handbook*. Prentice Hall.

Kendall, E.A., 1998. Agent Roles and Role Models: New Abstractions for Multiagent System Analysis and Design. *In: Proceedings of the International Workshop on Intelligent Agents in Information and Process Management*, Bremen, Germany.

Kilpia, J., Töyli, J. and Vepsäläinen, A., 2009. Cooperative strategies for the availability service of repairable aircraft components. *International Journal of Production Economics,* 117(2): 360–370.

Kim, Y.S., Wang, E., Lee, S.W. and Cho, Y.C., 2009. A product-service system representation and its application in a concept design scenario. *In: Proceedings of the CIRP IPS2 conference 2009*, 1st-2nd April, Cranfield, UK.

Komoto, H., Tomiyama, T., Nagel, M., Silvester, S. and Brezet, H., 2005. Life cycle simulation for analyzing product service systems. *In: Proceedings of the fourth international symposium on environmentally conscious design and inverse manufacturing*, 12th–14th December, Tokyo, 386–393.

Komoto, H. and Tomiyama, T., 2008. Integration of a service CAD and a life cycle simulator. *CIRP Annals – Manufacturing Technology*, 57(1): 9–12.

Kowalkowski, C., 2006. *Enhancing the Industrial Service Offering*. Thesis (PhD). Linköping University.

Kozanidis, J. and Skipis, A., 2006. Flight and maintenance planning of military aircraft for maximum fleet availability: a bio objective model. *In: Proceedings of the Multiple Criteria Decision Making*, 19th -23rd June, Chania, Greece.

Khumboon, R., Kara, S., Manmek, S. and Kayis, S., 2009. Environmental Impacts of Rental Service with Reconditioning - A Case Study. *In: Proceedings of the CIRP IPS2 conference 2009*, 1st-2nd April, Cranfield, UK.

Kuo, T.C., 2011. Simulation of purchase or rental decision-making based on product service system. *The International Journal of Advanced Manufacturing Technology*, 52(9–12): 1239–1249.

Lanza, G., Behmann, B., Werner, P. and Vöhringer, S., 2011. Simulation of Life Cycle Costs of a Product Service System. *Functional Thinking for Value Creation*, DOI: 10.1007/978-3-642-19689-8_29, 159-164.

Law, A.M., 2007. *Simulation modelling and analysis.* McGraw-Hill, New York.

Lee, Y.H., Cho, M.K., Kim, S.J. and Kim, Y.S., 2002. Supply chain simulation with discrete-continuous combined modelling. *Computer and Industrial Engineering*, 43(1–2): 375–392.

Lin, J., 2010. *An analytical framework for the development of product service systems-application of GTST-MLD and UML.* Thesis (MSc). The National Central University of Taiwan.

Lindahl, M., Sakao, T. and Öhrwall-Rönnbäck, A., 2009. Business Implications of Integrated Product and Service Offerings. *In: Proceedings of the 1st CIRP Industrial Product-Service Systems (IPS2) Conference*, 1st -2nd April, Cranfield, UK.

Lindahl, M., Sundin, E., Rönnbäck, A.Ö., Ölundh, G. and Östlin, J., 2006. Integrated product and service engineering-the IPSE project: changes to sustainable consumption. *In: Proceedings of the Changes to Sustainable Consumption, Workshop of the Sustainable Consumption Research Exchange Network,* Copenhagen, Denmark.

Low, M.K., Lamvik, T., Walsh, K. and Myklebust, O., 2000. Product to service eco-innovation: the TRIZ model of creativity explored. *In: Proceedings of the IEEE international symposium on electronics and the environment*, 8th–10th May, California, 209–214.

Lu, D., 2010. *Environmental life cycle driven decision making in product design*. Thesis (PhD). Georgia Institute of Technology.

Macal, C. M. and North, M. J., 2010. Tutorial on agent-based modelling and simulation. *Journal of Simulation*, 4: 151–162.

Macbeth, D.K. and de Opacua, A.I., 2010. Review of Services Science and possible application in rail maintenance*. European Management Journal*, 28(1): 1-13.

Macy, M. W. and Willer, R., 2002. From factors to actors: Computational sociology and agent-based modelling. *Annual Review of Sociology*, 28: 143–166.

Mahon, D., 2007. Performance-Based Logistics: Transforming Sustainment. *Journal of Contract Management,* 53-71.

Malleret, V., 2006. Value creation through service offers. *European Management Journal*, 24(1): 106-116.

MAN Truck & Bus UK., 2010. *MAN Truck & Bus UK services* [online]. Available from: http://www.man-mn.co.uk/en/Services/Services.jsp [Accessed 1 June 2010].

Manzini, E. and Vezzoli, C., 2003. A strategic design approach to develop sustainable product service systems: examples taken from the 'environmentally friendly innovation' Italian prize. *Journal of Cleaner Production*, 11(8): 851–857.

Maussang, N., Sakao, T., Zwolinski, P. and Brissaud, D., 2007. A model for designing product-service systems using functional analysis and agent based model. *In: Proceedings of the international conference on engineering design* ICED'07, 28th–31st August, Paris, France.

Meier, H., Roy, R. and Seliger, G., 2010. Industrial Product-Service Systems—IPS2. *CIRP Annals - Manufacturing Technology,* 59: 607-627.

Mont, O., 2002. Clarifying the concept of product-service system*. Journal of Cleaner Production*, 10(3): 237–245.

Mont, O., 2000. *Product-Service Systems: Final report*. Stockholm, Sweden: Swedish Environmental Protection Agency.

Morecroft, J. and Robinson, S., 2005. Explaining Puzzling Dynamics: Comparing the Use of System Dynamics and Discrete-Event Simulation. *In: Proceedings of the 2005 International Conference of the System Dynamics Society*, Boston, USA.

Morelli, N., 2006. Developing new product service systems (PSS): methodologies and operational tools. *Journal of Cleaner Production*, 14(17): 1495–1501.

Morelli, N., 2002. Product-service systems, a perspective shift for designers: a case study: the design of a telecentre. *Design Studies*, 24(1): 73–99.

Muller, P. and Blessing, L., 2007. Development of product-service-systems – comparison of product and service development process models. *In: Proceedings of the international conference on engineering design ICED'07*, 28th–31st August, Paris, France.

Neely, A., 2008. Exploring the financial consequences of the servitization of manufacturing. *Operations Management Research*, 1(2): 103–108.

Nelson, E., 2009. How Interface innovates with suppliers to create sustainability solutions. *Global Business and Organizational Excellence*, 28(6): 22–30.

Ng, I.C.L., Williams, J. and Neely, A., 2008. *Service Transformation and the New Landscape of Performance-based Contracting: An Executive Briefing* [online]. Centre for Service Research, University of Exeter. Available from: https://eric.exeter.ac.uk/repository/handle/10036/48236 [Accessed 22/12/2011].

North, M.J. and Macal, C.M., 2007. Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modelling and Simulation. Oxford, UK: Oxford University Press.

Oliva, R. and Kallenberg, R., 2003. Managing the transition from products to services. *International Journal of Service Industry Management*, 14(2): 160-172.

Pall, R., 2008. On the Availability of the CH149 Cormorant Fleet. *In: Proceedings of the 40th Conference on Winter Simulation*, Austin, TX, USA.

Phumbua, S. and Tjahjono, B., 2011a. Towards Product-Service Systems modelling: a quest for dynamic behaviour and model parameters. *International Journal of Production Research*, DOI: 10.1080/00207543.2010.539279.

Phumbua, S. and Tjahjono, B., 2011b. Simulation Modelling of Availability Contracts. *In: Proceedings of the 44th CIRP International Conference on Manufacturing Systems*, June 1st-3rd, Madison, WI, USA.

Phumbua, S. and Tjahjono, B., 2010. Simulation Modelling of Product-Service Systems: the Missing Link. *In: Proceedings of the 36th International MATADOR Conference*, July 14th-16th, Manchester, UK.

Pratt & Whitney, 2009. *Pratt & Whitney and Jet2.com Gear Up for Installation of First GMS Life Limited Parts for CFM56-3 Engine* [online]. Available from: http://www.pw.utc.com/media_center/press_releases/2009/04_apr/4-23-2009_10412146.asp [Accessed 26th June 2011].

Pratt & Whitney, 2011. *Pratt & Whitney Receives Maintenance Contract Extension to Support F100 Engines for Italian Air Force* [online]. Available from: http://www.pw.utc.com/media_center/press_releases/2011/03_mar/3-31-2011_00002.asp [Accessed 21st June 2011].

Quayzin, X. and Arbaretier, E., 2009. Performance modelling of a surveillance mission. *In: the Proceedings of Reliability and Maintainability Symposium RAMS*, Fort Worth, TX, USA.

Rabelo, L., Helal, M., Jones, A. and Min, H.S., 2005. Enterprise simulation: A hybrid system approach. *International Journal of Computer Integrated Manufacturing*, 18(6): 498–508.

Rajasekar, S., Philominathan, P. and Chinnathambi, V., 2010. *Research Methodology* [online]. Physics Educations. Available from: http://arxiv.org/abs/physics/0601009v2 [Accessed 27th June 2011].

Richardson, D. and Jacopino, A., 2006. Use of R&M measures in Australian Defence Aerospace Performance-based Contracts. *In: Proceedings of Reliability and Maintainability Symposium*, 331-336.

Robson, C., 2002. *Real world research.* Oxford: Blackwell Publishing.

Robinson, S., 2010. Modelling service operations: A mixed discrete-event and agent-based simulation approach. *In: the Proceedings of the Operational Research Society Simulation Workshop 2010 (SW10)*, Birmingham, UK.

Robinson, S., 2004. Simulation: The Practice of Model Development and Use: John Wiley & Sons.

Rolls-Royce, 2010. *Rolls-Royce signs £690 million contract to support UK Tornado fleet* [online]. Available from: http://www.rolls-royce.com/defence/news/2010/100407_contract_support_uk_tornado_fleet.jsp [Accessed 21st June 2011].

Roy, R. and Cheruvu, K.S., 2009. A competitive framework for industrial product-service systems. *International Journal of Internet Manufacturing and Services*, 2(1): 4–29.

Sakao, T. and Shimomura, Y., 2005. A novel design methodology for services to increase value combining service and product based on service engineering. *In: Proceedings of the 4th international symposium on environmentally conscious design and inverse manufacturing*, 12th–14th December, Tokyo, 402–409.

Sakao, T. and Shimomura, Y., 2007. Service Engineering: a novel engineering discipline for producers to increase value combining service and product. *Journal of Cleaner Production*, 15(6): 390–604.

Schieritz, N. and Größler, A., 2003. Emergent Structures in Supply Chains –A Study Integrating Agent-Based and System Dynamics Modelling. *in: Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, Washington, USA.

Scholl, H.J., 2001. Looking across the fence: Comparing findings from SD modelling efforts with those of other modelling techniques. *In: Proceedings of the 19thInternational Conference of the System Dynamics Society*, Atlanta.

Schuh, G., Boos, W. and Kozielski, S., 2009. Life cycle cost-oriented service models for tool and die companies. *In: Proceedings of the CIRP IPS2 conference*, 1st–2nd April, Cranfield, UK, 249.

Schuh, G. and Gudergan, G., 2009. Service engineering as an approach to designing industrial product service system. *In: Proceedings of the CIRP IPS2 conference*, 1st–2nd April, UK.

Schumann, B., Scanlan, J.P. and Takeda, K., 2011. A Generic Operational Simulation for Early Design Civil Unmanned Aerial Vehicles. *In: Proceedings of the 3rd International Conference on Advances in System Simulation*, Barcelona, Spain.

Shehory, O. and Sturm, A., 2001. Evaluation of modeling techniques for agent-based systems. *In: Proceedings of the International Conference on Autonomous Agents*, New York, USA.

Shen, J. and Wang, L., 2008. A methodology based on fuzzy extended quality function deployment for determining optimal engineering characteristics in product-service system design. *In: Proceedings of 2008 IEEE international conference on service operations and logistics and informatics*, 2nd–15th October, Beijing, China, 331–336.

Shostack, G.L., 1982. How to design a service. *European Journal of Marketing*, 16(1): 49–63.

Siebers, P.O., Macal, C.M., Garnett, J., Buxton, D. and Pidd, M., 2010. Discrete-Event Simulation is Dead, Long Live Agent-Based Simulation. *Journal of Simulation*, 4(3): 204-210.

Simon, M., Bee, G., Moore, P., Pu, J. and Xie, C., 2000. Life cycle data acquisition unit - design, implementation, economics and environmental benefits. *In: Proceedings of the IEEE Symposium on Electronics and the Environment*, Piscataway, NJ, USA, 284–289.

Stranjak, A., Dutta, P.S., Ebden, M., Rogers, A. and Vytelingum, P., 2008. A multi-agent simulation system for prediction and scheduling of aero engine overhaul. *In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, 12th -16th May, Estoril, Portugal.

Sterman, J.D., 2000. Business dynamics: systems thinking and modelling for a complex world. Boston, USA: Irwin/McGraw-Hill.

Stubbs, W. and Cocklin, C., 2008. An ecological modernist interpretation of sustainability: The case of Interface Inc. *Business Strategy and the Environment*, 17(8): 512-523.

Tako, A.A. and Robinson, S., 2009. Comparing discrete-event simulation and system dynamics: users' perceptions'. *Journal of the Operational Research Society*, 60(3): 296-312.

Tukker, A., 2004. Eight types of product-service system: eight ways to sustainability? Experiences from SusProNet. *Business Strategy and the Environment*, 13(4): 246–260.

Tukker, A. and Tischner, U., 2004. New business for old Europe. *Final report of SusProNet*. TNO-STB, Delft, the Netherlands.

Wakeland, W.W., Gallaher, E.J., Macovesky, L.M., and Aktipis, C.A., 2004. A comparison of system dynamics and agent-based simulation applied to the study of cellular receptor dynamics. *In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 86-95.

Watson, E.F., Chawda, P.P., McCarty, B., Drevna, M.J. and Sadowsk, R.P., 1998. A simulation metamodel for response-time planning. *Decision Sciences*, 29: 217–241.

Weber, C., Steinbach, M., Botta, C. and Deubel, T., 2004. Modelling of product-service systems (PSS) based on the PDD approach. *In: Proceedings of the international design conference-Design 2004*, 18th–21st May, Dubrovnik, Croatia, 547–554.

Xerox, 2010a. *Improving enterprise print services to achieve demanding business objectives* [online]. Available from: http://www.xerox.co.uk/consulting/case-studies/engb.html [Accessed 26th June 2011].

Xerox, 2010b. *Xerox document service* [online]. Available from: http://www.consulting.xerox.com/ [Accessed 26th June 2011].

Xerox, 2010c. *Customer Service Agreement Exhibit A* [online]. Available from: www2.dir.state.tx.us/DIR_Contracts/1243454341976.pdf [Accessed 26th June 2011].

Yin, R.K., 2009. *Case study research: design and methods.* Sage Publications.

Yu, T.T., 2008. *The Development of a Hybrid Simulation Modelling Approach Based on Agents and Discrete-Event Modelling*. Thesis (PhD). University of Southampton.

Appendices

A. Introduction to software elements

This section outlines commonly used elements in the models developed in this thesis, based in AnyLogic 6.5. The tool is developed from object-oriented concept which supports ABS, and enables SD and DES. The first part of this section introduces modelling techniques and their representations in the software. The second part highlights the functions and pitfalls that the modellers should keep in mind during model development to avoid errors during the compiling.

Modelling elements (as presented in Figure A-1) are in correspondence with SD, statechart, DES, or some commonly-used objects. The modelling elements corresponding to these techniques are presented.

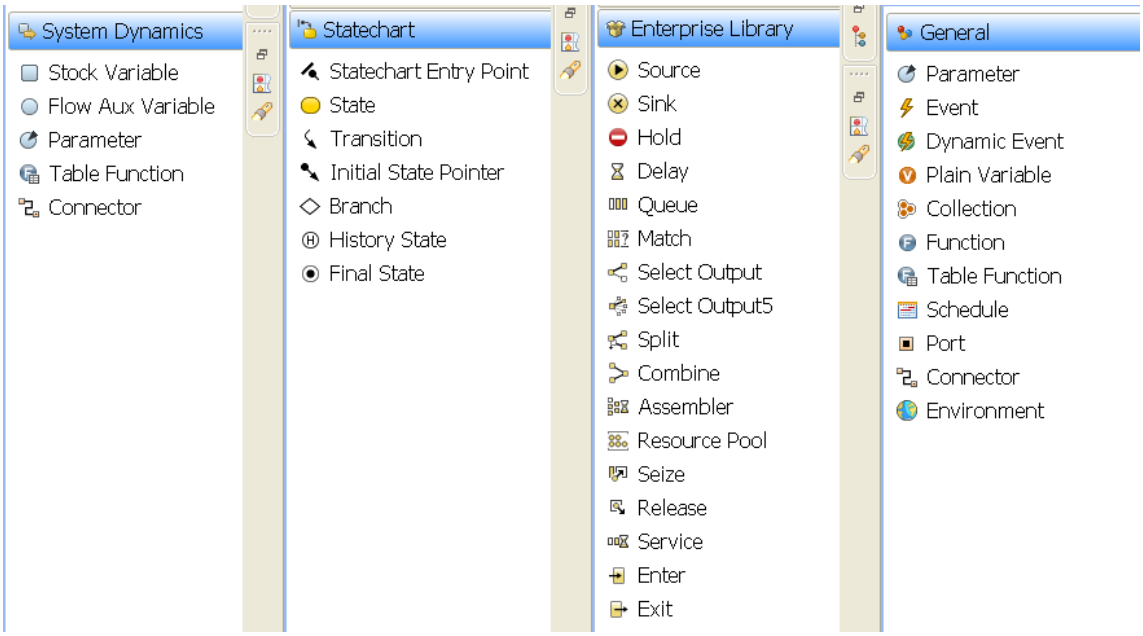


Figure A-1: Commonly used software elements

SD is ideal to study interconnected system. Generally, the frequently used elements in SD are stock variables, flow variables, and parameters. The stock can be seen as to the major variations in a system, whose continuous pattern of changes (flow variable) is governed by some factors (parameter). For instance, a model looks at the ratio between traditional manufacturers and product-service manufacturers. Manufacturers can change from one group to the other as a result of the pressure from the low cost economies. In this case, each group is represented by a stock variable, the transition

between the two groups refers to a flow variable, and the pressure depicts a parameter.

Besides the direct representation of systems, a flow variable can substitute a normal variable to enable continuous value updates. Nonetheless, the feature can slow down model execution tremendously. Therefore, it has been used only if it is unnatural to use other modelling techniques, or as a supplement when the model scale is small.

Even though state modelling also captures state changes, the change is modelled from an individual perspective (not a system) and can be governed by other events (e.g. upon receiving a message) in addition to rate of occurrences. The 'Statechart' elements are rarely used to perform other functions like the flow variable.

DES technique naturally represents a sequence of processes and queues in a system. It is generally used to study impacts of interconnected uncertainties. Although there are several DES elements provided by the software package, commonly ones in this thesis are as follows:

- *Enter* is performed as a gateway that allows an asset to get inside the system.
- *Queue* represents assets waiting to be serviced.
- *Selected output* separates assets for different services and customers.
- *Hold* is activating due to unavailable resources, off-shifts, or non-contracted hours.
- *Service* refers to a process used when the absence of a resource affects the problem of study.
- *Resource pool* represents resource used when the difference among entities in the resource is not significant to the study.
- *Delay* also refers to a process but used when the absence of a resource can be handled.
- *Sink* completes OEM service process.

Modelling an actor is not limited to the modelling techniques mentioned above, but also includes simple event, collection, parameter, or variable. Parameters are the inputs to the model, including actor's attributes. Collection can be used to store a message and ensure that no message is dropped if the agent is already occupied by the former message. Events (time-based or condition-based) are useful to initiate actions and update status. Nonetheless, several cares must be taken using this element. Particularly with a condition-based event, the condition must not embed the function

time(), otherwise the model can stop advancing. The alternative can be done using a single step cyclic event with the conditional action code since it ensures time progressing. However, additional code is required to guard action repetitions, which often leads to wrong outputs. Besides, it can slow down the model execution since the event is scheduled more frequent than necessary, and can be too excessive in many cases. This method is recognised as synchronous programming. An additional issue is related to the parameter types (e.g. boolean, double, integer), which must be match when a comparison between them is made. A condition is required if a situation can lead to the zero value of denominators.

B. SD models

Introduction

Prior to offering a particular contract to customers, it is vital for OEMs to realise the current capability in sustaining the contract, so that the essential and lagging capability can be improved or the offer can be modified. The capability can be measured in terms of operational capability and network managing capability. However, the major challenges are exposed from the dependencies among offering factors, operational factors and networking factors. Thus, the first important step is to realise which factor has the greatest influences to the offering. Therefore, this study aimed to study levels of influences each factor have on the entire system.

Methodology

This study was carried out jointly with a group project at Cranfield University which aimed to develop general PSS business models.

Based on the review in Section 4.1, the research context was matched with simulation techniques. The criteria were based on the nature of the problem, as highlighted by Siebers et al. (2010). Subsequently, DES was not chosen as 1) the complexity in the system is not arisen from randomness 2) the outputs are not designed to influence the inputs in DES whereas the offering and the capability tend to have two-way relations. On the contrary, SD incorporates the feedback feature that allows a two-way relation to be examined.

A number of factors related to PSS offers, operational capability and network capability were selected from the literature (described in Section 2.3.1) and refined with several PSS experts. These key variables include:

Customer value: This is linked with the 'value in use', which affects the success/failure in delivering service contracts directly.

Service offer: This is a key factor for customers to value service contracts.

Product offer: This is another key factor which influences the customer value, particularly in the product-centric PSS context.

Monitoring technology: The technology (e.g. pressure sensor, trouble shooting) enables the OEM to receive information quicker and realise the cause of asset failures, thus, it enhances the responsiveness and product innovation.

Skilled workforce: Staff's skill influences the quality of services and customer involvements in the product-service improvements.

Customer involvements: This factor enhances relationships with customers, **and enables the OEM to** receive feedback for product-service improvements in a timely manner.

Supplier involvements: This factor affects product availability directly, and enables the OEMs to commit their suppliers in improving product quality and carrying out some service activities (e.g. detail inspection of a product's component) to improve service availability.

Customer satisfaction: Ultimately, a contract can be sustained if the customer satisfies with the offer and the OEM's performances. Therefore, this variable is crucial.

Shareholder satisfaction: The PSS concept requires cooperation within the entire supply chain. Consequently, the shareholder satisfaction is an enabling factor in sustaining service contracts.

In general, a reference model is required in developing an SD model after having key variables identified. However, this study excluded the reference model as there was no available numerical report of the interconnected behaviour. Besides, the behaviour is expected to change across cases. Three qualitative SD models (i.e. influence diagrams) were developed during the project: traditional business model, intermediate-servitisation business model, and advance-servitisation business model. The intermediate level refers to the product-service offers which entail services to enhance product sales. An example of the intermediate sertitisation includes a warranty, whereas the advance level dictates the integrated offer such as Rolls-Royce's power by hour. In addition to the qualitative SD models, a stock and flow diagram was formulated to assess the impact of each factor on attracting more customers.

Three case studies were conducted within two water companies, two train manufacturers, and two wind turbine companies to validate the models. The team introduced the project and interviewed the industrial representatives in terms of their service offering mechanism during the company visits. Using PowerPoint presentation of the model, the interviewees were asked to give feedback and the weight importance of each factor in attracting their customers. In addition to the case studies, the models validated by the Integrated Vehicle Health Management (IVHM) community and PSS experts on a frequent basis.

After the project, the models were modified, as shown in Figure B-1. The influence diagram shows the feedback structure of service contracting. To illustrate, an increase in the customer value can attract more demands, and the rise in the demands necessitates capacity expansion, hence, more service facilities. This, in turn, improves service responsiveness, and hence, customer satisfaction. The satisfied customers tend to develop a good relationship with the OEM, which allows the OEM to receive useful feedback to improve the right product/service. Eventually, the improvements can increase the customer value, and so on.

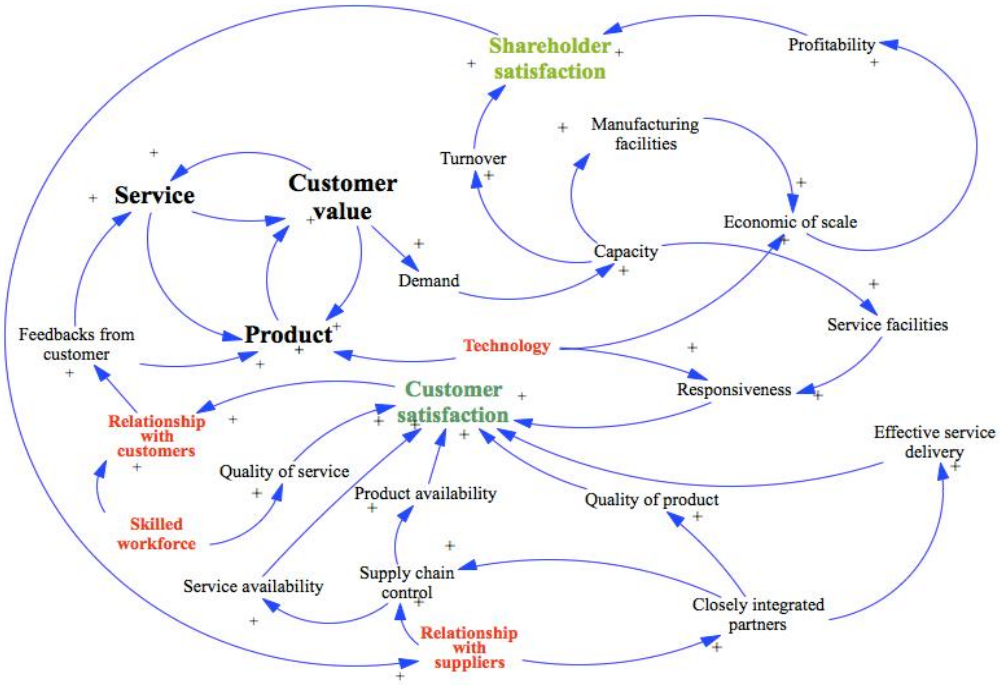


Figure B-1: An influence diagram of key enablers for sustaining service contracts

The influence diagram was converted to a stock and flow diagram, as represented in Figure B-2. In the Figure, general customers (*PotentialCustomers*) are attracted for service contracting and become contracted customers (*LongTermCustomers*) as a function of the customer value. This value is influenced by technology, skilled workforce, customer involvements and supplier involvements. A contracted customer can also leave the contract as a result of a decrease in the four factors and after the contract ends. The time plot captures the changes in the number of long term customers and potential customers in approximately 12 years period. The weight importance of the variables can be input by model users, depending on the case. Similarly, the values of the four factors can be adjusted by the users.

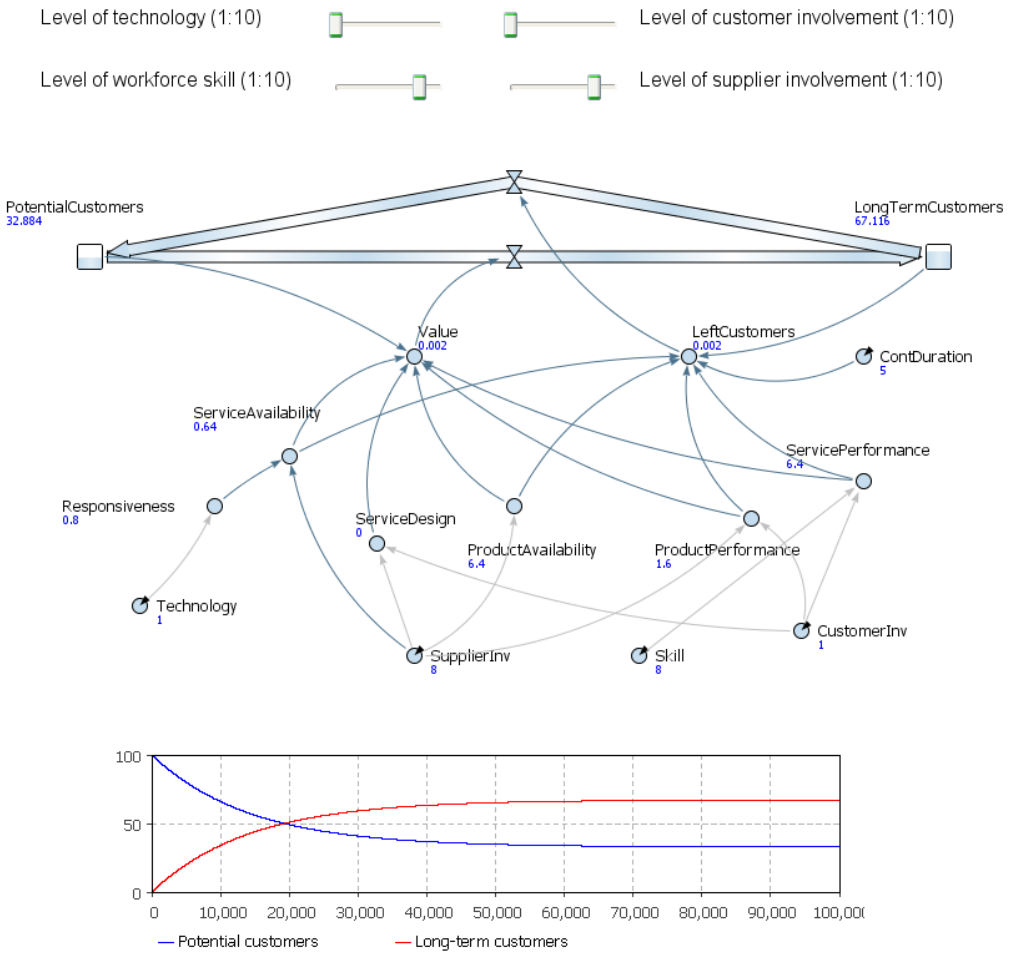


Figure B-2: A stock and flow diagram of key enablers for sustaining service contracts

Experimentation

To demonstrate the use of the model, a case from a wind turbine company is illustrated. In this case, the manufacturer traditionally sold a wind turbine to an energy company which was responsible for maintenance by itself. Currently, a maintenance service is performed in a reactive manner where technicians need to repair the turbines from a helicopter. Consequently, the task was a burden for the energy company and also involves safety issues. For this reason, the manufacturer is moving toward an integrated offering in which an availability of a wind turbine is guaranteed but the turbine belongs to the OEM. In other words, the OEM can lease a turbine to several companies on a product availability basis and carry out necessary maintenance services.

Based on this requirement, the *product availability* becomes the key variable for an energy company to sign a contract and leave the contract. On the contrary, *service design* has no influence as the service offering is already tied up with maintenance. Along this line, turbine specifications (i.e. *product performances*) has few influences in attracting the energy companies as it depicts the amount of energy a turbine can transform. *Service availability* and *performance* can have a greater impact as they depict how quick the OEM responds and recovers a malfunctioned turbine. As a result, the weight importance of these variables was given to the *Value* equation as 0.5, 0, 0.1, 0.2, and 0.2 respectively. In terms of the *LeftCustomers* equation, as a customer can leave a contract only if the guaranteed product availability is not achieved, the weight importance was input as 1, 0, 0, 0, and 0 respectively. A similar analysis was conducted to derive the weights for other 'Auxiliary variables'.

In terms of parameter scaling, the initial setting was set to the *technology, skilled workforce, customer involvements and supplier involvements*, as 1, 8, 1, 8 respectively. These values were based on the interpretation of the company's current capability. Potential and contracted customers were initialised as 100 and 0, respectively.

The first experiment aimed to demonstrate how the model can be used for investment strategy. In the first option, the OEM considers improving the monitoring technology for 70%, whilst the alternative strategy is to arrange more events with customers for a 70% stronger relationship. Accordingly, the technology's slider and the customer involvements' slider were set to 3 in the first and second run respectively. The term of contract is input as 5 years in all cases. The results are shown in Figures B-3 and B-4.

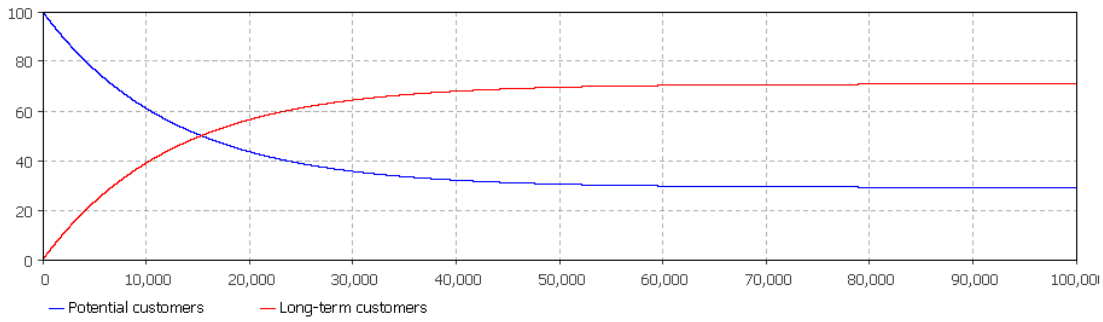


Figure B-3: The result from increasing 70% of monitoring technology

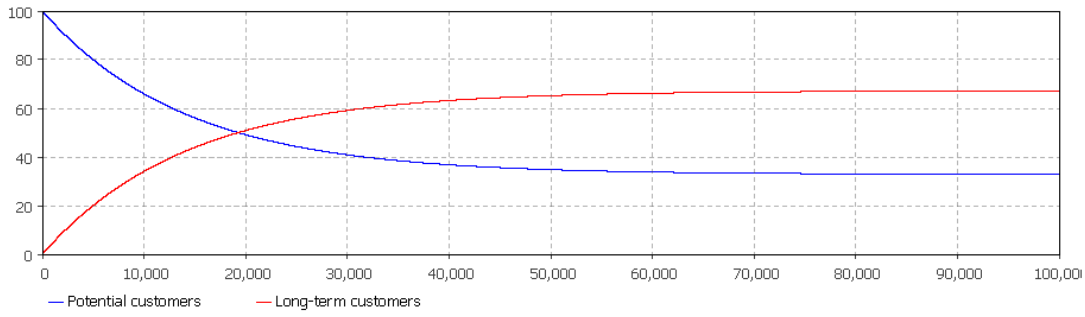


Figure B-4: The result from increasing 70% of customer involvements

These results reveal that improving technology capability can attract customers more quickly than increasing customer involvements. Nonetheless, there is no significant difference in terms of contracted customers.

The next experiment intended to illustrate the use of the model in designing service contracts. In this run, the setting was based on the improved technology experiment (i.e. Figure B-3), but the contract duration were input as ten years. The result is presented in Figure B-5, which shows an increase in contracted customers. Therefore, the OEM should extend the contracts to 10 years.

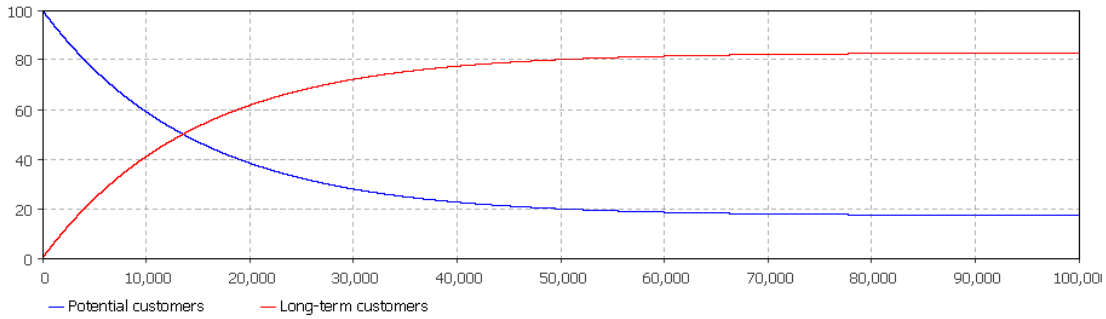


Figure B-5: The result from offering contracts on a 10-years basis

In conclusion, the experiments demonstrated the model's capability in realising a potential investment strategy and contracting strategy, as well as the influence levels among operational capability, network capability, and PSS offerings.

C. DES model

Introduction

The definition of PSS defined by Baines et al. (2007) emphasises the use aspect of asset. Existed contracts also indicate the significance of usage requirement, for example, up to 140 flying hours per week in the case of aircraft (BAE Systems, 2010), approximately 40 degree EGT margin in an engine case (Aircraft commerce, 2006), between 9 am and 5 pm in the case of photocopier (Xerox, 2010c), and all the time in the case of the vessel (EMSA, 2006). These examples imply that a contract can be customised on a usage basis to suit each customer need. However, it is vital for an OEM to realise if a particular usage requirement will bring benefit prior to making contracts Therefore, the aim of this study was to understand the risks and rewards from making a contract on different asset usage requirements.

Methodology

The risks in a service contract are caused by dynamic behaviour (Section 2.3.2) and can be absorbed or multiplied by management of services. Therefore, the focus of this problem naturally entails randomness and roots down to the process level, in which DES can potentially describe. For this reason, DES was chosen as the modelling technique for this study.

Three model scenarios were identified based on the usage requirements in existed contract. Assets in scenario one are contracted for continuous use throughout the contract, for example, the Tornado aircrafts are contracted for operating at any time. In the second scenario, the assets are contracted for specific duration in the contract period such as 5 am – 12 pm the case of LUL and 9 am – 5 pm in the Xerox DocuCare contract. This type of contracts allows the OEMs to monitor the asset's condition continuously outside the operating time. As a result, the chance that the assets fail unpredictably is expected less than the first scenario. Finally, the third scenario corresponds to the cyclic usage pattern with a multi-duration, for example, an airline may contract 20 aero-engines from an OEM during the peak period (i.e. December, January, April, July, August) and 12 engines for the rest of the year. In this case, the OEM can use the 8 engines elsewhere in the world to globalise seasonal effect and still keep assets at maximum utilisation, while the customer can customise and may pay less for the contract.

The risk and reward measures were proposed in correspondence with the common performance requirements. These measures comprise availability level, missed hours, man hours, and the thrown-away life of the assets.

All scenarios consist of one OEM and two business customers; the first customer (C1) requires three assets to be available, and the second customer (C2) agrees on two

assets. Asset's usage is continuously updated once being returned to the OEM. The inputs are associated with the contract hours, MTBF, service cycle times, and the number of staff. The basic unit is in hours and the model is simplified to always have available spare parts.

In summary, a conceptual model was developed as shown in Figure C-1.

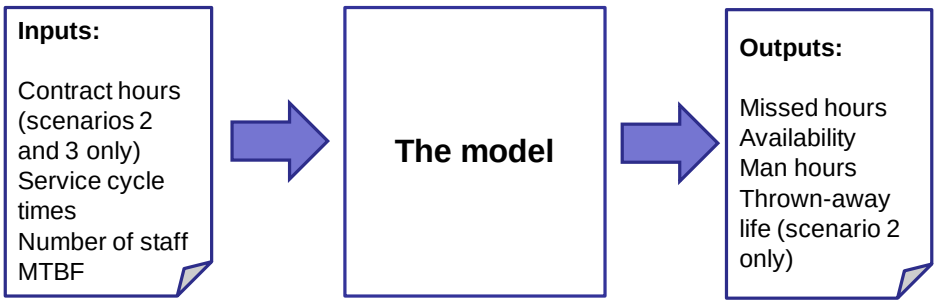


Figure C-1: Conceptual model for business case II

Model code

1. Scenario one: assets are contracted all the time.

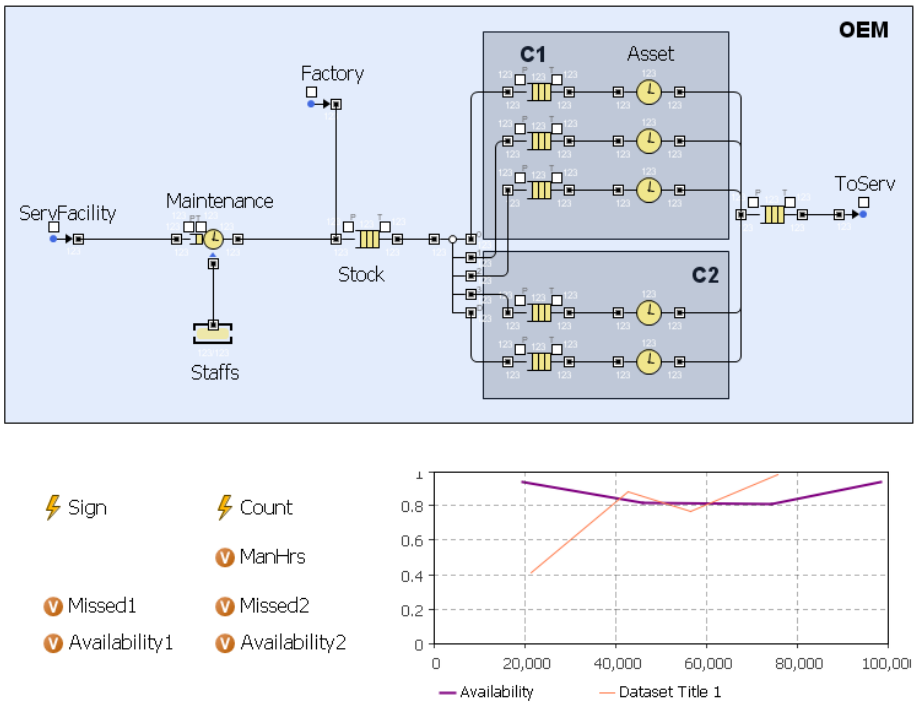


Figure C-2: Scenario 1 model

Sign:

Action:

```
//Issue 5 assets from the factory in the beginning of the model.
        for (int i=1;i<6;i++) {
                Job thisAsset = new Job();
                thisAsset.Asset=i;
                thisAsset.ServTime= 0;
                thisAsset.MTBF=(int)normal(10,5000);
                Factory.take(thisAsset);
        }
```

Count

Action:

```
//Update availability every one hour.
        if (Asset.size()==0){
                Missed1++;
        }
        if (Asset1.size()==0){
                Missed1++;
        }
        if (Asset2.size()==0){
                Missed1++;
        }
        if (Asset3.size()==0){
                Missed2++;
        }
        if (Asset4.size()==0){
                Missed2++;
        }
        Availability1 = ((3*time())-Missed1)/(3*time());
        Availability2 = ((2*time())-Missed2)/(2*time());
```

ServFacility

On enter:

```
//Define the required service cycle time of each asset.
```

entity.ServTime=normal(1,5);

Maintenance

Delay time:

//Apply the defined service duration.

entity.ServTime

On exit:

//Update service hours.

ManHrs=ManHrs+entity.ServTime;

selectedOutput5

On Condition 0: entity.Asset==5 // Return the leased asset to the leaser.

On Condition 1: entity.Asset==4

On Condition 2: entity.Asset==3

On Condition 3: entity.Asset==2

Asset

Delay time:

//i.e. in-service duration.

entity.MTBF

ToServ

On exit:

//Send the asset back to the OEM.

ServFacility.take(entity);

entity.Enter++;

2. Scenario 2: Both customers contract assets daily for 8 hours. If not stated, the code inside an element is identical to scenario 1.

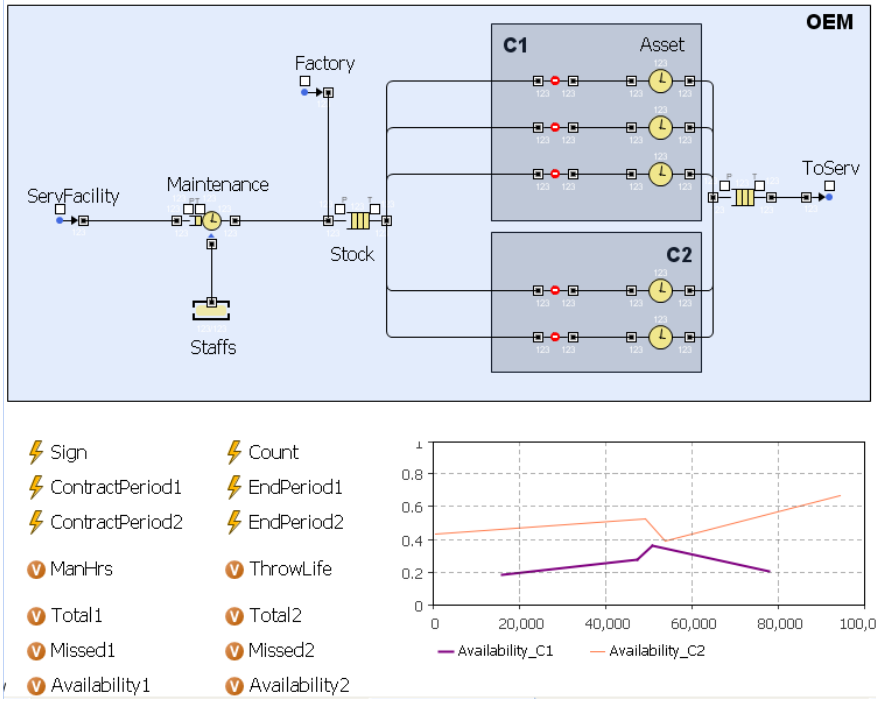


Figure C-3: Scenario 2 model

ContractPeriod1

Action:

```
//C1 retrieves 3 assets every beginning of each day.
        if (hold.isBlocked()){
                hold.setBlocked(false);
        }
        if (hold1.isBlocked()){
                hold1.setBlocked(false);
        }
        if (hold2.isBlocked()){
```

```
            hold2.setBlocked(false);
        }
//Update availability daily and reset the contract hours.
        Availability1=(Total1-Missed1)/Total1;
        RemainHrs1=8;
```

ContractPeriod2

Action:

```
//C2 retrieves 2 assets every beginning of each day.
        if (hold3.isBlocked()){
            hold3.setBlocked(false);
        }
        if (hold4.isBlocked()){
            hold4.setBlocked(false);
        }
//Update availability daily and reset contract hours.
        Availability2=(Total2-Missed2)/Total2;
        RemainHrs2=8;
```

EndPeriod1

Action:

```
// Activate the 'Hold' elements in C1 daily at the end of the contract hours.
        hold.setBlocked(true);
        hold1.setBlocked(true);
```

hold2.setBlocked(true);

EndPeriod2

Action:

//Activate the 'Hold' elements in C2 daily at the end of the contract hours.

hold3.setBlocked(true);

hold4.setBlocked(true);

Count

Action:

//Check if assets are operating on an hourly basis.

```
if (hold.isBlocked()==false){
        Total1++;
        if (Asset.size()==0){
                Missed1++;
        }
}
if (hold1.isBlocked()==false){
        Total1++;
        if (Asset1.size()==0){
                Missed1++;
        }
}
if (hold2.isBlocked()==false){
        Total1++;
```

```
            if (Asset2.size()==0){
                  Missed1++;
            }
      }
      if (hold3.isBlocked()==false){
            Total2++;
            if (Asset3.size()==0){
                  Missed2++;
            }
      }
      if (hold4.isBlocked()==false){
            Total2++;
            if (Asset4.size()==0){
                  Missed2++;
            }
      }
```

ServFacility

On enter:

```
//Replace the asset if a failure is expected soon.
        if (entity.MTBF-entity.Usage<normal(2,50)){
                ThrowLife=ThrowLife+entity.MTBF-entity.Usage;
                entity.Usage=0;
                entity.ServTime=normal(1,24);
        }else{
                entity.ServTime=normal(0.2,2);
        }
```

Hold

On enter:

```
//Input service cycle time of the followed 'Delay' and update the asset usage.
        int AveOperations = 50;
        int RealOperation = (int)normal(10, AveOperations);
        if (RealOperation + entity.Usage >= entity.MTBF) {
                if (entity.MTBF-entity.Usage<RemainHrs1){
                        entity.OpTime= triangular(0,entity.MTBF-entity.Usage);
                        entity.Usage=entity.MTBF;
                }else{
                        entity.OpTime= RemainHrs1;
                        entity.Usage=entity.Usage+RemainHrs1;
                }
```

```
}else{
        if (RealOperation>RemainHrs1){
                entity.OpTime= RemainHrs1;
                entity.Usage=entity.Usage+RemainHrs1;
        }else{
                entity.OpTime=RealOperation;
                entity.Usage=entity.Usage+RealOperation;
        }
}
```

3. Scenario 3: Variable contracted assets. Unless stated, the code for the presented elements are identical to scenario 1.

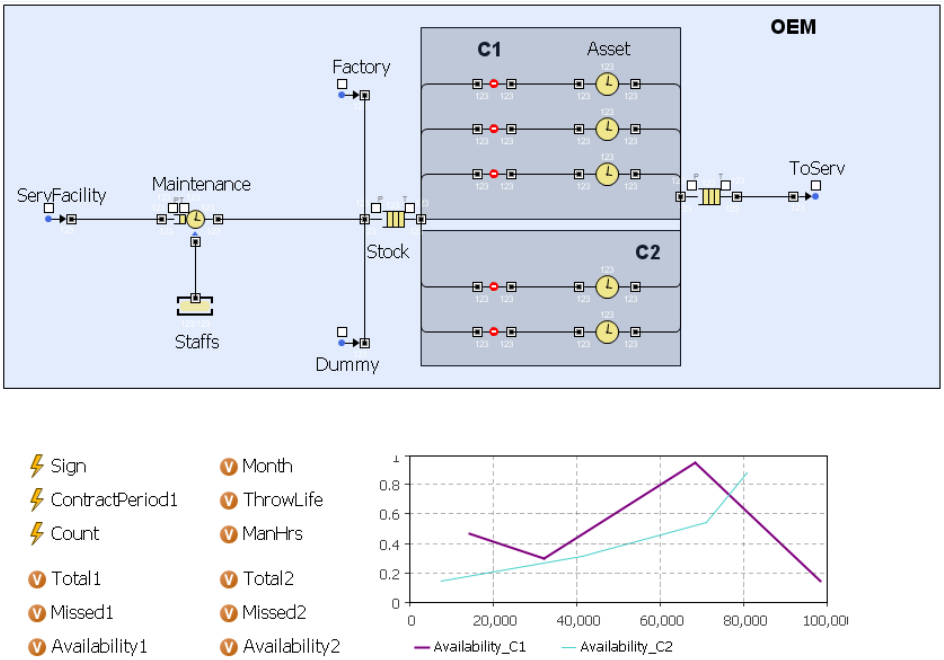


Figure C-4: Scenario 3 model

ContractPeriod1

Action:

```
// C1: 3 assets at month 1, 4, 7, 8, 12, otherwise 2 assets
```

```
// C2: 1 assets at month 1, 4, 7, 8, 12, otherwise 2 assets
        Month++;
        if ((Month==1)||(Month==4)||(Month==7)||(Month==8)||(Month==12)){
                        hold.setBlocked(false);
                        hold1.setBlocked(false);
                        hold2.setBlocked(false);
                        hold3.setBlocked(true);
                        hold4.setBlocked(false);
        }else{
                        hold.setBlocked(false);
                        hold1.setBlocked(false);
                        hold2.setBlocked(true);
                        hold3.setBlocked(false);
                        hold4.setBlocked(false);
        }
        if (Month>12){
                Month=0;
        }
        Availability1=(Total1-Missed1)/Total1;
        Availability2=(Total2-Missed2)/Total2;
```

ServFacility

```
//Replace the asset if a failure is expected soon.
        if (entity.MTBF-entity.Usage<normal(2,50)){
```

```
        entity.Usage=0;

        entity.ServTime=normal(1,24);

}else{

        entity.ServTime=normal(0.2,2);

}
```

Asset

Input an Asset from C1 and another from C2 with the cycle time of one time unit.

ToServ

//Send the failed assets to the OEM but the mission-capable asset back to the customer.

```
if (entity.MTBF-entity.Usage>0){

        Dummy.take(entity);

}else{

        ServFacility.take(entity);

}
```

Count

```
if (hold.isBlocked()==false){

        Total1++;

        if (Asset.size()==0){

                Missed1++;

        }

}

if (hold1.isBlocked()==false){

        Total1++;
```

```
            if (Asset1.size()==0){
                    Missed1++;
            }
    }
    if (hold2.isBlocked()==false){
            Total1++;
            if (Asset2.size()==0){
                    Missed1++;
            }
    }
    if (hold3.isBlocked()==false){
            Total2++;
            if (Asset3.size()==0){
                    Missed2++;
            }
    }
    if (hold4.isBlocked()==false){
            Total2++;
            if (Asset4.size()==0){
                    Missed2++;}}
```

Experiment

The models allow the implications of making a contract from different usage requirements to be investigated. To illustrate this point, three experiments were conducted to the three usage scenarios based on the inputs shown in the model code section. Practically, all three models provide an estimation of contract performances

(in terms of average asset availability, downtime, and total service duration) over a 10-years period, subject to the current number of staff, their productivity, and the asset health (i.e. MTBF). The results are presented in Table C-1.

Table C-1: Summary of experiment results

Experiment	Availability (%)		Downtime (hours)		Man hours	Thrown-away life
	C1	C2	C1	C2		
1 (scenario 1)	99.5	99.5	1458	1017	2276	-
2(scenario 2)	99.5	99.6	546	251	42202	1539
3(scenario 3)	99.9	1	244	27	1530	-
4(scenario 2)	99.5	99.5	362	320	50551	1478
5(scenario 2)	99.9	99.8	119	154	42345	1567

The results from Experiments 1 to 3 reveal that the scenario with the multi-level usage and multi-period contract can give the highest availability level and require the lowest man hours to perform services. This means the same level of operational capability can lead to different contract performances, thus, the OEM can customise the contract configuration to increase the performance level based on the OEM's current capability.

Besides the understanding of contract implication, the models can be used to evaluate investment strategies. To demonstrate, the fourth and fifth experiments were carried out based on the second usage scenario. The objective of the two experiments was for the OEM to decide a better strategy in improving contract performance: between stocking a spare asset (Experiment 4) and recruiting an additional staff (Experiment 5). To set up these experiments, the *i* variable in the action code of the *Sign* element was increased to 7 in Experiment 4, and the capacity of the *Staff* element was adjusted to 2 in Experiment 5.

The results (Table C-1) reveal that the staff recruitment strategy can lead to a better availability and downtime value. However, it can lead to an increased thrown-away life of the assets.

It should be noted that more experiment runs and sensitivity analysis should be performed in practice to obtain a more reliable solution. Nevertheless, this thesis aims to illustrate the use of the models. Therefore, the interpretations of results are made on one single run and the sensitivity analysis was not performed at this stage. Similarly, a warm-up period is usually taken into account in a simulation study. However, the warm-up period can imply contracting on new assets, which is typical in the context of service contracts. On the other hand, excluding this period can mean a contract on a used assets basis. Accordingly, a warm-up period was not excluded in the experiments of this research.

D. Hybrid SD-ABS model

Introduction

In business cases I and II, the feasibility in using single simulation techniques were explored within PSS offering context. The results reveal the capability to corporate dynamic behaviour with the passage of time, however, a few PSS characteristics could not be effectively captured. Therefore, this study attempted to enhance the modelling capability by collaborating ABS with the applied simulation techniques.

Methodology

This study was the first attempt in exploring hybrid technique in the contracting decisions, thus, the detail modelling technique was avoided. Accordingly, this study combined SD and ABS to initially explore the capability of the hybrid technique.

A modelling scenario was formulated based on an existing business case in the aircraft engine sector, due to the following reasons:

1. Aircraft engines have high value which encourages airline companies to lease them from an OEM and pay for a service contract instead of acquiring an engine.

2. An engine's overhaul and repair require high level of servicing skill and knowledge, which can become burdens for airline companies.

3. Monitoring technologies in the aero-industry are well advanced thus the prediction of asset's failure is sufficiently accurate. This means the risks can now be shifted to other external dynamic behaviour and allows them to be investigated rather than the asset's breakdown itself.

The data required for model development were considered from the literature. First, the inputs and outputs were collated from the aircraft contract literature which included Jacopino (2007), Pall (2008), and Quayzin and Arbaretier (2009). Jacopino (2007) examined profitability and risks via Monte Carlo simulation. In his study, a number of available aircrafts, categorised into three levels depending on their conditions, was used as an input parameter. Operating hours of an aircraft were used as a unit of inputs. Also via Monte Carlo, Quayzin and Arbaretier (2009) evaluated helicopter availability, penalty, and corrective/preventive maintenance cost, with subject to the actual asset usage. Unlike the others, Pall (2008) investigated the level of aircraft availability using DES based on the actual running hours. These literature findings on inputs and outputs were used as a basis in developing measures for this study.

On top of the measures, the literature associated with aircraft maintenance was investigated to acquire and develop the structure in the model. Typically, an aircraft constitutes three major systems; avionic for communication, engines, and structural components such as wings. An airline often has their own MRO but subcontract the availability of the components and general supports to business partners (Kilpia, 2008). It was reported in Bazargan and McGrath (2003) that an aircraft was scheduled to A,B or C checks after 50, 100, 600 of flight hours, and it could take 2 hrs, 2-4 hrs or more than 4 hours for the minor, medium, and major repair respectively. The first level of maintenance check, carried out on site, involves inspection, repair and parts replacement. The second level performs deeper inspections and analysis, and the third level necessitates high skill professionals to perform thorough repair and replace (Kozanidis and Skipis, 2008).

The conceptual model is represented in Figure D-1. The model encompasses service efficiency measures such as availability level and delay hours, and separates assets as another layer embedded under the OEM layer. Within the asset layer, its state is exposed using state chart modelling which is one type of UML (Section 2.4.1). Its state triggers the OEM service processes modelled using SD to visualise the amount of workload (i.e. engines queuing for maintenance). The outcome from this model enables the OEM to estimate contract performances with subject to the given resources, maintenance schedule, and external risks that cause engine failure such as bird strike.

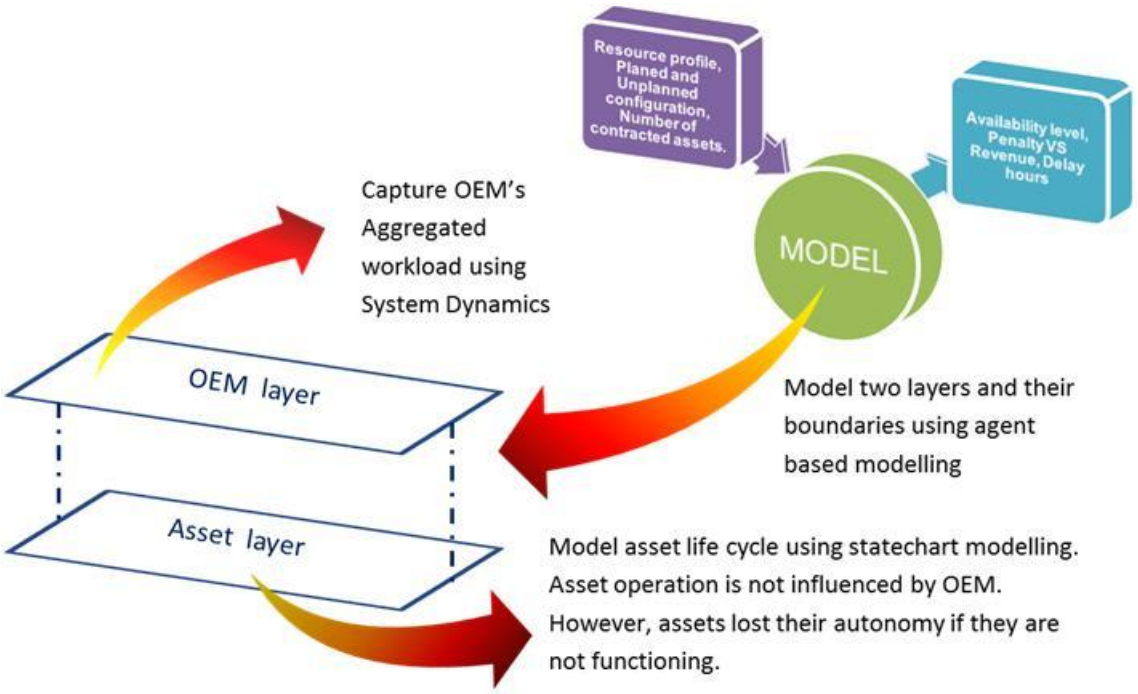


Figure D-1: Business case III's conceptual model

The high level model mechanism is illustrated using BPMN as shown in Figure D-2. The maintenance schedule and the external causes of failure are input by users and stored in the simulation engine that, in turns, keeps monitoring these events. Generally, an asset is operating according to an operating schedule that is independent from the OEM, unless a trigger is received by the simulation engine. Once this happens, the asset changes its state to maintenance and further activates the OEM service's process. Once the OEM finishes servicing, the asset is triggered back to the operating state.

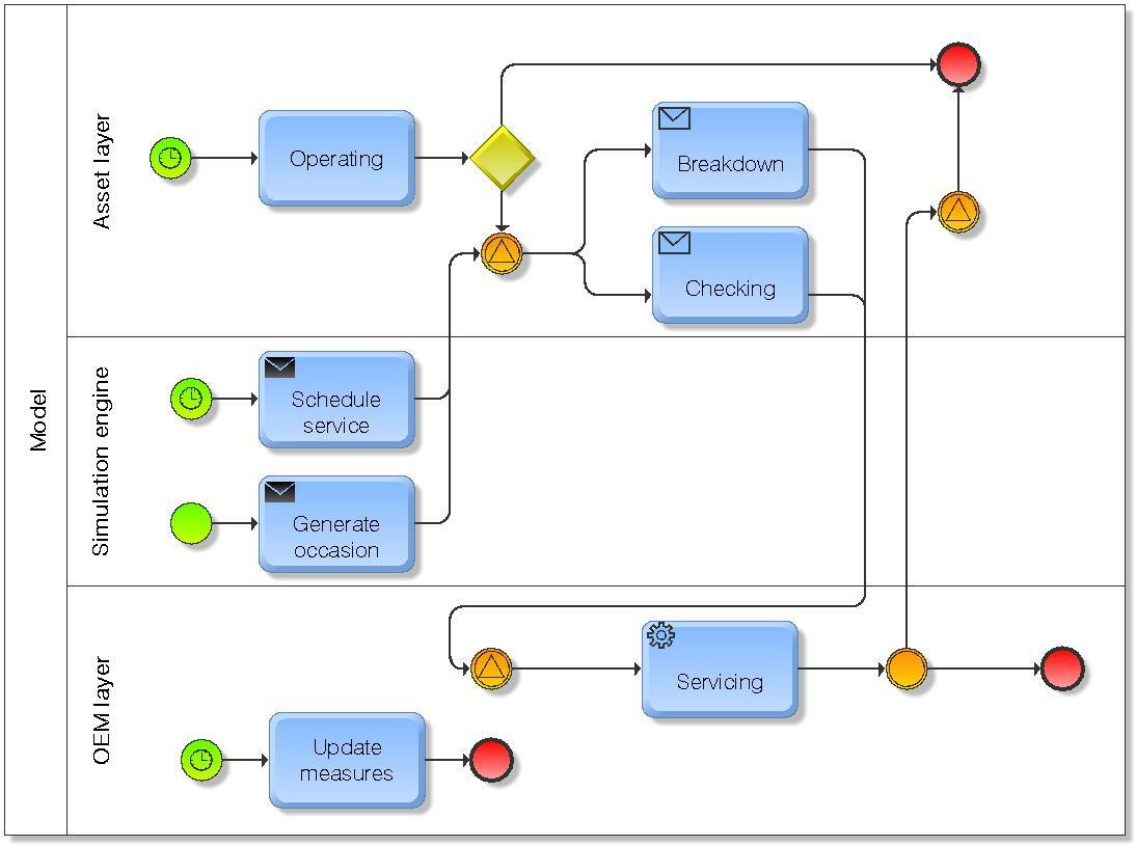


Figure D-2: High level agent behaviour and their interactions

As a result, the agent models were formulated as presented in Figure D-3, D-4 The OEM is responsible for all maintenance activities, in which service hours required to finish the task are customized on tasks, servicing skill and the number of technician carrying out the task (input by the user). The stock *A, B ,C*, and *unplanned* represent engines that are queuing for the different services, the stock *DoneA, DoneB, DoneC*, and *DoneZ* describe serviced assets. The servicing rates (*ServiceA, ServiceB, ServiceC, ServiceZ*) are influenced by the skill and the number of technicians.

Within the engine layer, an engine can be in the *maintenance* state (*A, B, C and unplanned*) or capable for a mission (*CanWork*). Since the engines in PSS are not necessary to be newly made, it requires different maintenance schedule. A mission-

capable engine is either flying or waiting for the flight hours. Since an aircraft is operating on more than one engine, it can still fly even if an engine stops working. An engine can stop working due to external dynamic behaviour, which is captured by the rate of occurrence (*RandFail*).

With regards to the performance measures, several calculations of availability have been reported in literature, for example, inherent, achieved and operational availability (Kececioglu, 1995). This study applies the calculation from Xerox's DocuCare contract (Xerox, 2010c), defined as having the agreed number of assets ready to fly at any time. The penalty is based on the turnaround time required by an airline customer. In other words, if the aircraft is supposed to have an A-check, and the maximum allowable downtime is 2 hours, a minute delay from the schedule leads to a penalty.

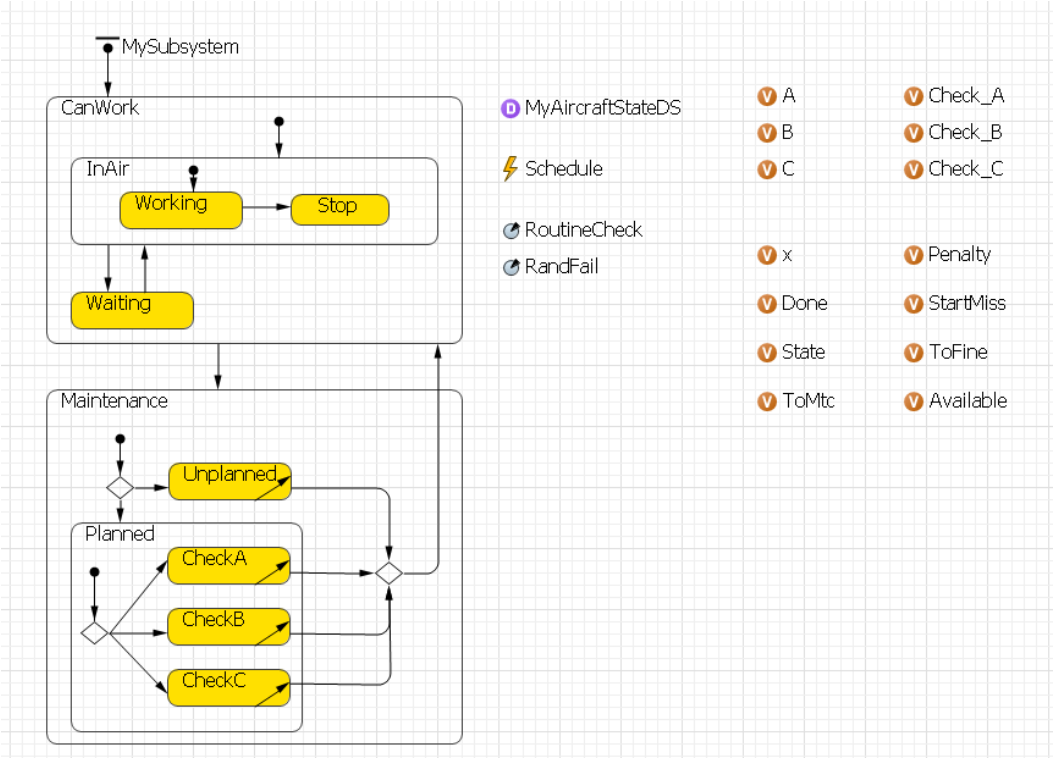


Figure D-3: Subsystem agent

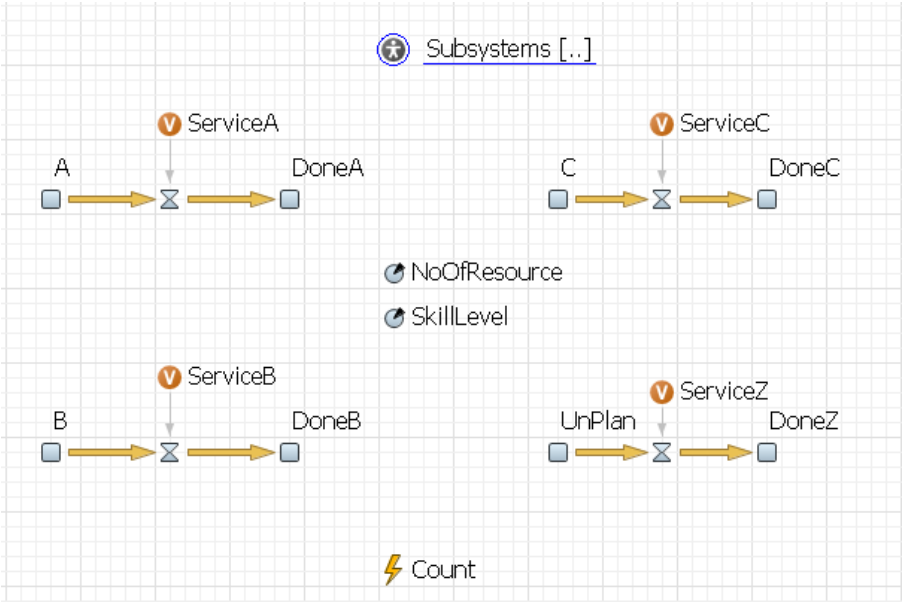


Figure D-4: OEM/Main model

Model code

1. **Subsystem agent**

Schedule

Action:

// The service schedule is notified.

ToMtc=true;

A=1;

B++;

C++;

// Monitor if C-check is due.

if (C==12*A){

Check_C = true;

} else {

```
            Check_C = false;
        }
// If not C-check, monitor whether B-check is due.
        if ((C<12*A)&&(B==2*A)) {
            Check_B = true;;
        } else {
            Check_B = false;
        }
// if C- Check and B-Check are not due, A-check is performed.
        if ((Check_C== false)&&(Check_B==false)) {
            Check_A = true;
        }
//Reset the variables.
        if (B>2) {
            B=0;
        }
        if (C>12) {
            C=0;
        }
```

CanWork

Entry action:

```
//Report the engine status.
        State=1;
```

MyAircraftStateDS.add(time(), CanWork);

Maintenance

Entry action:

State=2;

Exit action

Check_A=false;

Check_B=false;

Check_C=false;

The transition to *Maintenance* is triggered by the condition 'ToMtc==true'.

The transition back to *CanWork* is set as default with the following action:

//Reset status and update penalty.

```
ToMtc=false;
if (ToFine==true){
        Penalty=time()-StartMiss;
        get_Main().TotalPenalty=get_Main().TotalPenalty+Penalty;
        get_Main().NoOfMiss++;
}else{
        Penalty=0;
}
ToFine=false;
```

The transition to *Stop* is a function of *RandFail* rate, and the action code is

//Reset the status.

```
ToMtc=true;
```

x++;

The transition to *Unplanned* is based on the condition x>0, and the action code is

//Reset the status.

x--;

The transition to *CheckA* is based on the condition *Check_A==true*, similar to other checks. Their inner-state transitions are triggered by the agreed turnaround time, and the subsequent action is

//Activate the penalty counts.

StartMiss=time();

ToFine=true;

The exit transition from *CheckA* follows the conditions 'get_Main().DoneA>=1', and the subsequent action is

//Update the stock variable.

get_Main().DoneA--;

The other transitions of the maintenance's sub-states apply the same structure.

2. **Main model**

Subsystem statistics:

Name: SubsystemsStat1

Condition: item.MySubsystem.isStateActive(Subsystem.CheckA)

Name: SubsystemsStat2

Condition: item.MySubsystem.isStateActive(Subsystem.CheckB)

Name: SubsystemsStat3

Condition: item.MySubsystem.isStateActive(Subsystem.CheckC)

Name: SubsystemsStat4

Condition: item.MySubsystem.isStateActive(Subsystem.Unplanned)

Count

Action:

```
// Count the number of the assets currently undergone the 1st-level maintenance.
        A= Subsystems.SubsystemsStat1();
//Define the productivity when there is a 'Check-A' job.
        if (A>0){
                ServiceA=0.2*NoOfResource*SkillLevel;
        }else{
                ServiceA=0;
        }
// Count the number of the assets currently undergone the 2nd-level maintenance.
        B= Subsystems.SubsystemsStat2();
//Define the productivity when there is a 'Check-B' job.
        if (B>0){
                ServiceB=0.1*NoOfResource*SkillLevel;
        }else{
                ServiceB=0;
        }
// Count the number of the assets currently undergone the 3rd-level maintenance.
        C= Subsystems.SubsystemsStat3();
//Define the productivity when there is a 'Check-B' job.
        if (C>0){
                ServiceC=0.05*NoOfResource*SkillLevel;
```

```
        }else{
                ServiceC=0;
        }
// Count the number of the assets currently undergone the unplanned maintenance.
        UnPlan= Subsystems.SubsystemsStat4();
//Define the productivity when there is an 'unplanned' job.
        if (UnPlan>0){
                ServiceZ=0.75*NoOfResource*SkillLevel;
        }else{
                ServiceZ=0;
        }
//Update the availability if the assets can work.
        int availableCount = 0;
        for( int i=0; i<Subsystems.size(); i++ ) {
                if( Subsystems.get(i).State==1 ) {
                        availableCount++;
                }
        }
        if (time()>0){
                Availability = (Availability*(time()-1) +(100.0 * availableCount /
                Subsystems.size()))/time();
        }
//Update the monthly revenue.
        for (int i=0; i<100;i++){
```

```
            if (time()==i*720){
                Revenue=Revenue+100;
            }
        }
//Update the penalty.
        for( int i=0; i<Subsystems.size(); i++ ) {
            if( Subsystems.get(i).State==2 ) {
                ToOEM++;
                AveMissHrs=TotalPenalty/ToOEM;}}
```

Experimentation

To demonstrate the use of the model, an experiment was conducted and the simulation results could be interpreted as explained in Figure D-5.

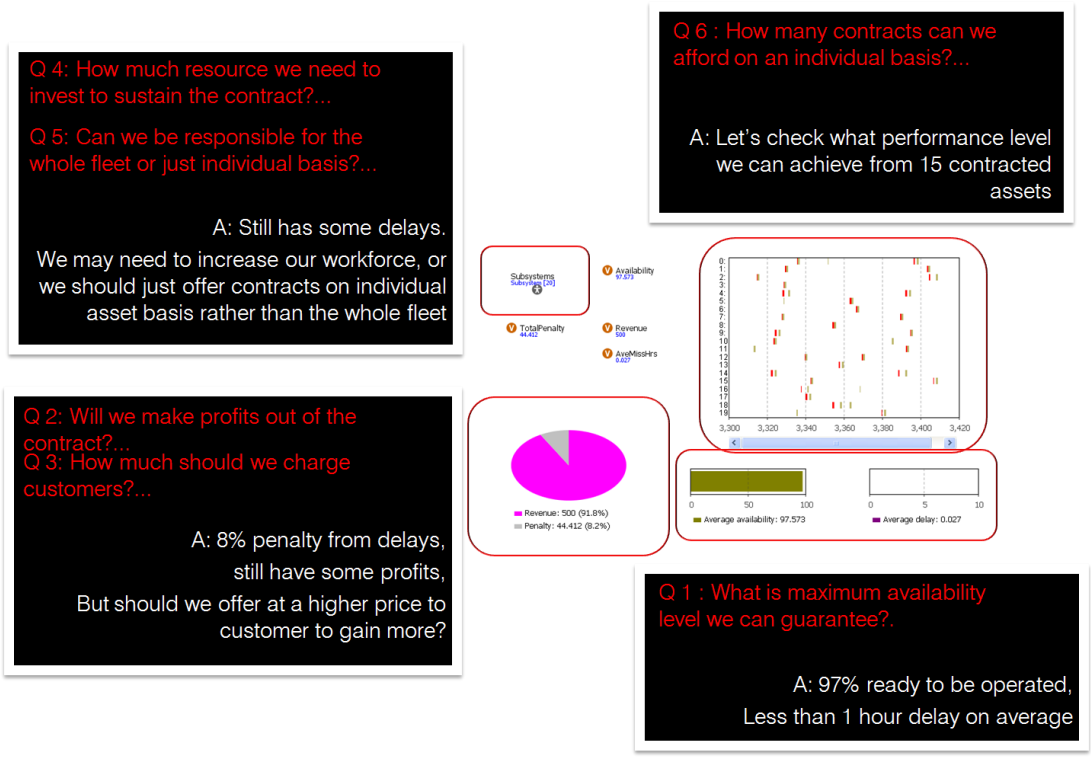


Figure D-5: Implications

E. Hybrid DES-ABS aircraft model

Performance measure

With regards to the measure calculation, the following notations have been applied.

n_s = The number of aircrafts under the tradition businesses

n_p = The number of aircrafts under the availability-type contracts

n_a = The total number of airline operators

n_e = The number of subsystems in each aircraft

n_m = The number of demands for maintenance

t = Current time

First, the measures for the tradition business were inserted. From an airline perspective, the income is a function of sold tickets driven by the number of flights. The model therefore estimates the income as:

$$\text{Income} = \text{Average Profit} * \left(\sum_{i=1}^{n_s} \text{Number of flight}\right) \quad (1)$$

where

n_s is the number of aircrafts under traditional airline business model.

The airline may encounter penalty on an hourly basis, as a function of *Charge,* in case of flight delays. Though denoted as Charge, the penalty is postulated in the form of the airline's reputation lost from each delay and customer's dissatisfaction.

$$\text{Total penalty} = \sum_{i=1}^{n_s} \text{Penalty} \quad (2)$$

$$\text{Penalty}_t = \text{Penalty}_{t-1} + \text{Charge} \quad (3)$$

Once an engine reaches the end of its life, the operator needs to pay the OEM for new engine acquisition. On top of the acquisition cost, the operator needs to hold spare parts for maintenance activities. Consequently,

$$\text{Total cost} = (\text{Inventory} * \text{Holding cost per item}) + \sum_{j=1}^{n_s} \sum_{i=1}^{n_e} \text{Price} \quad (4)$$

Availability is derived as a percentage of mission-capable aircrafts in relation to the fleet size at a particular time, while the average availability is an accumulated value. Thus,

$$\text{Availability} = 100 * \frac{\text{AvailCount}}{\text{Fleet size}} \quad (5)$$

$$\text{Average availability} = \frac{\left(\text{Availability}_{t-1} * (t-1)\right) + \text{Availability}_t}{t} \quad (6)$$

The demand satisfaction rate refers to a percentage of jobs finished within the agreed turnaround time. In case of a delay, the *NoOfMiss* is incremented and the *MissedHrs* is updated on an hourly basis.

$$\text{Demand satisfaction rate} = 100 * \left(\frac{n_m - \text{Number of Missed}}{n_m}\right) \quad (7)$$

$$\text{Average missed hours} = \sum_{i=1}^{n_s} \frac{\text{Missed Hours}}{n_m} \quad (8)$$

Next, the measures for the contracting scenario were defined. The majority are identical to the traditional scenario. Still, the operators now shifts the burden of inventory holding cost to the OEM and only pays for the contract price.

$$\text{Contract price} = \sum_{i=1}^{n_p} \text{ContPrice} \quad (9)$$

From the OEM perspective, the revenue is accumulated from the contract paid by the airline companies. Hence,

$$\text{Revenue} = \sum_{j=1}^{n_a} \sum_{i=1}^{n_p} \text{ContPrice} \quad (10)$$

Similarly, the total penalty is deducted from all operators. A unit of inventory is updated with subject to the production lead time. This gives the total profit of,

$$\text{Profit} = \sum_{i=1}^{n_a} \text{Revenue} - \sum_{i=1}^{n_a} \text{Total penalty} - (\text{Holding\_Cost\_per\_Item} * \text{Inventory}) \quad (11)$$

The differences between the two contracting scenarios take place in the availability calculation: justified monthly against the whole fleet and against individual turnaround time. Similarly in this case, the penalty compares the actual availability against the require availability.

$$\text{Availability} = 100 * \text{PthAvail} \left(\frac{1}{720 * n_p}\right) \quad (12)$$

Where PthAvail is point availability and can be formulated as:

$$\text{PthAvail} = \sum_{j=1}^{n_p} \sum_{i=1}^{t} x \quad (13)$$

x = 0 if the aircraft is in maintenance, otherwise x=1.

Experiment

This section provides an overview on how the simulation model works and demonstrates some of the key features of the model. The model itself is versatile and consequently numerous experimental cases can be tried out by varying the input parameters (listed in section 4.1). Three experiments have been carefully designed to illustrate how the model can be used to aid decision making in the contracts:

- Experiment 1 aims to explore the implications of the different contracting scenarios (e.g. fleet-based vs aircraft-based) to the OEM;
- Experiment 2 aims to estimate the fleet availability that the OEM could offer to the airline based on a particular engine's MTBF;
- Experiment 3 aims to illustrate the Aircraft's agent adaptive capability in rescheduling maintenance services (maintenance cycle).

For all experiments, the run time is 100,000 simulation unit times, which is equivalent to approximately 11 years contract period. Replications could have been performed by using different random number seeds, but this is not the main scope of the study.

Experiment 1

This experiment aims to compare financial outcomes (in terms of profit) the OEM can obtain from offering different contracting scenarios. Two scenarios were tested. First, the OEM offers a contract of a fleet of three aircrafts with 95% availability level, and second, the OEM offered a contract of three aircrafts with turnaround time of 3 hours, 5 hours, 7 hours, and 5 hours for A, B, C, and unplanned maintenance respectively. In either case, the airline operator pays the contract amount (say $30K per month) and demands 10% penalty charges to the OEM if the required performance is not achieved. The outcomes from the two experiments are illustrated in Figure E-2.

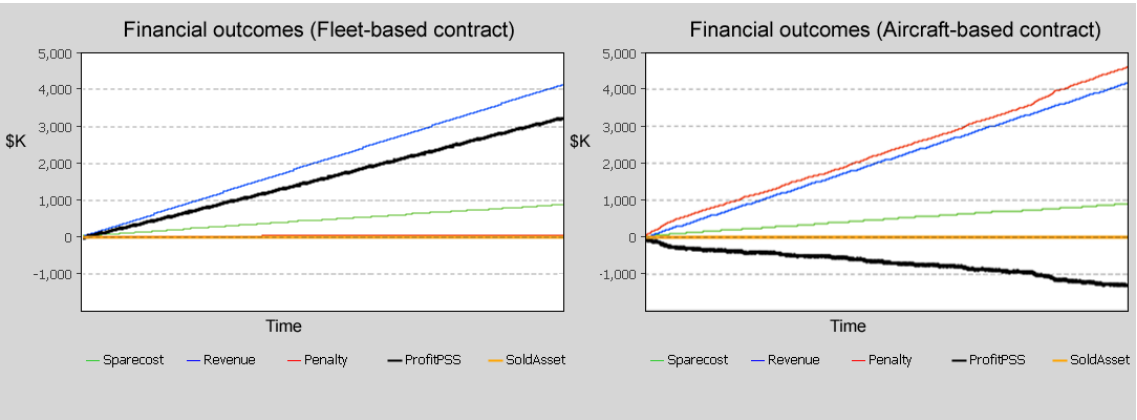
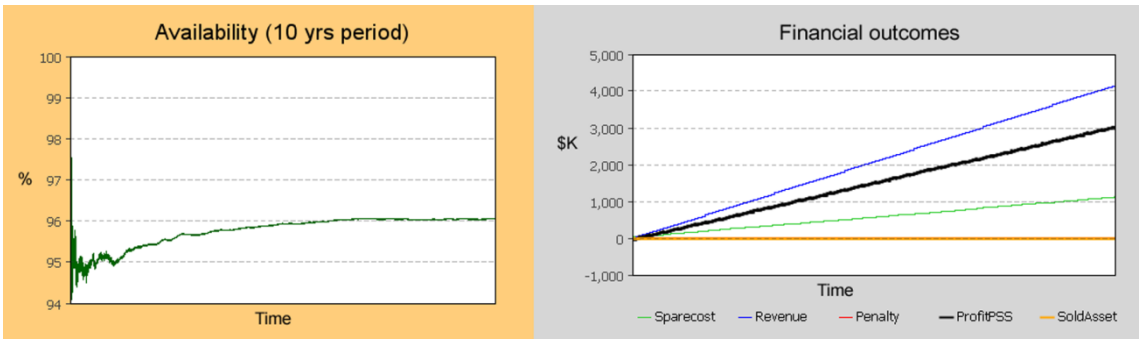


Figure E-1: Results from Experiment 1

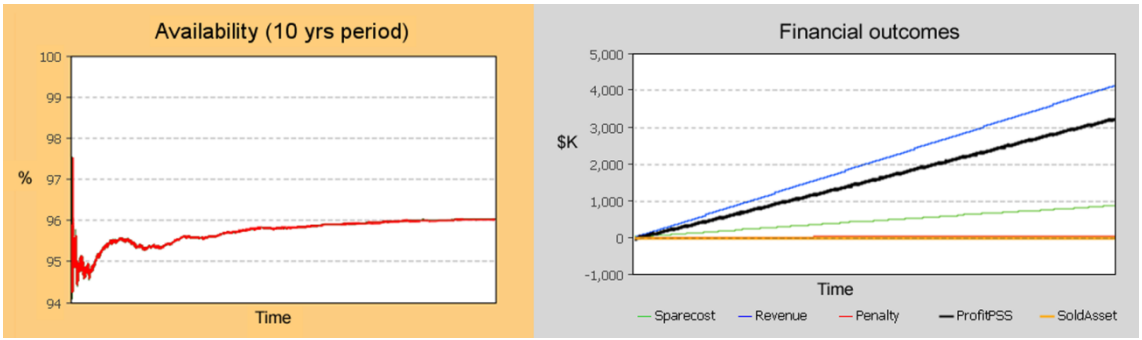
The result demonstrates a substantial difference between the two contract scenarios. The OEM can make profit from the fleet-based contract but will have a loss from the aircraft-based contract. This suggests that the OEM should offer the fleet-based contract instead of the aircraft-based contract.

Experiment 2

This experimentation intends to estimate the availability performance that the OEM can offer based on the engine's MTBF. In this scenario, an airline proposes a required level of fleet availability and the OEM would like to investigate whether or not this can be achieved by the existing engine's specification. Furthermore, the OEM may also investigate, for instance, if an investment should be made in redesigning the engine to extend the engine's MTBF. In this experimentation, only the fleet-based contract (SC2) will be considered, thus, the Aircraft agents in SC1 and SC3 are disabled. The two inputs experimented in the model are the *required availability* (at 95%) and the two ranges of engine's MTBF (1000-1200 and 1200-1500 flying hours). The actual availability and financial performances are shown in Figure E-3.



(a) MTBF between 1000-1200 flying hours



(b) MTBF between 1200-1500 flying hours

Figure E-2: Results from Experiment 2

The results indicate that approximately 96% availability level can be achieved from both MTBF ranges, thus, the OEM can guarantee the 95% fleet availability to the airline. However, there is only a slight increase in profit that can be obtained by

extending the engine's MTBF. In this case, the OEM is not recommended to invest in redesigning engines in order to improve their MTBF.

Experiment 3

This experiment aims to highlight the agent's adaptive capability to monitor the current availability performance of a fleet contract and to adjust the maintenance cycle to optimise availability in the following months. Subsequently, the mechanism summarised in Figure E-4 is enabled inside the Airline agent. Additionally, the Aircraft agents in SC1 and SC3 are disabled.

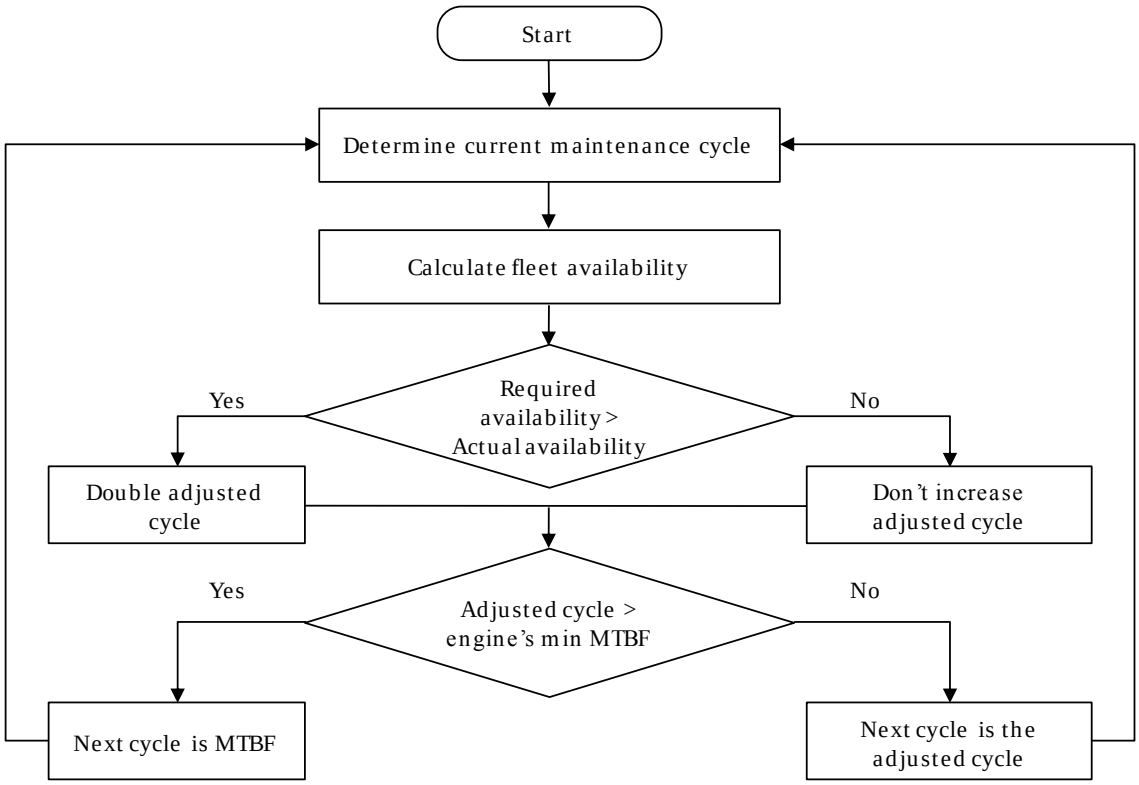


Figure E-3: Self-adaptive maintenance schedule mechanism

Initially, maintenance cycle (*RoutineCheck*) is set to every 200 flying hours. In effect, A, B and C checks take place at every 200, 400, 2400 flying hours respectively. Engine's minimum expected MTBF value is set at 1000, and the availability is required at 95% minimum level. The model is run in equivalent to 100,000 hours. The logic results in the availability performance shown in Figure E-5.

Figure E-4: Results from Experiment 3

It can be seen that availability is improved from approximately 96% to 98% and the maintenance cycle is adjusted to 1000 eventually. This means that the OEM can consider scheduling maintenance approximately in every 1000 flying hours rather than 200 in the contract.

Code

1. **Subsystem**

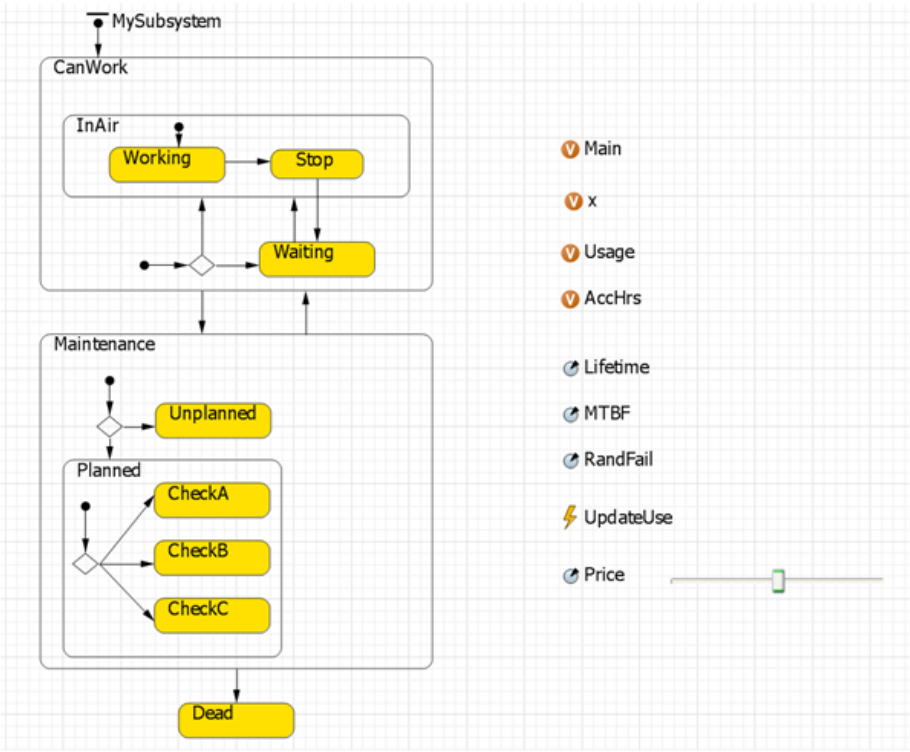


Figure E-5: Subsystem agent

A subsystem can entitle to 3 phases; workable, maintenance, and scrapped. The *CanWork* state includes flying and waiting (either for other systems or itself to be check). During a flight (*InAir*), a subsystem can be operating or broken and the aircraft is flying under redundancy.

In terms of maintenance, the signal is received from the Aircraft agent for a scheduled service, and generated from the random failure rate (*RandFail*) in the case of unplanned maintenance. In the latter case, a *Working* subsystem is transferred to the *Stop* state and immediately in a *Waiting* state until the aircraft lands and undergoes maintenance. Once a maintenance service is required, a Subsystem agent changes to the *Maintenance* state, triggered by the condition:

```
(get_Aircraft().ToMtc==true)&&(get_Aircraft().Gate==0)
```

The *Gate* variable ensures no-repetitive update since each subsystem is ready to work at different time. The variable is incremented when the subsystem is ready to work. Without this variable, the Subsystem agent would be triggered back to maintenance state since the *ToMtc* variable is updated after the whole aircraft is ready to fly.

When the Subsystem agent undergoes maintenance, it sends a signal to the repair station depending on scenario.

```
JobOrder thisOrder = new JobOrder();

thisOrder.From = this;

thisOrder.Family = get_Aircraft().Family;

if (MTBF-Usage<get_Aircraft().RoutineCheck){

        thisOrder.NeedPart=1;

        Usage=0;

}

if (get_Aircraft().Scenario==1) {

        get_Aircraft().get_Airline().MROs.Family3.take(thisOrder);


}else{

        Main.OEMs.Family3.take( thisOrder );}
```

During a service, if the subsystem's condition is unlikely to survive until the next routine check, it will be scrapped. Hence, the transition to the *Dead* state is

Lifetime-AccHrs<get_Aircraft().RoutineCheck

In which case, a new subsystem is replaced, generated by the *Replacement* event in the Aircraft agent under the condition

Subsystems.size()-Subsystems.DeadStat()<ReqSubS

Note that the *size()* method is not used since the dead subsystem is not destroyed from being an agent. Also, the replacement is done immediately based on the assumption that there is always available spares in stock. This is realistic since the lifetime of engine dictates and allows enough time for the OEM to manufacture it.

Since the old engine is out of system, the new engine must update the status as if it was undergone maintenance.

```
add_Subsystems();

Gate++;

Replacement.restart();
```

After checked, the Subsystem agent is waiting for other Subsystem agents to be ready, triggered by the condition:

get_Aircraft().MyAircraft.isStateActive(Maintenance)==true

The agent is triggered to be in air by the condition:

(get_Aircraft().MyAircraft.isStateActive(Maintenance)==false)&&(GateBroke==0);

GateBreak represents the breakdown maintenance. Without this parameter, the agent can resume the *Working* state directly from *Waiting* without being checked. The *GateBreak* parameter is triggered when the Subsystem agent breaks and when the aircraft is recovered.

2. Aircraft

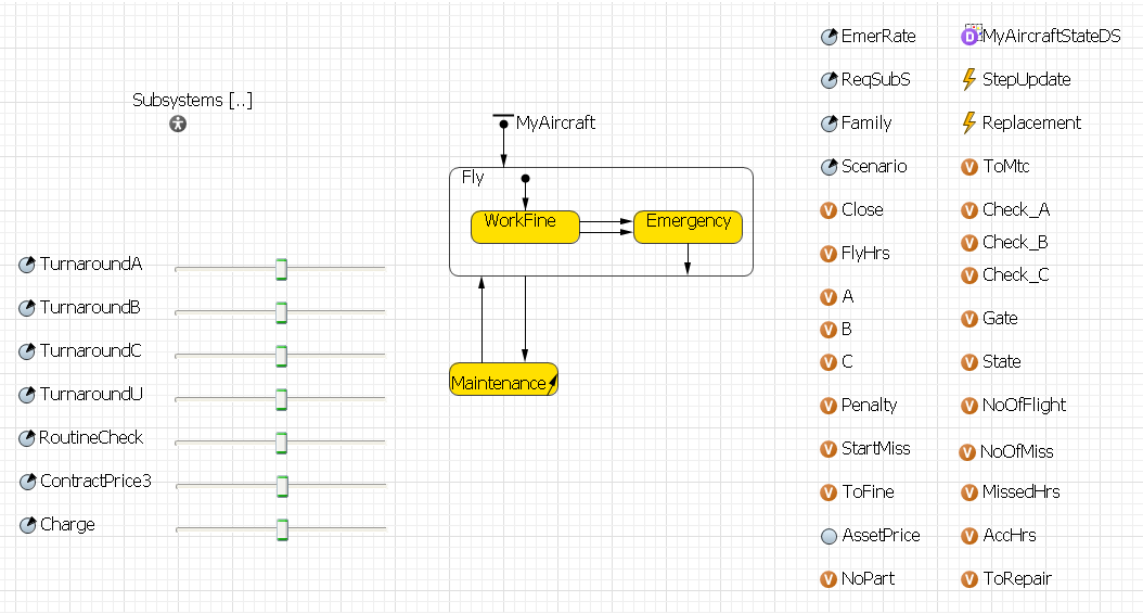


Figure E-6: Aircraft agent

On the aircraft level, an aircraft is assumed to fly if not maintenance. However, an emergency landing is needed if only one engine (i.e. Subsystem agent) is (randomly) broken. Therefore, a transition to the *Emergency* state is

```
(Subsystems.WorkingStat()<=2)&&(Subsystems.RandFailStat()>2)
```

Besides, an external factor (such as the volcanic ash) can lead to emergency landing. This is governed by the *EmerRate*.

In the case of a scheduled maintenance, the Aircraft agent receives a trigger from the *StepUpdate* event as:

```
if (FlyHrs>=RoutineCheck) {

        ToMtc=true;

        A=1;

        B++;

        C++;

        if (C==12*A){

                Check_C = true;
```

```
            } else {
                    Check_C = false;
            }
            if ((C<12*A)&&(B==2*A)) {
                    Check_B = true;;
            } else {
                    Check_B = false;
            }
            if ((Check_C== false)&&(Check_B==false)) {
                    Check_A = true;
            }
            if (B>2) {
                    B=0;
            }
            if (C>12) {
                    C=0;
            }
    }
```

During a maintenance service, the penalty is incremented if the agreed turnaround time in an aircraft-contract is exceeded, dependent on the type of checking. Thus, the timeout transition is

```
(Check_A==true)?TurnaroundA: (Check_B==true)?TurnaroundB:
(Check_C==true)?TurnaroundC: TurnaroundU
```

Once this happens, it activates

```
StartMiss=time();
```

```
ToFine=true;
```

And the S*tep* event updates the status.

```
if (ToFine==true){
        Penalty=Penalty+Charge;
        MissedHrs++;
        if (Close==false){
                NoOfMiss++;
                Close=true;
        }
}
```

Note that the penalty is incremented per hour, therefore, the *Close* variable is added to increment the *NoOfMiss* variable. The *Close* variable is initially set as fault.

The aircraft is recovered if all the subsystems are functional. Hence,

```
(Subsystems.CanWorkStat()==ReqSubS)&&(Gate>0)
```

The aircraft usage is recorded during the flight via the *Step* event.

```
if (MyAircraft.isStateActive(Fly)==true){
   FlyHrs++;
   AccHrs++;
}
```

The *FlyHrs* variable is for scheduling a maintenance service whereas the *AccHrs* is compared against lifetime for the subsystem replacement.

3. Airline

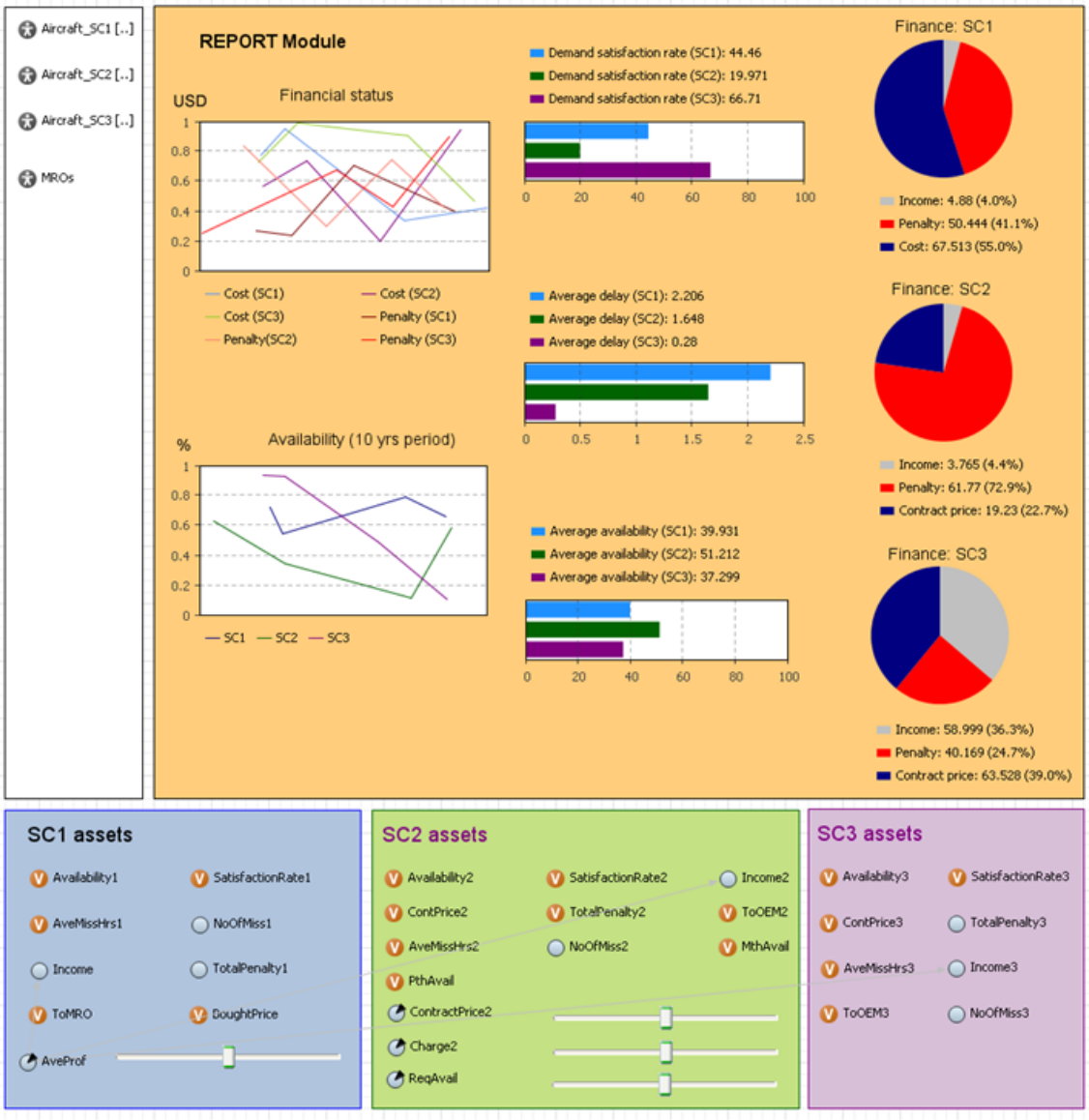


Figure E-7: Airline agent

An airline operator can operate aircrafts in three ways. In scenario 1, the airline buys the aircrafts and carries out maintenance on their own. The second scenario operates under the contract on the fleet basis where monthly fleet uptime is specified and penalty is incurred if this level is not met at the end of the month. The third scenario also operates under contracts but on aircraft basis, where the penalty is incurred if the aircraft is not ready to fly at the end of defined turnaround times. The measures are formed as follows:

Availability

If an aircraft is mission capable, the *AvailableCount* is incremented on an hourly basis. Hence, the average availability up to this point in time is the comparison of this variable and the whole fleet size. This number is recorded in the bar chart and time plot shown in Figure E-7.

```
int availableCount = 0;

for( int i=0; i<Aircraft_SC1.size(); i++ ) {

        if( Aircraft_SC1.get(i).State==1 ) {

                availableCount++;

        }

}

Availability1 = (Availability1*(time()-1) +(100.0 * availableCount1 /
Aircraft_SC1.size()))/time();
```

Note that the time is a denominator, thus the condition *time()>0* is necessary.

Nonetheless, there are two types of availability for the second scenario, one is used for penalty calculation and the other is for consistent comparison between scenarios. Therefore, the monthly availability is calculated additionally as follow:

```
PthAvail=PthAvail+Aircraft_SC2.AvailStat2();

for (int i=1; i<500;i++){

        if (time()==i*720){

                MthAvail=100*PthAvail/(Aircraft_SC2.size()*720);

                if (MthAvail<ReqAvail){

                        TotalPenalty2=TotalPenalty2+Charge2;

                  }

                PthAvail = 0;

        }}
```

The *PthAvail* accumulates the available aircrafts over a month and compares against the whole fleet to get a monthly availability at the end of each month. It is then reset for next month.

Average delay

This measure is unchanged across scenarios. At every hour, the accumulated missed hours of each aircraft is summed up and contrasted against the number of all maintenance jobs.

```
for( int i=0; i<Aircraft_SC3.size(); i++ ) {

        if( Aircraft_SC3.get(i).State==2 ) {

                AveMissHrs3=Aircraft_SC3.MissedHrsStat3()/ToOEM3;

        }

}
```

Demand satisfaction rate

The DSR is also fixed across scenarios. The measure compares the jobs waiting for an available spare with those without waiting.

```
if (Aircraft_SC1.ToRepair1Stat()>0){

        SatisfactionRate1=100*(Aircraft_SC1.ToRepair1Stat()-
        Aircraft_SC1.NoPart1Stat())/Aircraft_SC1.ToRepair1Stat();

}
```

The statistic *ToRepair* comes from the *ToRepair* variable in the Subsystem agents. The *NoPart* statistic was considered against the stock-outs represented by the blocked 'Hold'. Thus, the action at the 'Queue' before the 'Hold' is

```
if (hold14.isBlocked()==true) {

        entity.From.NoPart++;

}
```

Note that each 'Hold' represents different types of checking and different spares.

Pie chart

Each pie chart represents an income, penalty, and cost in each scenario. The income is identical across scenarios: a function of the total fly hours and the average profit on an hourly basis. Hence,

```
AveProf*Aircraft_SC1.FlyHrsStat1()
```

The cost incurred to the airline from SC2 and SC3 are from monthly contract fee, whereas that of SC1 is from the asset price and the spare cost. The spare cost is a function of the total inventory and the cost per unit, whereas the asset price is collected from total subsystems, thus,

```
(MROs.Inventory*get_Main().InvCostPerItem)+(Aircraft_SC1.PriceStat1())
```

The penalty in SC1 and SC3 are calculated from the delayed turnaround time whereas that of SC2 is based on the agreed availability level at the end of each month. Therefore,

The penalty in SC1 and SC3 is collected from the Aircraft agents where

```
if (ToFine==true){

        Penalty=Penalty+Charge;

}
```

4. MRO

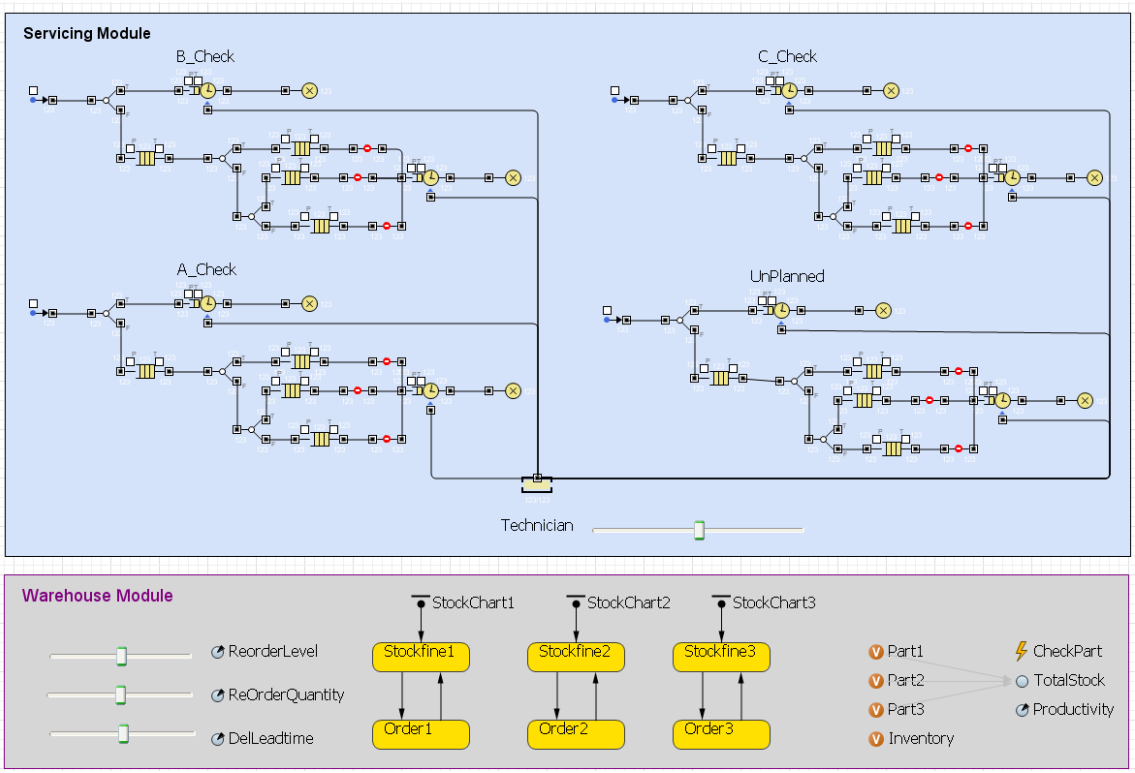


Figure E-8: Airline's MRO agent

The airline's MRO deals with maintenance services and stocking activities. The *JobOrder* Java object is recorded in all DES elements as the entity. The first 'selected output' segregates the job that required a part replacement from simple checks, dictated by the *NeedPart* property of the Java object.

If the job requires a part replacement, the code is required to update the stock as follows:

```
if (entity.Family==1){

        Part1--;

}

if (entity.Family==2){

        Part2--;

}
```

```
if (entity.Family==3){
        Part3--;
}
```

The second and the third 'Selected outputs' address different types of spare parts. The 'Holds' are blocked if there is no available part in the stock, controlled by the *CheckPart* event as:

```
if (Part1>=1) {
        hold3.setBlocked(false);
        hold12.setBlocked(false);
        hold15.setBlocked(false);
        hold18.setBlocked(false);
} else {
        hold3.setBlocked(true);
        hold12.setBlocked(true);
        hold15.setBlocked(true);
        hold18.setBlocked(true);
}
```

In which case, the DSR is updated at the 'Queue' before the 'Hold' via the code

```
if (Part1<1){
        entity.From.NoPart++;
}
```

Similarly, the total part replacements are recorded at the 'Selected outputs'.

```
entity.From.ToRepair++;
```

The stocking decision is based on the reorder level and quantity. The delivery lead time is taken into account. Note that the airline operator may not buy parts from the OEM, thereby the interactions between the two agents are absent in the model.

When the service finishes, a signal is sent back to the Subsystem agent and triggers the agent's state. Hence, the code at the 'Sinks' is

```
send("Finish!", entity.From);
```

5. Part

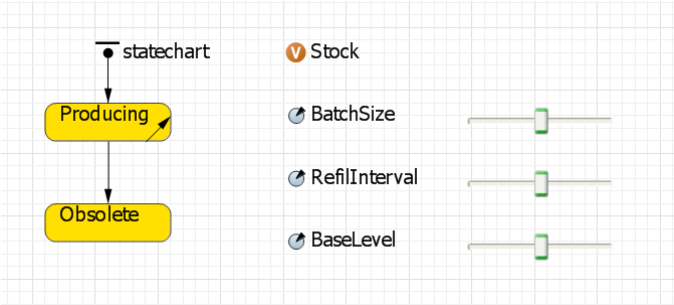


Figure E-9: Part agent

This agent is created for capturing the impact from obsolescence on the OEM's upgrade service. The OEM's stocks are checked regularly as a function of the *RefillInterval*. If the level falls below the *base level*, the order of the *BatchSize* quantity is made. Unlike the MRO, the logistic time is assumed negligible in this model as the parts are also manufactured by the OEM.

6. OEM

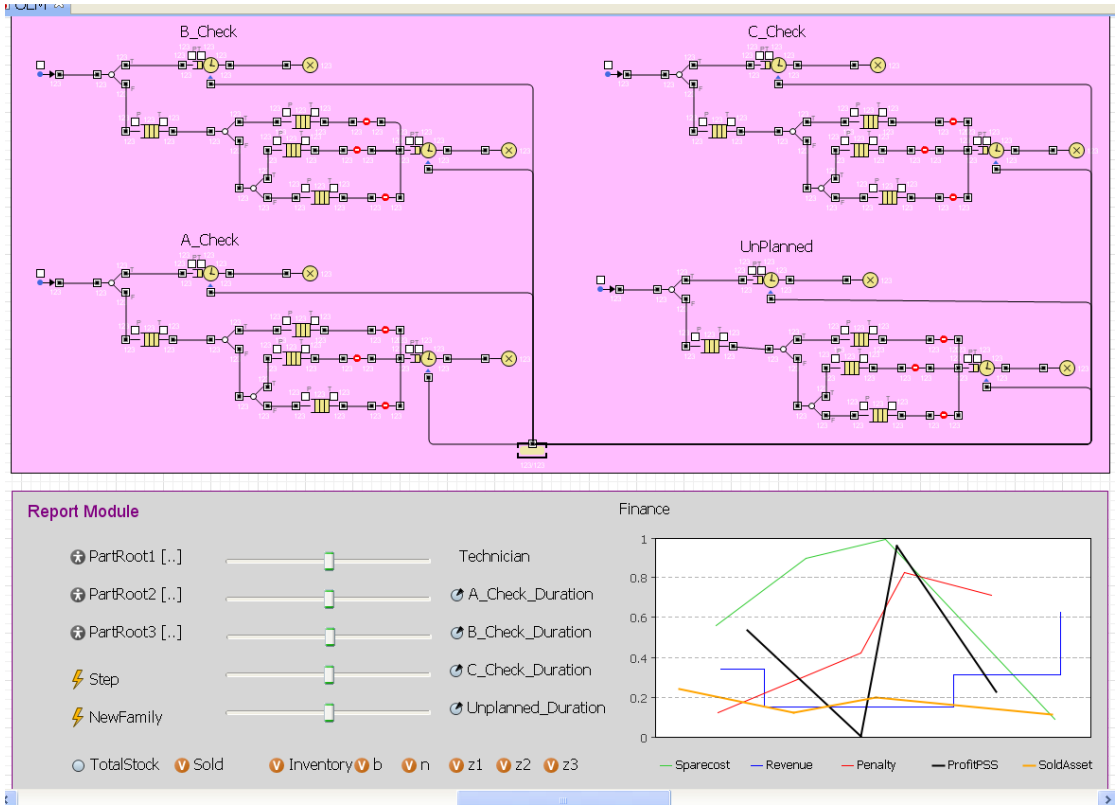


Figure E-10: OEM agent

The OEM structure is similar to MRO, however, the difference are as follows:

- The *NoPart* statistic is checked from the status of the 'Holds' rather than the stock module due to the different hierarchies. Hence,

```
if (hold12.isBlocked()==true) {

        entity.From.NoPart++;

}
```

- Product developments become a part of the model, controlled by the *NewFamily* random event. When this occurs, a message is sent to a particular Part agent to trigger its state to *Obsolete* and a new agent is added to the array.

```
//Randomly generates root family.

        b=uniform_discr(1,3);
```

```
//Stop producing the root family.
        if (b==1){
                send("Dead!", PartRoot1.get((int)z1));
                add_PartRoot1();
                z1++;
        }
        if (b==2){
                send("Dead!", PartRoot2.get((int)z2));
                add_PartRoot2();
                z2++;
        }
        if (b==3){
                send("Dead!", PartRoot3.get((int)z3));
                add_PartRoot3();
                z3++;
        }
```

- The chart illustrates
    - Spare cost = Inventory*get_Main().InvCostPerItem
    - Revenue = get_Main().Airlines.RevenueStat3()+get_Main().Airlines.RevenueStat2()
    - Penalty = get_Main().Airlines.PenaltyStat3()+get_Main().Airlines.PenaltyStat2()
    - Profit = get_Main().Airlines.RevenueStat3()+get_Main().Airlines.RevenueStat2()-get_Main().Airlines.PenaltyStat3()-get_Main().Airlines.PenaltyStat2()-(Inventory*get_Main().InvCostPerItem)

- Soldasset = get_Main().Airlines.SoldStat();

The *SoldStat* statistic is created from the Subsystem agent's *Price* parameter.

F. Hybrid DES-ABS photocopier model code

1. Information Java object

Request From; //To keep the track of a job, enable the queuing rule, and return the message back to the right Subsystem (or Component) agent.

double DueIn; // To prioritise the urgent job.

double Enter; // To record the time that the job enters the OEM's system.

boolean NeedEng; // to separate the job that can be solved on phone from those require a site visit.

boolean NeedPart; // To classify the jobs that require a part replacement from other jobs.

int Family; // To indicate the right stock.

double CoX; // To record the site of customer.

double CoY; // To record the site of customer.

int No; // To allocate part to the right part.

2. Request

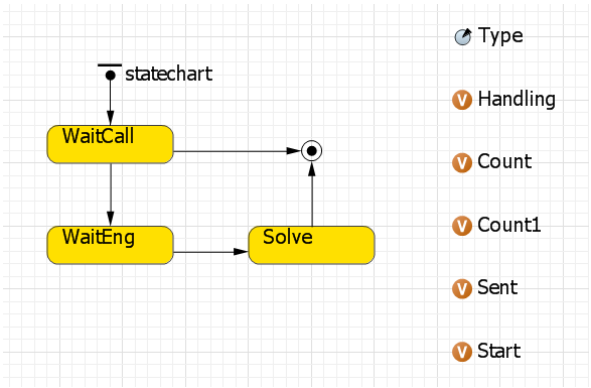


Figure F-1: Request agent

There are four types of requests:

1 = A general question from a customer that can be solved on phone, thus, *NeedEng*=0 and *NeedPart*=0.

2 = A general question from a customer that requires a site visit, thus, *NeedEng*=1 and *NeedPart*=0.

3 = A misuse sent by a photocopier, solvable on phone, thus, *NeedEng*=0 and *NeedPart*=0.

4 = A component replacement, thus, *NeedEng*=1 and *NeedPart*=1;

Upon the agent creation, an order is sent to OEM agent.

```
Start=time();

Order thisOrder = new Order();

thisOrder.From = this;

thisOrder.Enter = time();

if (Type==1){

        thisOrder.DueIn=get_Customer().TargEng;

        thisOrder.CoX=get_Customer().CoX;

        thisOrder.CoY=get_Customer().CoY;

        thisOrder.NeedEng=false;

        thisOrder.NeedPart=false;

        get_Customer().get_Main().enter.take( thisOrder );

}else if (Type==2){

        thisOrder.DueIn=get_Customer().TargEng;

        thisOrder.CoX=get_Customer().CoX;

        thisOrder.CoY=get_Customer().CoY;

        thisOrder.NeedEng=true;

        thisOrder.NeedPart=false;

        get_Customer().get_Main().enter.take( thisOrder );
```

```
}else if (Type==3){
        thisOrder.DueIn=get_Photocopier().get_Customer().TargEng;
        thisOrder.CoX=get_Photocopier().get_Customer().CoX;
        thisOrder.CoY=get_Photocopier().get_Customer().CoY;
        thisOrder.NeedEng=false;
        thisOrder.NeedPart=false;
        get_Photocopier().get_Customer().get_Main().enter.take( thisOrder );
}else if (Type==4){
        thisOrder.NeedEng=true;
        thisOrder.NeedPart=true;
        thisOrder.DueIn=get_Component().get_Photocopier().
        get_Customer().TargEng;
        thisOrder.CoX=get_Component().get_Photocopier().
        get_Customer().CoX;
        thisOrder.CoY=get_Component().get_Photocopier().
        get_Customer().CoY;
        if (get_Component().Standard==false){
                thisOrder.Family=get_Component(). get_Photocopier().Family;
        }else{
                thisOrder.Family=0;
                if (get_Component().getIndex()==1){
                        thisOrder.No=2;
                }else if ((get_Component().getIndex()==0)
                &&(get_Component().Recyclable==true)){
                        thisOrder.No=1;
```

```
                }else{

                        thisOrder.No=0;

                }

        }

        get_Component().get_Photocopier().get_Customer().get_Main().enter.
        take(thisOrder);

}
```

And the record must be incremented on the exit of the *WaitCall* state for contract performance evaluation.

```
//Count the late call response.

        double duration = time()-Start;

        x=duration;

        if (Type<3) {

                if (duration>get_Customer().TargCall){

                        get_Customer().NoOfMiss++;

                }

                get_Customer().ToOEM++;

        }else if (Type==3){

                if (duration>get_Photocopier().get_Customer().TargCall){

                        get_Photocopier().get_Customer().NoOfMiss++;

                }

                get_Photocopier().get_Customer().ToOEM++;

        }else if (Type==4) {

                If (duration>get_Component().get_Photocopier().
                get_Customer().TargCall){
```

```
                get_Component().get_Photocopier().get_Customer().
                NoOfMiss++;

        }

        get_Component().get_Photocopier().get_Customer().ToOEM++;

}
```

Similarly, the job progress must be updated again after signalled by a Technician agent. Therefore, the on exit action of *WaitEng* is

```
//type 2 and 4

    if (Type<3) {

        get_Customer().ToSite++;

        get_Customer().ServTime=get_Customer().ServTime+(time()-Start);

    }else{

        get_Component().get_Photocopier().get_Customer().ToSite++;

        get_Component().get_Photocopier().get_Customer().ServTime=
        get_Component().get_Photocopier().get_Customer().ServTime+(time()-
        Start);

    }
```

After an issue is solved, it triggers the component's state.

```
    if (Type==4){

        get_Component().Replaced=true;

        get_Component().get_Photocopier().get_Customer().x=x;

    }
```

A misuse also triggers the photocopy's state, therefore, it must be removed once handled.

```
// Type 1 and 3
```

```
if (Type==3){
        get_Photocopier().remove_Signal(this);
        get_Photocopier().get_Customer().x=x;
}
```

To set up the communication protocol, the following code was applied to the *on message received* command.

```
statechart.receiveMessage((String)msg);
```

2. **Component**

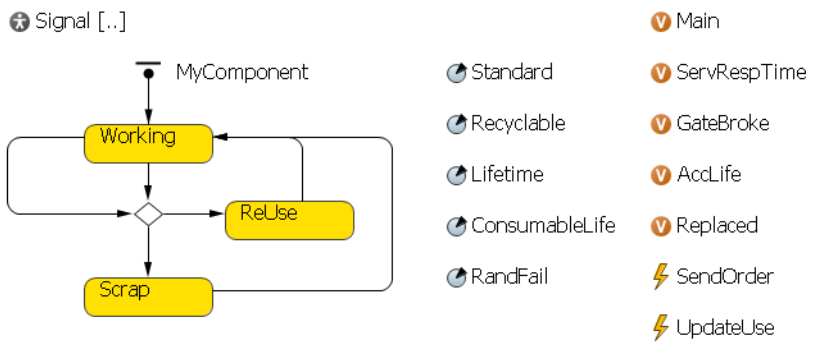


Figure F-2: Component agent

There are several types of components in a photocopier included in the model.

1 = Recyclable parts that are not obsolete, such as cartridge. In this case, *Recycle* =true and *Standard*=true.

2 = Recyclable parts that can be obsolete e.g. a software control chip. Thus, *Recycle*=true and *Standard*=false.

3 = Non-recyclable parts that are not obsolete, such as connectors. Hence, *Recycle*=false and *Standard*=true.

4 = Non-recyclable parts that can be obsolete e.g. motor. Therefore, *Recycle*=false and *Standard*=false.

A component can undergo 2 stages once manufactured: in-use and after-use (once this happens the component can be reused again or scraped, dependent on the level of damage and the raw material). The life of the component is reset at the *Working* state.

```
AccLife=0
```

The variable is incremented via the *UpdateUse* event if the component is in use, via the code:

```
if ((MyComponent.isStateActive(Working))&&
(get_Photocopier().MyPhotocopier.isStateActive(Photocopier.Operating))){

        AccLife++;

}
```

The value of a photocopier's component is not very high, unlike the aircraft context. Accordingly, it is thrown away or recycled once broken. Therefore, there is no presence of the broken state in the model. A service related to a component is often centred around consumable replenishment and cleaning.

A component is disposed due to 2 reasons: a random breakdown (governed by the *RandFail* rate) or the end of life (triggered by the timeout transition of the *Lifetime* value). Upon the disposal, a signal is created (i.e. a request of type 4). A replaced component is treated as the same component that is recovered and ready for an operation. The replacement is controlled by the condition:

```
Replaced==true;
```

Therefore, the variable must be initialised as false, and reset upon the replacement.

If the component is recyclable, there is no waste. Otherwise, the stock must be decremented. A workaround is needed to update the right stock, as follows.

```
if (Standard==false) { //NN

        if (get_Photocopier().Family==1){

                if (getIndex()==0) {

                        Main.StockF1.get(Main.z1).Stock1--;

                }else if (getIndex()==1) {

                        Main.StockF1.get(Main.z1).Stock2--;

                }else{
```

```
                    Main.StockF1.get(Main.z1).Stock3--;
                }
            }else if (get_Photocopier().Family==2){
                if (getIndex()==0) {
                    Main.StockF2.get(Main.z2).Stock1--;
                }else if (getIndex()==1) {
                    Main.StockF2.get(Main.z2).Stock2--;
                }else{
                    Main.StockF2.get(Main.z2).Stock3--;
                }
            }else if (get_Photocopier().Family==3){
                if (getIndex()==0) {
                    Main.StockF3.get(Main.z3).Stock1--;
                }else if (getIndex()==1) {
                    Main.StockF3.get(Main.z3).Stock2--;
                }else{
                    Main.StockF3.get(Main.z3).Stock3--;
                }
            }
        }else{ //SN
            Main.StdParts.get(getIndex()).Stock1--;
        }
        Replaced=false;
```

However, the consumable stocks are monitored weekly if the component is still in use, via the *SendOrder* event. An order is placed for the right stock.

```
if (Lifetime-AccLife<168){
        if (Standard==false) {
            if (get_Photocopier().Family==1){
                if (Recyclable==false){
                    if (getIndex()==1) {
                        Main.StockF1.get(Main.z1).Order3++;
                    }else{
                        Main.StockF1.get(Main.z1).Order1++;
                    }
                }else{
                    if (getIndex()==1) {
                        Main.StockF1.get(Main.z1).Order3++;
                    }else{
                        Main.StockF1.get(Main.z1).Order2++;
                    }
                }
            }else if (get_Photocopier().Family==2){
                if (Recyclable==false){
                    if (getIndex()==1) {
                        Main.StockF2.get(Main.z2).Order3++;
                    }else{
                        Main.StockF2.get(Main.z2).Order1++;
```

```
                }
            }else{
                if (getIndex()==1) {
                    Main.StockF2.get(Main.z2).Order3++;
                }else{
                    Main.StockF2.get(Main.z2).Order2++;
                }
            }
        }else if (get_Photocopier().Family==3){
            if (Recyclable==false){
                if (getIndex()==1) {
                    Main.StockF3.get(Main.z3).Order3++;
                }else{
                    Main.StockF3.get(Main.z3).Order1++;
                }
            }else{
                if (getIndex()==1) {
                    Main.StockF3.get(Main.z3).Order3++;
                }else{
                    Main.StockF3.get(Main.z3).Order2++;
                }
            }
        }
```

```
        }else{
                Main.StdParts.get(getIndex()).Order1++;
        }
    }
```

3. **Photocopier**

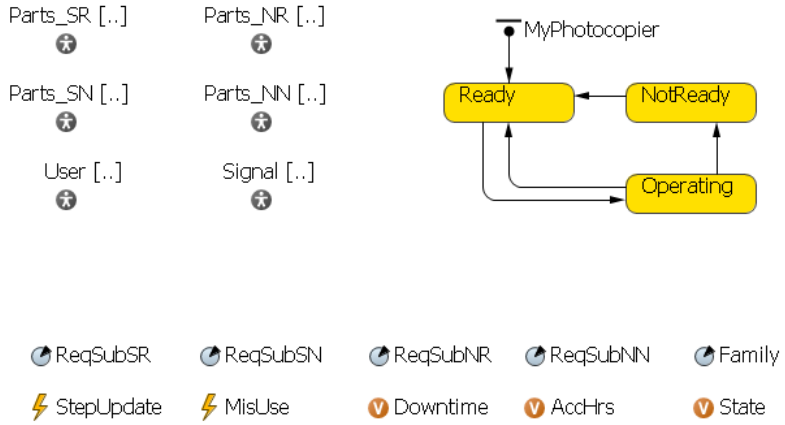


Figure F-3: Photocopier agent

A photocopier contains four types of components as mentioned earlier. A Photocopier agent is operating when a User agent demands for it, hence, the condition of the transition is

```
User.size()>0
```

It finishes the operation as a function of time. Once finished, the User agent is removed and the paper and ink stocks are updated.

```
get_Customer().Paper=get_Customer().Paper-User.get(0).Paper;

get_Customer().Ink=get_Customer().Ink-User.get(0).Ink;

remove_User(User.get(0));
```

The Photocopier agent changes its state from ‘Ready’ and ‘NotReady’ as a result of its Component agent and misuse, governed by the condition

```
Parts_SN.WorkingStat()<ReqSubSN ||
```

```
Parts_SR.WorkingStat()<ReqSubSR ||

Parts_NR.WorkingStat()<ReqSubNR ||

Parts_NN.WorkStat()<ReqSubNN ||

Signal.size()>0
```

If the photocopier is broken during an operation, the user leaves to others. Therefore, on the *NotReady* state:

```
State=1;

get_Customer().Transfer=get_Customer().Transfer+User.size();

for (User u:User){

        u.Leave=true;

}
```

It becomes ready again by the condition

```
(Parts_SN.WorkingStat()==ReqSubSN) &&

(Parts_SR.WorkingStat()==ReqSubSR) &&

(Parts_NR.WorkingStat()==ReqSubNR) &&

(Parts_NN.WorkStat()==ReqSubNN) &&

(Signal.size()==0)
```

The *StepUpdate* event keeps updating the downtime and the asset usage.

4. Customer

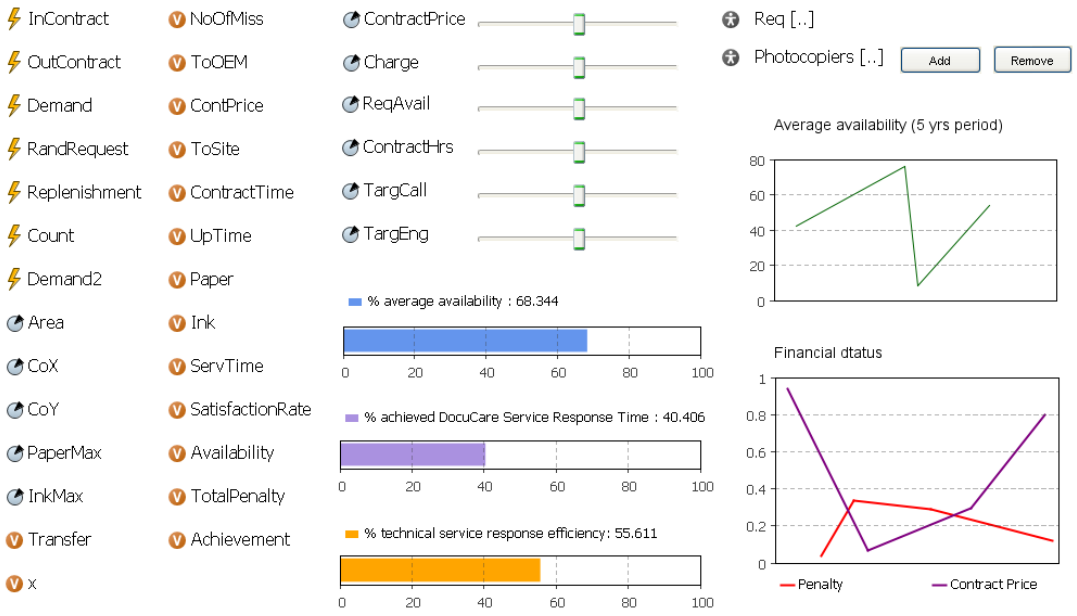


Figure F-4: Customer agent

There are three primary functions within a Customer agent; to enquire the OEM, to generate demands for using photocopiers, and to refill consumables (inks and papers).

The first function is controlled by the *RandRequest* event, occurring randomly. Requests from customer can be divided into 2 groups; those can be solved by call and those require a site visit. Hence,

```
if (ContractTime==true){

        int j;

        double i = random();

        if (i>0.9){

                j=1;

        }else{

                j=2;

        }
```

```
            add_Req(j);

        }
```

The second activity is triggered by *Demand* and *Demand2* events. The *Demand* represents the situation when a photocopier is broken during an operation and the user must go to other photocopiers. Thus, the condition is

```
(Transfer>0)&&(Photocopiers.ReadyStat()>0)
```

Once this happens,

```
for (int r=0; r<Photocopiers.size();r++){

        if (Photocopiers.get(r).State!=1){

                Photocopiers.get(r).add_User();

                Transfer--;

        }

}

Demand.restart();
```

The *Demand2* is a rate function that a photocopy is required by a user (separated by the time of the day). Thus, the condition is

```
((ContractTime==true)&&(Transfer<=0))?1:

((ContractTime==false)&&(Transfer<=0))?0.5:0
```

And it will record the number of users in the system, as

```
int m = (int)triangular(0,Photocopiers.size()-1,0);

if     (Photocopiers.get(m).MyPhotocopier.isStateActive(Photocopier.NotReady)
==false){

        Photocopiers.get(m).add_User();

        Transfer++;

}
```

The contract period is controlled by the events *OutContrat*' and *InContract.*

The last activity is the consumable replenishment, controlled by the *Replenishment,* event which takes place weekly. When this happens, papers and inks are refilled to the levels input by users (denoted as *PaperMax* and *InkMax*).

DocuCare Service Response Time (i.e. DSR(1))

This measure compares the number of calls responded within 1 hour with the total number of calls.

Average technical service response time (by product family)

This measure compares the target time for an engineer arrived at site against the actual performance.

Equipment downtime (i.e. availability)

This is considered against the contract hours.

All these measures are monitored by the *count* event, as follows:

```
if (time()>0){

        if( ContractTime==true) {

                Availability = ((Availability*(time()-1)) +(100.0 *
                (Photocopiers.size()-Photocopiers.DownStat()) /
                Photocopiers.size()))/time();

        }

}

double TotalHrs;

double MthAvail;

for (int i=0; i<500;i++){

        if (time()==i*720){

                ContPrice=ContPrice+ContractPrice;
```

```
MthAvail= 100*((30*ContractHrs*Photocopiers.size())-
(Photocopiers.DowntimeStat()))/(30*ContractHrs*Photocopiers.
size());

UpTime=MthAvail;

if (MthAvail<ReqAvail){

        TotalPenalty=TotalPenalty+Charge;

}

MthAvail = 0;

if (ToOEM>0){

        SatisfactionRate=100*(ToOEM-NoOfMiss)/ToOEM;

        get_Main().AggTime=get_Main().AggTime+
        SatisfactionRate;

        ToOEM=0;

        NoOfMiss=0;

}else{

        SatisfactionRate=100;

        get_Main().AggTime=get_Main().AggTime+
        SatisfactionRate;

}

if (ToSite>0) {

        Achievement=100*TargEng/(ServTime/ToSite);

        get_Main().AggEff=get_Main().AggEff+Achievement;

        ToSite=0;

        ServTime=0;

}
```

```
                    for(Photocopier p: Photocopiers)
                        p.Downtime=0;
                    }
                }
            }
```

Note that all denominators must not be zero, therefore, the initialisation is crucial.

5. Call centre (Main)

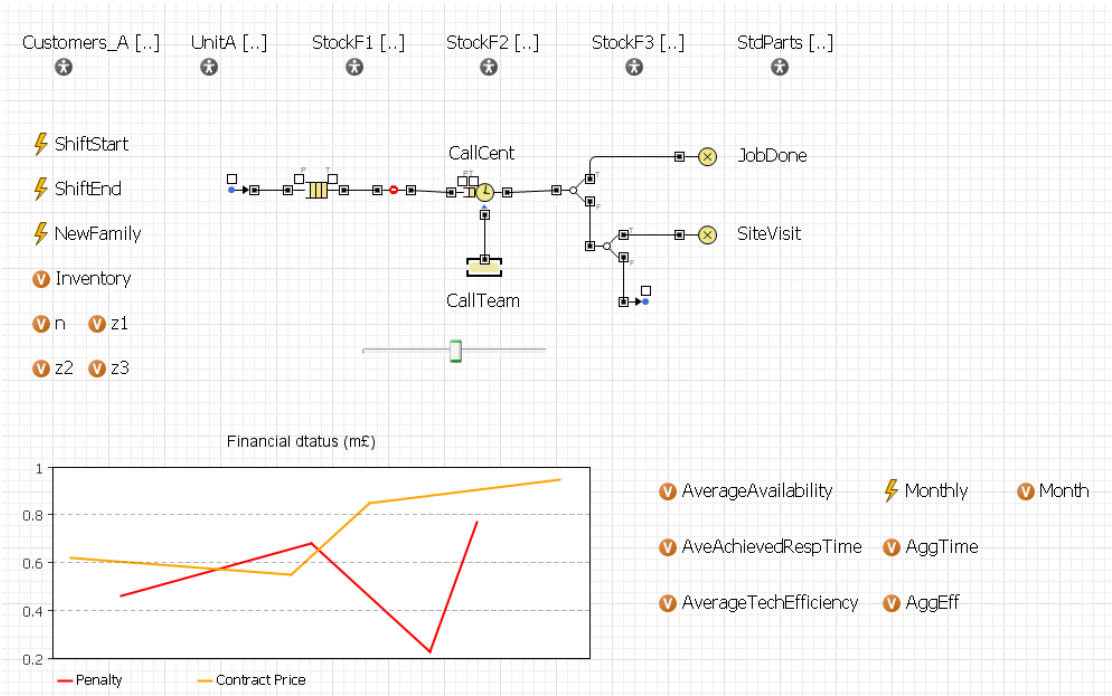


Figure F-5: Call centre agent

The main model illustrates a call centre where all the requests from the Customer agents and signals from the Photocopier and Subsystem agents are initially handled. The first 'Hold' element responds to the staff working hours, controlled by the twp timeout events: *ShiftStart* and *ShiftEnd.*

```
hold.setBlocked(false);
```

If the job can be handled, it is terminated. Otherwise the job is considered whether a part replacement is needed. If so, the job is forward to a particular stock to check the part's availability at the 'Exit'.

```
if (entity.Family==1) {

        StockF1.get(z1).enter.take(entity);

}else if (entity.Family==2) {

        StockF2.get(z2).enter.take(entity);

}else if (entity.Family==3) {

        StockF3.get(z3).enter.take(entity);

}else if (entity.Family==0) {

        StdParts.get(entity.No).enter.take(entity);

}
```

If no part replacement is necessary, the job is allocated to the local business unit, and update the state of the request at the *SiteVisit*.

```
send("Wait!", entity.From);

send(entity, UnitA.get(entity.From.get_Customer().Area));
```

6. **Service unit**

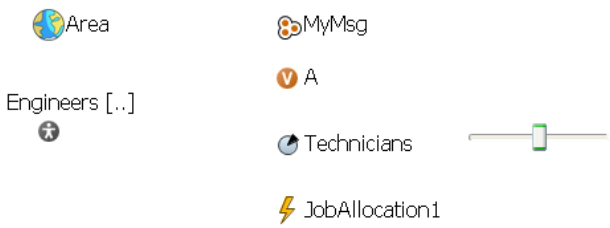


Figure F-6: Service unit agent

A Service unit agent receives the jobs passed from the Call centre agent, therefore, the following code must be inserted upon the message received:

```
MyMsg.add((Order)msg);
```

The message is stored in the *MyMsg* collection of class *Order*. The *JobAllocation* event checks if a technician is available upon receiving a job. Thus, the event is activated based on the condition

```
(A<Engineers.size())&&(Engineers.get(A).statechart.isStateActive(ServEng.Idle))
&&( MyMsg.size()>0 )
```

Note that the first part of the code is to ensure that the simulation engine does not check beyond the technician array. The job is assigned to a technician by the code:

```
int i;

int s;

double m;

double n;

//select the urgent job.

for (s=0; s<MyMsg.size()-1; s++){

    if (time()-MyMsg.get(s).Enter>MyMsg.get(s).DueIn-1){

        send(MyMsg.get(s),Engineers.get(A) );

        Engineers.get(A).moveTo(MyMsg.get(s).CoX,  MyMsg.get(s).CoY
        );

        MyMsg.remove(s);

        A++;

        if (A==Engineers.size()){

            A=0;

        }

        JobAllocation1.restart();
```

```
            }

        }

        s=0;

//If not an urgent job, select the closest job to the technician.

        for (i=0; i<MyMsg.size()-1; i++){

            m= hypot(MyMsg.get(s).CoX-Engineers.get(A).getX(),
            MyMsg.get(s).CoY-Engineers.get(A).getY());

            n =hypot(MyMsg.get(i+1).CoX-Engineers.get(A).getX(),
            MyMsg.get(i+1).CoY-Engineers.get(A).getY());

            if (m>n) {

                s=i+1;

            }

        }

        send(MyMsg.get(s),Engineers.get(A) );

        Engineers.get(A).moveTo(MyMsg.get(s).CoX, MyMsg.get(s).CoY );

        MyMsg.remove(s);

        A++;

        if (A==Engineers.size()){

            A=0;

        }

        JobAllocation1.restart();
```

7. **Technician**

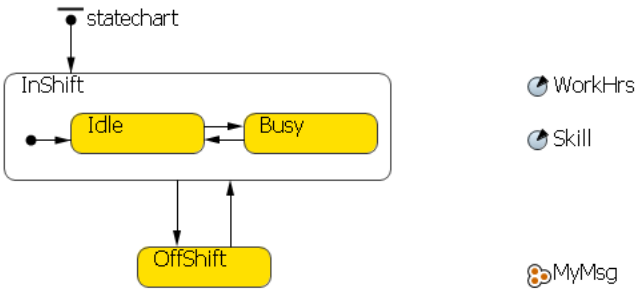


Figure F-7: Technician agent

A Technician agent receives a message from the Service unit agent, and the message must be stored globally. Thus,

```
MyMsg.add((Order)msg);
```

The technician is working according to his shift, triggered by a timeout transition. When he receives a message from the Service unit agent, he becomes busy and updates the job progress in the Request agent.

```
MyMsg.get(0).From.Handling=true;
```

The technician finishes his task at a random time, dependent on his skill. Once this happens, a signal is sent to the Request agent to update its status.

```
send("Solved!", MyMsg.get(0).From);

MyMsg.remove(0);
```

Note that the signal is sent on the exit of the *Busy* state rather than the transition. This is to ensure that the signal is not missing if the off-shift is activated during an operation.

8. **User**

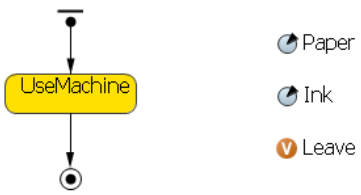


Figure F-8: User agent

A User agent is removed from the Photocopier agent when the job is completed.

```
get_Photocopier().remove_User(this);
```

9. **Part**

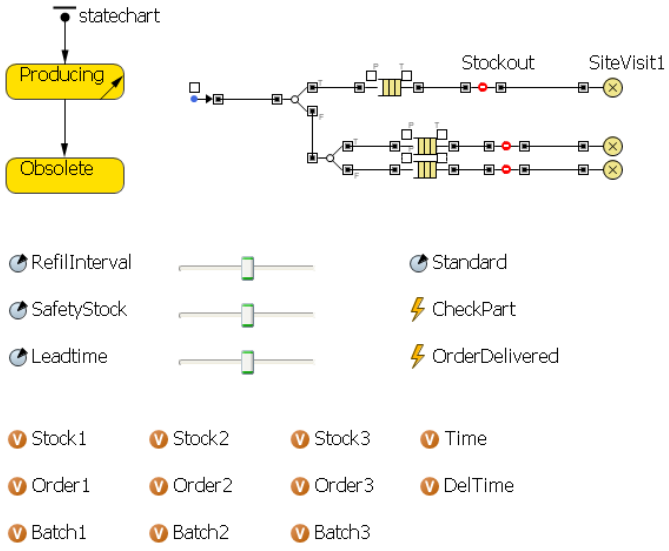


Figure F-9: Part agent

The component health is checked weekly whether a replacement is expected during the week. The information is fed to the *Order* variable within a particular Part agent. Once the agent checks its stock (governed by the *RefillInterval*), the order is sent and reset. Therefore at the transition inside the *Producing* state:

```
Time=time();

double a = Order1+SafetyStock;

if (a>Stock1){

        Batch1 = a - Stock1;

}

if (a>Stock2){

        Batch2 = a - Stock2;
```

```
}

if (a>Stock3){

        Batch3 = a - Stock3;

}

DelTime=exponential(Leadtime);

Order1=0;

Order2=0;

Order3=0;
```

Once the goods are received, the stock is updated. This is controlled by the *OrderDelivered* event with the condition

```
(time()-Time>DelTime)&&(Batch!=0);
```

With the following action code:

```
Stock1=Stock1+Batch1;

Stock2=Stock2+Batch2;

Stock3=Stock3+Batch3;

get_Main().Inventory=get_Main().Inventory+Batch1+Batch2+Batch3;

Batch1=0;

Batch2=0;

Batch3=0;

OrderDelivered.restart();
```

If no part is available, the 'Holds' are blocked by the *CheckPart* event.

```
if ((Standard==true)&&(Stock1<1)){

        Stockout.setBlocked(true);

}else if ((Standard==false)&&(Stock1<1)){
```

```
            Stockout.setBlocked(true);

        }else if ((Standard==false)&&(Stock2<1)){

            hold.setBlocked(true);

        }else if ((Standard==false)&&(Stock3<1)){

            hold1.setBlocked(true);

        }
```

The job is allocated to the local business unit, and the Request agent's state is updated at the 'Sinks'.

```
        send("Wait!", entity.From);

        send (entity, get_Main().UnitA.get(entity.From.get_Component().
        get_Photocopier().get_Customer().Area));
```

G. Hybrid DES-ABS underground model code

1. Information Java object

Subsystem From;

int NeedPart; //To indicate stock

double ManHrs; //To control cycletime

int Site; // To locate service operation

2. Main model

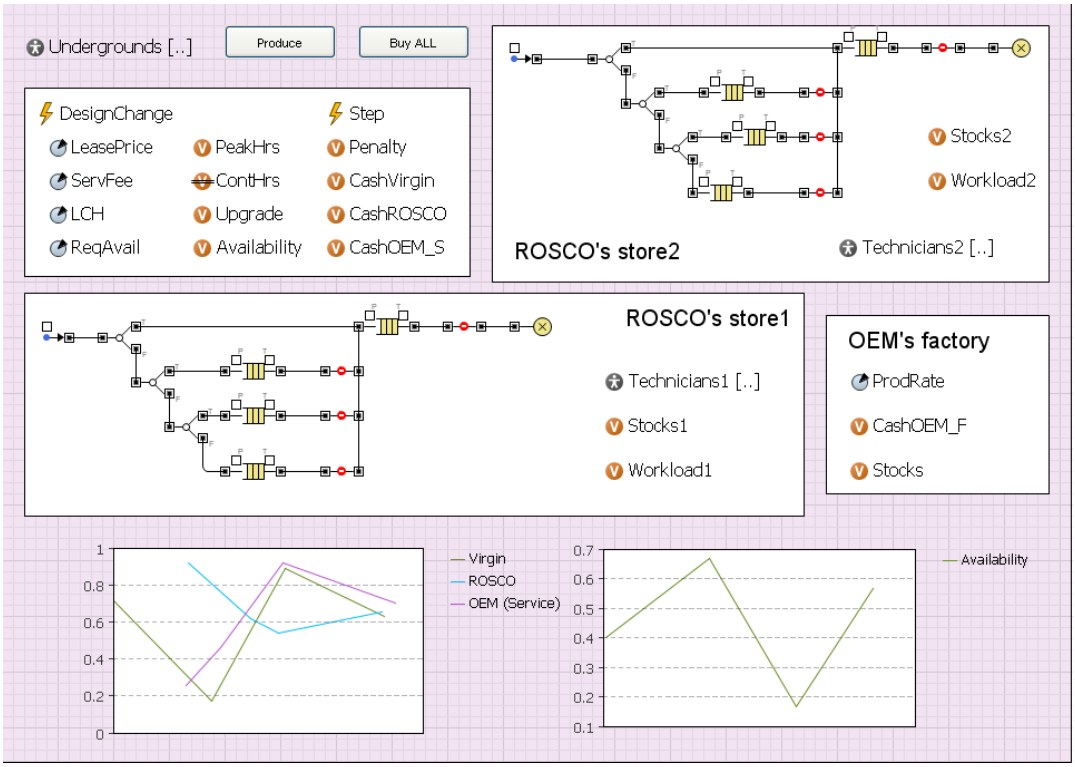


Figure G-1: Main model

Three major parts are delivered from the plant to the depots prior to failures, denoted as *Stock1* and *Stock2*.

These two array variables are of type *double[]* with the initial value 'new double[3]'.

The *Step* event continuously produces parts to the plant's stocks, using the command

```
for( int i=0; i<3; i++ ) {
        Stocks[i] = Stocks[i]+ProdRate[i];
}
```

Services take place at both sides and are separated based on the requited parts. Thus, on the *SelectOutputs*'s exit:

```
entity.NeedPart==10
```

A technician starts servicing if the required part is available in the stock. Otherwise, all 'Hold' elements are activated and lead to queues of jobs. This is also controlled by the *Step* event.

```
if ((queue2.size()>0)&&(Stocks1[0]<1)&&(Stocks[0]>=1)){
        hold2.setBlocked(false);
}else{
        hold2.setBlocked(true);
}
if ((queue3.size()>0)&&(Stocks1[1]<1)&&(Stocks[1]>=1)){
        hold3.setBlocked(false);
}else{
        hold3.setBlocked(true);
}
if ((queue4.size()>0)&&(Stocks1[2]<1)&&(Stocks[2]>=1)){
        hold4.setBlocked(false);
}else{
        hold4.setBlocked(true);
```

```
}
if ((queue6.size()>0)&&(Stocks2[0]<1)&&(Stocks[0]>=1)){
        hold6.setBlocked(false);
}else{
        hold6.setBlocked(true);
}
if ((queue7.size()>0)&&(Stocks2[1]<1)&&(Stocks[1]>=1)){
        hold7.setBlocked(false);
}else{
        hold7.setBlocked(true);
}
if ((queue8.size()>0)&&(Stocks2[2]<1)&&(Stocks[2]>=1)){
        hold8.setBlocked(false);
}else{
        hold8.setBlocked(true);
}
```

If the job can be carried out without a part replacement or the replacement takes place with non-critical parts, the trigger is only based on the technician's availability. This is controlled separately by a Technician agent.

Upgrading can take place randomly after a design change, controlled by the cyclic *DesignChange* event.

```
int i = uniform_discr(0,2);
Stocks[i]=1;
Stocks1[i]=0;
Stocks2[i]=0;
```

```
Upgrade[i]++;
```

Outside the contract hours, the idle technician notifies particular Subsystem agents to be upgraded. This is triggered by the *Step* event.

```
for (Train t:Undergrounds){

for (int n=0; n<3; n++){

        if ((t.Package[n]!=Upgrade[n])&&(Technicians1.Idle1Stat()>0)&&
        (t.InContract==false) &&(t.statechart.isStateActive(Train.InUse))){

                t.Package[n]=Upgrade[n];

                send("Upgrade!", t.Subsystems.get(n));

                }

        }

}
```

As with other cases, the performance monitoring is carried out by the *Step* event

```
for (int k=0; k<500;k++){

        if (time()==k*720){

                CashROSCO=CashROSCO+LeasePrice;

                Availability= 100*((30*ContHrs*Undergrounds.size())-
                (Undergrounds.DowntimeStat()))/
                 (30*ContHrs*Undergrounds.size());

                if (Availability<ReqAvail){

                        Penalty=Penalty+normal(10,LCH);

                }

                CashVirgin=CashVirgin-LeasePrice+Penalty-ServFee;

                CashOEM_S=CashOEM_S+ServFee-Penalty;
```

```
                    Penalty=0;

                    for(Train t:Undergrounds){

                        t.DownTime=0;

                    }

                }

            }
```

3. **Underground**

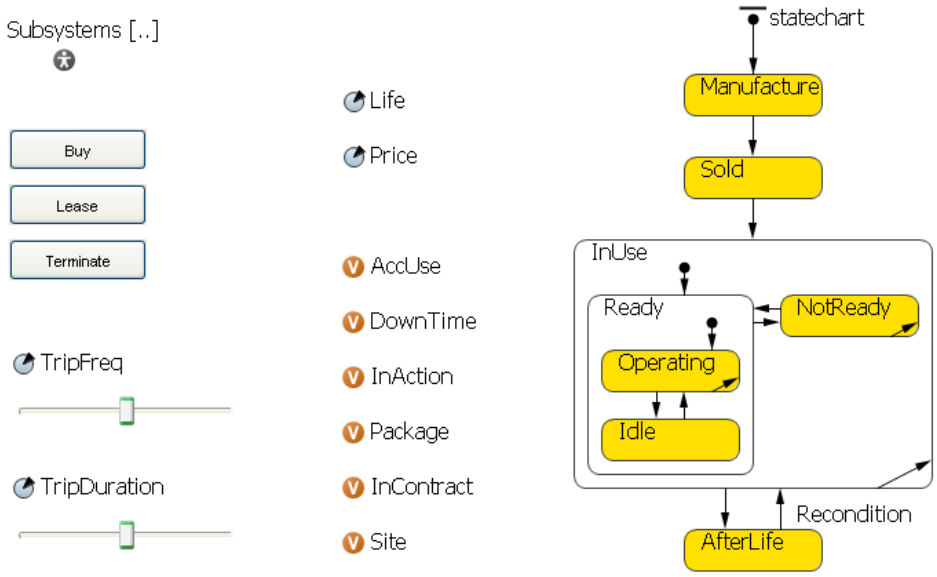


Figure G-2: Underground agent

Once a train is manufactured, it is sold using the manual trigger *Buy*. Thus, the button has the following command:

```
statechart.fireEvent("Buy!");
```

A train is leased in the same way, if it still has a useful life. Once leased, it starts operating from any site.

```
for (Subsystem s: Subsystems) {
        if (s.AccLife < s.Life){
                statechart.fireEvent("Lease!");
        }
}
boolean x=randomTrue(0.5);
if (x==true){
        Site=1;
}else{
        Site=2;
}
```

This also activates all Subsystem agents, triggered by the transition to the *InUse* state.

```
for (Subsystem s: Subsystems) {
                s.statechart.fireEvent("Use!");
}
```

A train can also be manually terminated before it reaches the end of life, via the button that has the command

```
statechart.fireEvent("Dead!");
```

Within the in-service phase, the trains operate on a daily contract-hours basis. This is controlled by the transition inside *InUse*, with the action code

```
for (int i=0; i<20000;i++){
        if ((time()>i*24)&&(time()<=get_Main().ContHrs+(i*24))){
                InContract=true;
        }
```

```
}
```

When the train is *InUs*e, it may be ready for an operation or not. If not, the downtime is recorded and a penalty is incremented in the case of broken during the contract hours. This is monitored by the transition inside *NotReady*, with the action code

```
if (Subsystems.ReadyStat()==Subsystems.size()){

        statechart.fireEvent("Done!");

}

if (Subsystems.ReadyStat()<Subsystems.size()){

//To input availability.

        if (InContract){

                DownTime++;

        }

        for (int i=0; i<20000;i++){

        //Penalise if broken during the peak hours.

                if ((time()>=(i*24)+2)&&(time()<=4+(i*24))){

                get_Main().Penalty=get_Main().Penalty+
                normal(10,get_Main().LCH);

                }

        }

}
```

Although a train is ready for an operation, it may or may not be used. Once operated, the variable *Usage* is updated.

```
if (statechart.isStateActive(Operating)==true){

        AccUse++;

        for (Subsystem s: Subsystems){
```

```
            s.AccLife++;

            }

    }
```

The operations are input by *TripFreq rate* and *TripDuration*.

After terminated, all Subsystem agents are deactivated by the code

```
    for (Subsystem s: Subsystems) {

            s.statechart.fireEvent("Dead!");

    }
```

On the contrary, if the train is retrieved for services, all Subsystem agents are activated.

```
    for (Subsystem s: Subsystems) {

        s.statechart.fireEvent("Use!");

    }
```

4. **Subsystem**

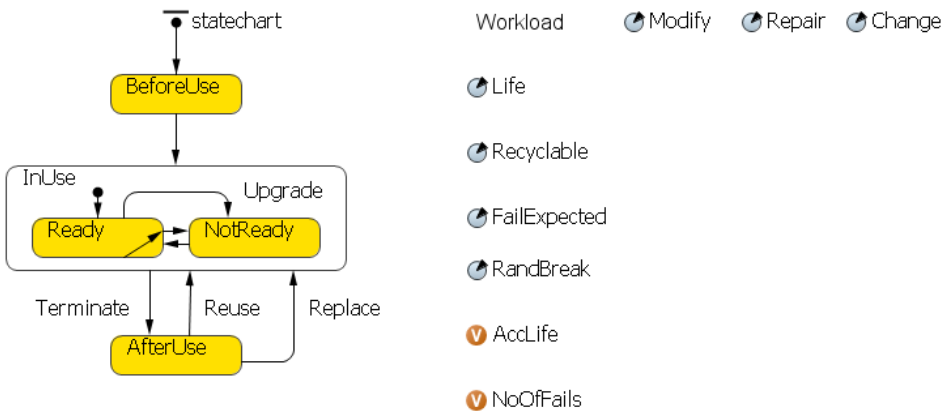


Figure G-3: Subsystem agent

The subsystem's communication was setup using the code:

```
statechart.receiveMessage((String)msg);
```

Within the in-service phase, the transition inside *Ready* keeps monitoring if the subsystem requires a service. Thus, it has the following action code:

```
//Dead --> Update stock, trigger the state, and send a replacement query to the OEM.
    if (AccLife>=normal(10,Life)){
        statechart.fireEvent("Dead!");
        if (Recyclable==true){
            int i = getIndex();
            get_Train().get_Main().Stocks[i]++;
        }
        JobOrder thisOrder = new JobOrder();
        thisOrder.From = this;
        thisOrder.NeedPart = getIndex();
        thisOrder.ManHrs = Change;
        double i=get_Train().getX();
        double j=get_Train().getY();
        if (hypot(40-i, 360-j)< hypot(470-i, 100-j)){
            get_Train().get_Main().enter1.take(thisOrder);
        }else{
            get_Train().get_Main().enter.take(thisOrder);
        }
    }
//Broken --> trigger the state and send a replacement query to the OEM.
```

```
if ((AccLife==NoOfFails*FailExpected)||(randomTrue(RandBreak))){
        statechart.fireEvent("Break!");
        JobOrder thisOrder = new JobOrder();
        thisOrder.From = this;
        thisOrder.NeedPart = 10;
        thisOrder.ManHrs = Repair;
        double i=get_Train().getX();
        double j=get_Train().getY();
        if (hypot(40-i, 360-j)< hypot(470-i, 100-j)){
                get_Train().get_Main().enter1.take(thisOrder);
        }else{
                get_Train().get_Main().enter.take(thisOrder);
        }
}
```

Regarding an upgrade service, the agent receives a trigger from the main model as mentioned earlier. Once this happens, it issues the Java object to the registered site and resets its life using the code:

```
JobOrder thisOrder = new JobOrder();
thisOrder.From = this;
thisOrder.NeedPart = getIndex();
thisOrder.ManHrs = Modify;
if (get_Train().Site==1){
        get_Train().get_Main().enter1.take(thisOrder);
}else{
```

```
        get_Train().get_Main().enter.take(thisOrder);
    }
    AccLife=0;
```

The serviced subsystem becomes ready again after receiving a trigger from a Technician agent.

A Subsystem agent moves to the after-life phase once it receives a trigger from the Train agent, and it can be used again for the same reason or after being replaced and activated by a Technician agent.

5. Technician

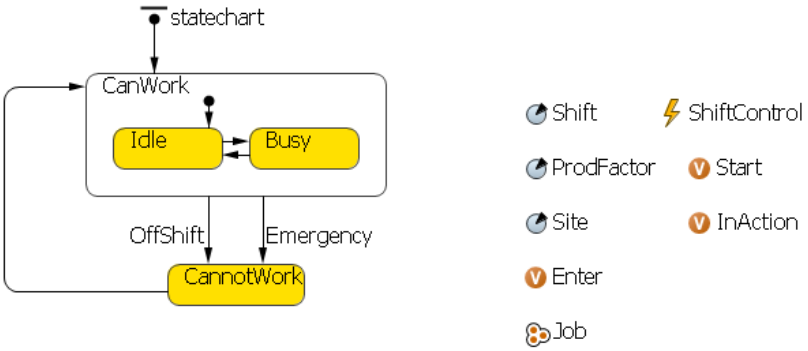


Figure G-4: Technician agent

A technician stores an assigned job in the *Job collection* via the *on message received* code

```
    Job.add((JobOrder)msg);
    InAction=true;
```

This triggers the state to become busy. The service cycle time depends on the type of jobs, thus, the transition is based on a time out i.e.

```
    normal(0.5,Job.get(0).ManHrs)
```

Once the job is finished, the staff approve the subsystem for operational-readiness.

```
send("Ready!", Job.get(0).From);
if (Job.get(0).NeedPart<10){
        if (Site==1){
            get_Main().Stocks1[Job.get(0).NeedPart]--;
        }else{
            get_Main().Stocks2[Job.get(0).NeedPart]--;
        }
}
Job.remove(Job.get(0));
InAction=false;
```

Staff cannot work because of two reasons; an emergency or off-shift. In the case of emergency during an operation, the job is transferred to another technician with the code:

```
if (InAction==true){
        Job.get(0).ManHrs=Job.get(0).ManHrs-(time()-Enter);
        Job.get(0).NeedPart=10;
        if (Site==1){
            get_Main().enter1.take(Job.get(0));
        }else{
            get_Main().enter.take(Job.get(0));
        }
        InAction=false;
        Job.remove(Job.get(0));
}
```

In the case of shift ending during an operation, the staff still need to contibue the job and approve the subsystem for operational-readiness.

```
if (InAction==true){
        send("Ready!", Job.get(0).From);
        if (Job.get(0).NeedPart<10){
            if (Site==1){
                get_Main().Stocks1[Job.get(0).NeedPart]--;
            }else{
                get_Main().Stocks2[Job.get(0).NeedPart]--;
            }
        }
        InAction=false;
        Job.remove(Job.get(0));
}
```

The staff can work again when the shift starts. This is controlled by the *ShiftControl* event with action code

```
Start++;
if (Start==Shift){
    statechart.fireEvent("Start!");
}else{
    statechart.fireEvent("Stop!");
}
if (Start==3){
    Start=0;}
```

H. Hybrid DES-ABS carpet model code

1. Carpet

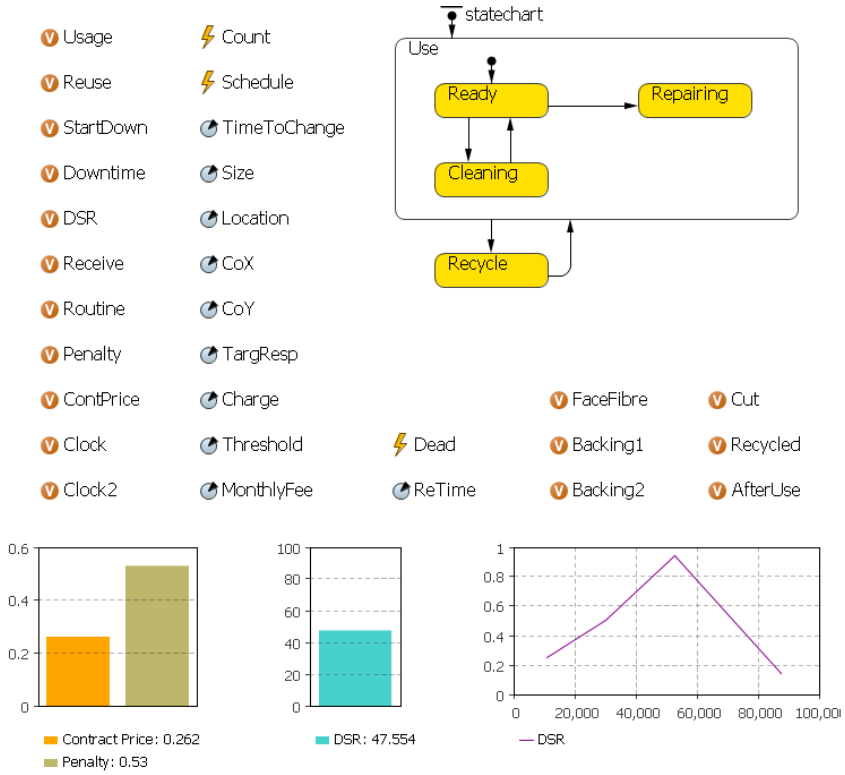


Figure H-1: Carpet agent

After a contract is signed, Carpet agents are added to the model. The cleaning schedule for each customer is arranged with service staff. Therefore, the following code is placed at the state chart entry.

```
Order thisOrder = new Order();

thisOrder.From = this;

thisOrder.Enter = time();

thisOrder.CoX= CoX;

thisOrder.CoY= CoY;

thisOrder.Size=Size;

send(thisOrder,get_Unit());
```

After the schedule is arranged, the information is sent back to the customer. Hence, the *on message received* is

```
Routine=(Routine)msg;

Receive=true;
```

Once this happens, a weekly cleaning schedule is activated, controlled by the *Schedule* event with the following code:

```
if (Receive==true){

        if (time()>=Routine.CleanHrs+(168*Clock)) {

                statechart.fireEvent("Clean!");

        }

}
```

If the carpet is due to be changed, a Staff agent signals the centre upon the cleaning service.

```
if (Usage-time()<168){

        statechart.fireEvent("Change!");

        Cut=Size;

        Usage=Usage+TimeToChange;

}

x++;

Clock++;
```

There are few points to be noted here. Firstly, the *Clock* must be updated to activate the next cleaning schedule independent from how long the staff take to clean the carpet. Secondly, the *Usage* was used rather than *TimeToChange* since the actual time is already progressing ahead of the *TimeToChange* after the 1st recycling. Thirdly, the *Cut* was used rather than *Size,* since only the cut-out part due to random damage is undergone recycling (not the whole carpet). This damage occurs as a rate, and triggers the following action:

```
x1++;

Cut=triangular(1,Size);

statechart.fireEvent("Change!");
```

If not yet the time to change, the staff finish cleaning as function of carpet's size, and the consumable stock is updated as

```
get_Unit().Consumables--;
```

When a recycling is required (either due to a random damage or a scheduled replacement), the order is sent to the main model. Hence, the transition's action is

```
Order thisOrder = new Order();

thisOrder.From = this;

thisOrder.Enter = time();

thisOrder.CoX= CoX;

thisOrder.CoY= CoY;

thisOrder.Size=Cut;

send(thisOrder,get_Unit().get_Main());

StartDown=time();

AfterUse=true;
```

The *AfterUse* triggers the material record of wastes (face fibre and 1st backing) and recycling (2nd backing), controlled by the *Dead* event using the following action code:

```
if (AfterUse==true) {

        Clock2++;

        if (Clock2>=ReTime) {

                Recycled=get_Unit().get_Main().R_Back2*Cut;

                Backing2=Backing2+Recycled;
```

```
                    get_Unit().get_Main().Backing2=get_Unit().get_Main().Backing2
                    + Recycled;

                    FaceFibre=FaceFibre+Cut*get_Unit().get_Main().R_Face;

                    Backing1=Backing1+Cut*get_Unit().get_Main().R_Back1;

                    Clock2=0;

                    AfterUse=false;

            }

        }
```

Note that only the 2nd backing is fed back to the main model, and the *ReTime* parameter allows delays in the collection and recycling cycle time.

2. **Main**

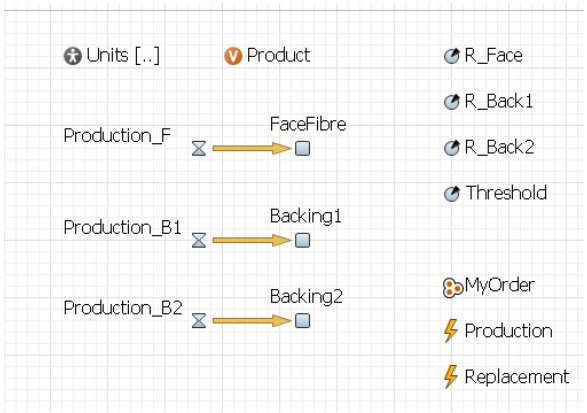


Figure H-2: Main model

This model refers to the central OEM. Unlike other industries, the recycle activity is linked closely with the production and the OEM manufactures a carpet from raw materials. Consequently, the production element is modelled explicitly.

SD was used to model the manufacturing process since a carpet is not a discrete but a continuous entity. Accordingly, it was more natural to use SD.

The main model receives a signal from a Carpet agent upon a recycling, hence, the *on receive message* has the code:

```
MyOrder.add((Order)msg);
```

A unit of a carpet is manufactured from a fixed ratio of face fibre, 1st backing, and 2nd backing. Thus the *Production* event has the condition

```
(FaceFibre>=R_Face)&&(Backing1>=R_Back1)&&(Backing2>=R_Back2)
```

Once produced, the raw material stocks are updated.

```
Product++;

FaceFibre=FaceFibre-R_Face;

Backing1=Backing1-R_Back1;

Backing2=Backing2-R_Back2;

Production.restart();
```

The OEM needs to keep checking their order, triggered by the *Replacement* event.

```
if (MyOrder.size()>0){

int j=0;

//Check if the carpet is enough for an order.

    if (Product>MyOrder.get(j).Size){

        MyOrder.get(j).From.Reuse=true;

        Product=Product-MyOrder.get(j).Size;

        MyOrder.remove(j);

    }

}
```

3. **Service unit**

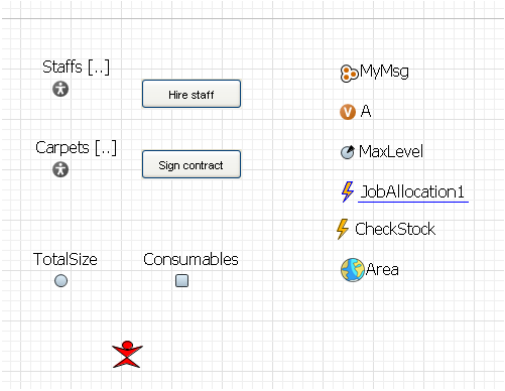


Figure H-3: Service unit agent

The unit allocates the job sent by the customer, hence, the *on message receive*:

```
MyMsg.add((Order)msg);
```

When there is a message and available staff, the *JobAllocation* event is triggered i.e.

```
(A<Staffs.size())&&( MyMsg.size()>0 }
```

The element checks whether there are idle staffs in the reasonable travel distance. The allocation always starts from the 1st staff unless she is no longer free or too far from the customer site. If the increment is made to the final member but nobody is available in the acceptable reach, new staff are hired.

```
//A is free.

    if (40-Staffs.get(A).Workload>MyMsg.get(0).Size/100){

    //A is in range.

        if (hypot(Staffs.get(A).getX()-MyMsg.get(0).CoX, Staffs.get(A).getY()-
        MyMsg.get(0).CoY)<50){

            send(MyMsg.get(0),Staffs.get(A) );

            Staffs.get(A).Workload=Staffs.get(A).Workload+
            MyMsg.get(0).Size/100;

            MyMsg.remove(0);

            A=0;
```

```
        }else{   //A is not in range.

                A++;

                if (A==Staffs.size()){ //None is in range

                        add_Staffs(MyMsg.get(0).CoX, MyMsg.get(0).CoY);

                        send(MyMsg.get(0),Staffs.get(Staffs.size()-1));

                        Staffs.get(Staffs.size()-1).Workload=
                        Staffs.get(Staffs.size()-1).Workload
                        +MyMsg.get(0).Size/100;

                        MyMsg.remove(0);

                A=0;

                }

        }

}else{ //A is not free.

        A++;

//None is free.

        if (A==Staffs.size()){

                add_Staffs(MyMsg.get(0).CoX, MyMsg.get(0).CoY);

                send(MyMsg.get(0),Staffs.get(Staffs.size()-1));

                Staffs.get(Staffs.size()-1).Workload= Staffs.get(Staffs.size()-
                1).Workload+MyMsg.get(0).Size/100;

                MyMsg.remove(0);

                A=0;

        }

}

JobAllocation1.restart();
```

The model allows a staff hiring in the case that the schedule is too tight. Also, the consumables are refilled every week, controlled by the event *CheckStock* even. Once this happens,

Consumables=MaxLevel;

4. **Staff**

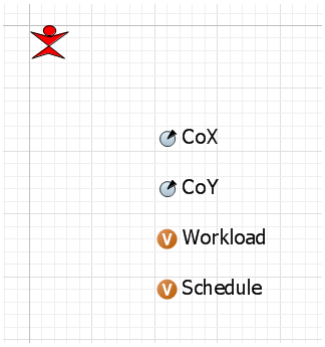


Figure H-4: Staff agent

After a job is allocated by the unit, the status is updated and the information is sent to the customer as mention earlier. Hence, *on the receive* action

```
Schedule=(Order)msg;

Routine thisRoutine = new Routine();

if (Workload<8) {

        thisRoutine.CleanHrs = Workload;

}else if ((Workload>=8)&& (Workload<16)) {

        thisRoutine.CleanHrs = Workload+16;

}else if ((Workload>=16)&& (Workload<24)) {

        thisRoutine.CleanHrs = Workload+32;

}else if ((Workload>=24)&& (Workload<32)) {

        thisRoutine.CleanHrs = Workload+48;

}else{
```

```
            thisRoutine.CleanHrs = Workload+64;

        }

        send(thisRoutine, Schedule.From);
```

Note that the *Routine* Java object was added to the model. On top of that, the logic adopted since the workload refers to working hours whereas the routine is the running time.

I. Case study protocol

Case study protocol

The protocol is developed as part of the PhD research "Simulation modelling of service contracts within the context of Product-Service Systems (PSS)". The first part of this protocol entails a series of open-ended questions, aimed at understanding problem. The second part lists characteristics for the purpose of case customisation.

Part I: General information

Issues		Questions
Contractual level	Design	What are the service offerings in the contract? Who owns what? Who is responsible for what? When does the OEM/service provider interact with customer? What issues are covered in a contract? When is the OEM/service provider penalised? How to decide whether to accept the proposed contract? What is the major concern in delivery the contract?
	Modification	Can existed contracts be renegotiated? If yes, what happen next?
	Asset's operation	Is asset operation defined? If yes, how is it defined? Is the operating condition defined? If yes, how is it defined?
Operational level	Service operations	What are the factors that trigger the activity? How to predict it? How accurate is the prediction? How is the activity carried out? What are decision/factors involved in the activity? How long does the activity take? What circumstances can delay the processes? And how to manage it?
Individual level		How can an individual entity in the system influence overall system behaviour?

Part II: Case characteristics

1. OEM service decision making

A0: OEM has fixed routine in performing services. ☐

A1: Adaptive productivity, fixed routine, global view of situation. ☐

A2: Adaptive productivity, flexible routine, no global view of situation. ☐

2. Subsystem characteristic

B0: The contracted product's state can be predicted on aggregate level. ☐

B1: The contracted unit requires breakdown analysis into subsystem levels, but there is no influence between subsystems. ☐

B2: Subsystem behaviour can influence one another and the contracted unit. ☐

3. Work breakdown structures

C0: Service performance is measured only at the end of all operations. ☐

C1: Jobs are preceded by several department and service performances are measured separately (A1). ☐

C2: Jobs are preceded by several department and service performances are measured separately (A2). ☐

4. Contract creation policy

D1: A new contract is signed based on utilisation. ☐

D2: There is no predefined situation when the OEM should make new contract. ☐

5. Capacity adjustment policy

E0: There is no rule when to adjust capacity (A1). ☐

E1: There is no rule when to adjust capacity (A2). ☐

E2: Capacity is regularly adjusted based on certain rule. ☐

6. Contract termination likelihood

F0: Contract termination is allowed. ☐

F1: Customer can negotiate to end the contract. ☐

7. Relationship protocol

G0: The OEM must accept all the services required by the customer. ☐

G1: The OEM does not need to respond to all services, but it may cost the OEM penalty (in case of A1). ☐

G2: The OEM does not need to respond to all services, but it may cost the OEM penalty (in case of A2). ☐

J. Questionnaires

Pre-testing questionnaire

This questionnaire is part of the PhD research "Simulation modelling of service contracts within the context of Product-Service Systems (PSS)", which aims to evaluate the efficiency and effectiveness of the developed constructs. Along with this questionnaire, a case study description and the constructs will be provided. The participants will be asked to apply the constructs in building a simulation model based on the case. The purpose of the model is to enable evaluations of offering alternatives for the OEMs to design a service contract.

Researcher: Sarocha Phumbua

1. Name:

2. Have you been involved in PSS research?

☐ Yes

☐ No

If yes, how long?

3. How confident are you in using simulation software to perform the task?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

4. Have you used any of the following simulation techniques before?

☐ Spreadsheet

☐ System Dynamic

☐ Discrete-Event Simulation

☐ Agent-Based Simulation

☐ Never

If yes, how long and what software package?

5. Have you followed any guideline/methodology for evaluating alternative PSS offers before?

☐ Yes

☐ No

6. If yes,

Can it be used to capture PSS characteristics effectively?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

How easy is it to be followed?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

How easy is it to be modified if it cannot be used directly?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

How quicker it can shorten the model development?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

How easier it can help to perform the task?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

What is the methodology?

Post-testing questionnaire

1. Name:

2 How confident are you in using simulation software?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

3. How effective can the constructs capture PSS characteristics?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

If no, please state why?

4. Are the constructs clearly presented?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

If not, what can be improved?

5. Can the constructs be used directly or some modifications are required?

☐ Yes, it can be used directly

☐ No, some modifications are required

If no, what modification?

6. How easy are the constructs to be modified?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

If not, what can be improved?

7. How quick can the constructs help to perform the task?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

If not, what can be improved?

8. How easy can the constructs help to perform the task?

☐ 1 Not at all ☐ 2 ☐ 3 ☐ 4 ☐ 5 Very

If not, what can be improved?

9. Will you follow these constructs or recommend to others in the future?

☐ Yes

☐ No

If no, please state why not?

10. Please state your perception on the constructs

K. Audit trail from the pilot sessions

All participants were given the pre-test questionnaire prior to the testing (see Appendix J).

1. Simulation learner

Date: 11/08/2011 – 12/08/2011

Scheduled time: 09.30-15.30, 09.30-14.00

Place: Cranfield University, Building 50, Dr. Benny Tjahjono 's research area

Participants: Anonymous

List of activities (Day 1):

1. Brief the aim of the meeting – to validate efficiency and effectiveness of the constructs.

2. Introduce the software's interfaces, basic operations and shortcuts, how to create a model, Java object, agents, attributes, and variables.

3. The participant gets familiar with the software and creates a model, agents, Java object.

4. Introduce the DES elements used in the constructs from the software and their important input parameters to be defined.

5. The participant builds the high-level OEM agent (no detailed actions).

6. Introduce state modelling and important commands.

7. The participant builds the high-level Asset agent.

8. Introduce different communication methods.

9. The participant completes the Asset agent and runs the model.

10. Lunch

11. Introduce analysis and presentation functions e.g. slider, button, graphs, and associated Java commands e.g. add/remove agents, monitor utilisation.

12. The participant continues developing the model.

List of activities (Day 2):

1. The participant completes the model.

2. Questions and answers.

2. DES expert

Date: 01/09/11, 09/09/11, 01/10/11, 09/10/11

Scheduled time: 12.00-14.00, 12.00-14.00, 15.00-17.00, 13.00-14.00

Place: VDO conference

Participants: Anonymous

List of activities (Day 1):

1. Brief the aim of the meeting.

1. Explain the document briefly – case description, applied Java command, the constructs.

Activity (Day 2): Explain the constructs in details.

Activity (day 3): Complete the basic model.

List of activities (Day 4):

1. Check the selected variants and the participant's model.

2. Questions and answers.

3. Simulation expert

Date: 15/08/2011

Scheduled time: 09.30-13.00

Place: Cranfield University, Building 50, Dr. Benny Tjahjono 's research area

Participants: Anonymous

List of activities

1. Brief the aim of the meeting.

2. Brief the session – the participant was asked to model a case based on constructs.

3. Highlight the role of the participant that he needs to complete the model with no intervention from the author.

4. Explain the constructs.

5. The participant completes the basic model.

L. Example of the constructs' implementation on a software tool

This section aims to prove the concept inside the constructs by implementing it in AnyLogic.

The shared model

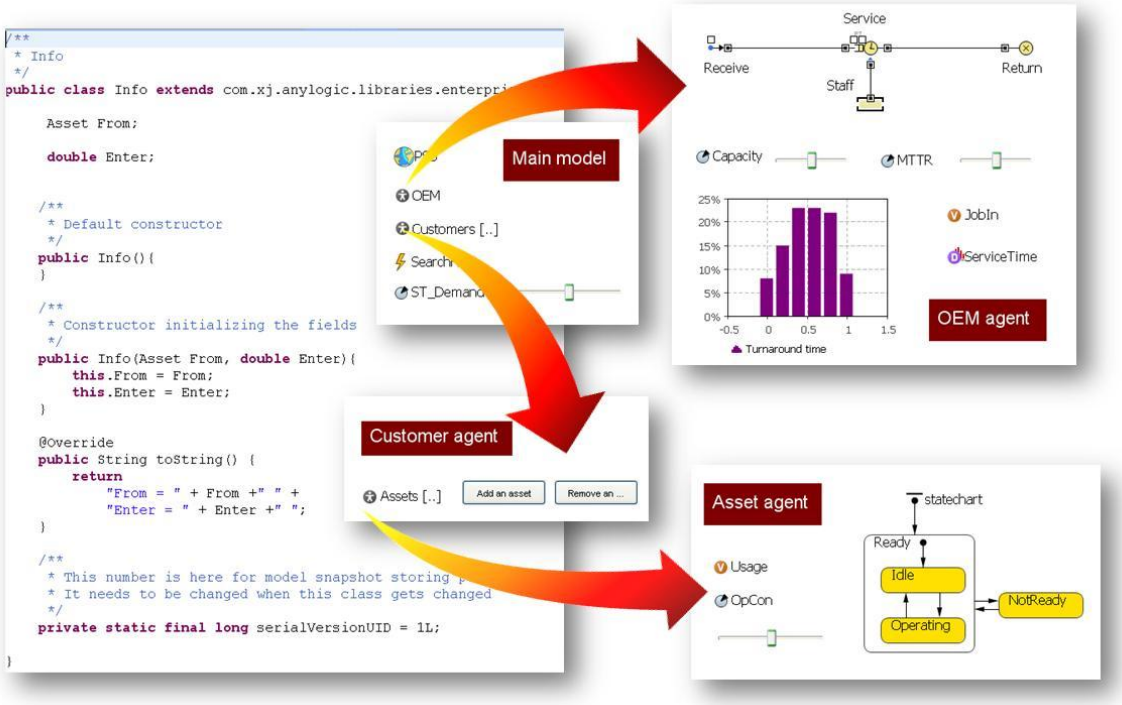


Figure L-1: The shared service contract model

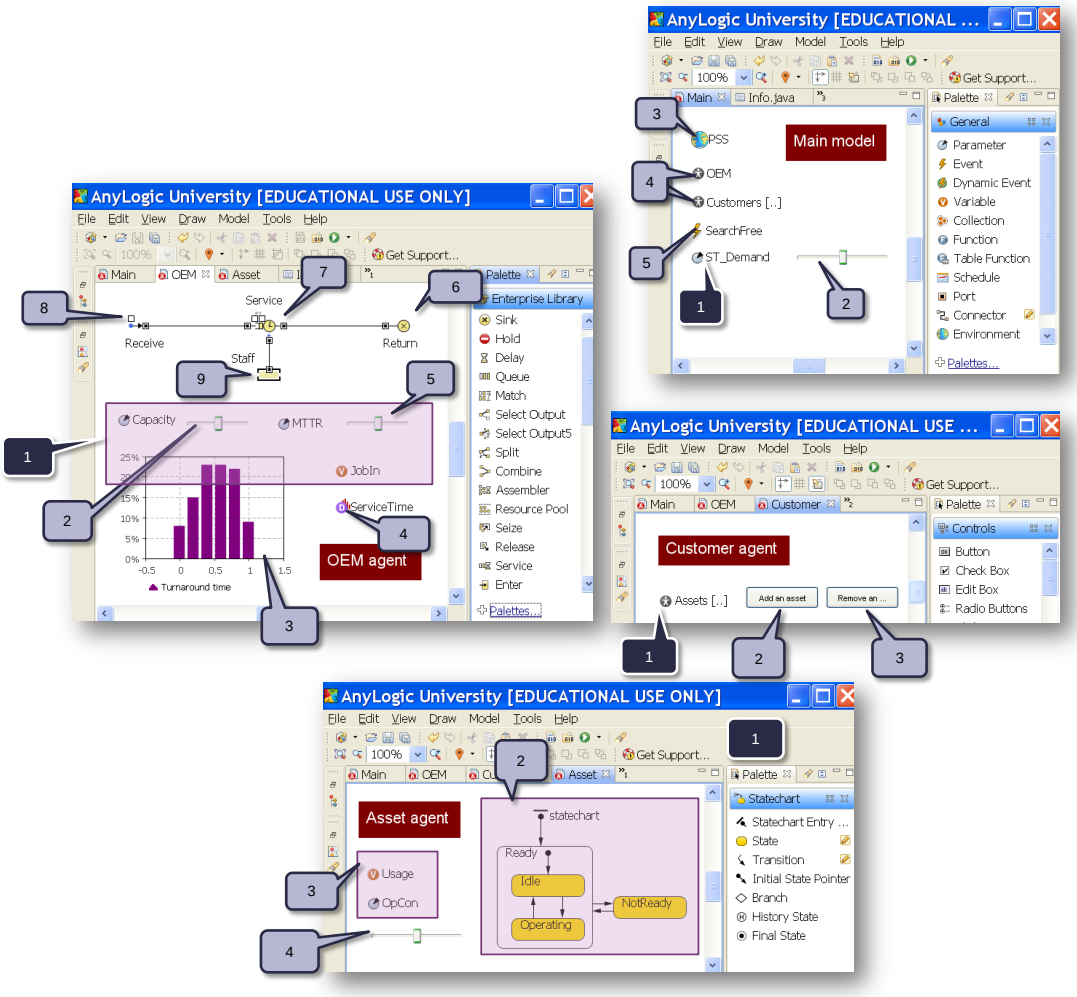


Figure L-2: The shared service contract model in AnyLogic

Main model

1. Drag and drop a *Parameter* from the *General* Pallete.

   Name: ST_Demand

2. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: ST_Demand

   Action code: set_ ST_Demand (value);

3. Drag and drop an *Environment* from the *General* Pallete.

Name: PSS

4. Create an OEM and Customer agents and define their environments.

Environment: PSS

5. Drag and drop an *Event* from the *General* Pallete.

Trigger type: Rate M**ode:** ST_Demand

Action code:

```
int m=0; int n=0;
for (Customer c: Customers){
        for (Asset s: c.Assets){
                if (s.statechart.isStateActive(Asset.Idle)){
                        m =c.getIndex();
                        n=s.getIndex();}}}
if (Customers.get(m).Assets.get(n).statechart.isStateActive(Asset.Idle)){
        send("Work!",Customers.get(m).Assets.get(n));}
```

OEM agent

1. Create these elements from the *General* Pallete.

2. Drag and drop a *Slider* from the *Control* Pallete.

Input Min and Max values.

Default value: Capacity

Action code: set_Capacity((int)value);

3. Drag and drop a *Histogram* from the *Analysis* Pallete.

Histogram: ServiceTime

4. Drag and drop a *Histogram Data* from the *Analysis* Pallete.

    Name: ServiceTime

5. Drag and drop a *Slider* from the *Control* Pallete.

    Input Min and Max values.

    Default value: MTTR

    Action code: set_MTTR(value);

6. Drag and drop a *Sink* from the *Enterprise Library* Pallete.

    Entity class: Info

    On enter:

    send("Approved!", entity.From);

    ServiceTime.add(time()-entity.Enter);

    JobIn--;

7. Drag and drop a S*ervice* from the *Enterprise Library* Pallete.

    Entity class: Info

    Delay time: normal(0.05*MTTR,MTTR)

8. Drag and drop an *Enter* from the *Enterprise Library* Pallete.

    Element class: Info

    On enter:

    entity.Enter=time();

    JobIn++;

9. Drag and drop a *Resource Pool* from the *Enterprise Library* Pallete.

    Capacity: Capacity

Customer agent

1. Create an Asset agent.

2. Drag and drop a *Button* from the *Control* Pallete.

    Action : add_Assets();

3. Drag and drop a *Button* from the *Control* Pallete.

    Action:

```
int m=0;
if (Assets.size()>0){
        for (Asset s: Assets){
                if (s.statechart.isStateActive(Asset.Idle)){
                        m=s.getIndex();}}
        if (Assets.get(m).statechart.isStateActive(Asset.Idle)){
                remove_Assets(Assets.get(m));}}
```

Asset agent

1. Go to the *Agent* tab of the *Properties* window.

    On message received: statechart.receiveMessage((String)msg);

2. Create these elements from the *Statechart* Pallete.

3. Create these elements from the *General* Pallete.

4. Drag and drop a *Slider* from the *Control* Pallete.

    Input Min and Max values.

    Default value: OpCon

    Action code: set_OpCon(value);

A0 Model

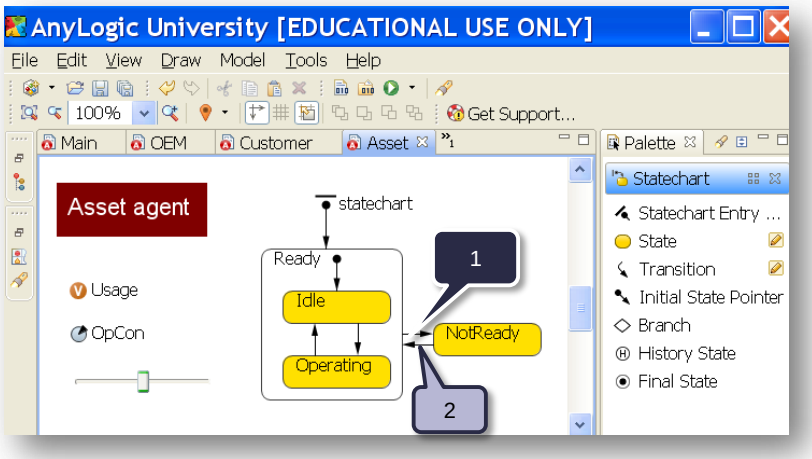


Figure L-3: A0 model in AnyLogic

Asset agent

1. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: "ToOEM!"

   Action:

   Info thisAsset = new Info();

   thisAsset.From=this;

   get_Customer().get_Main().OEM.enter.take(thisAsset);

3. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: "Approved!"

A1 model

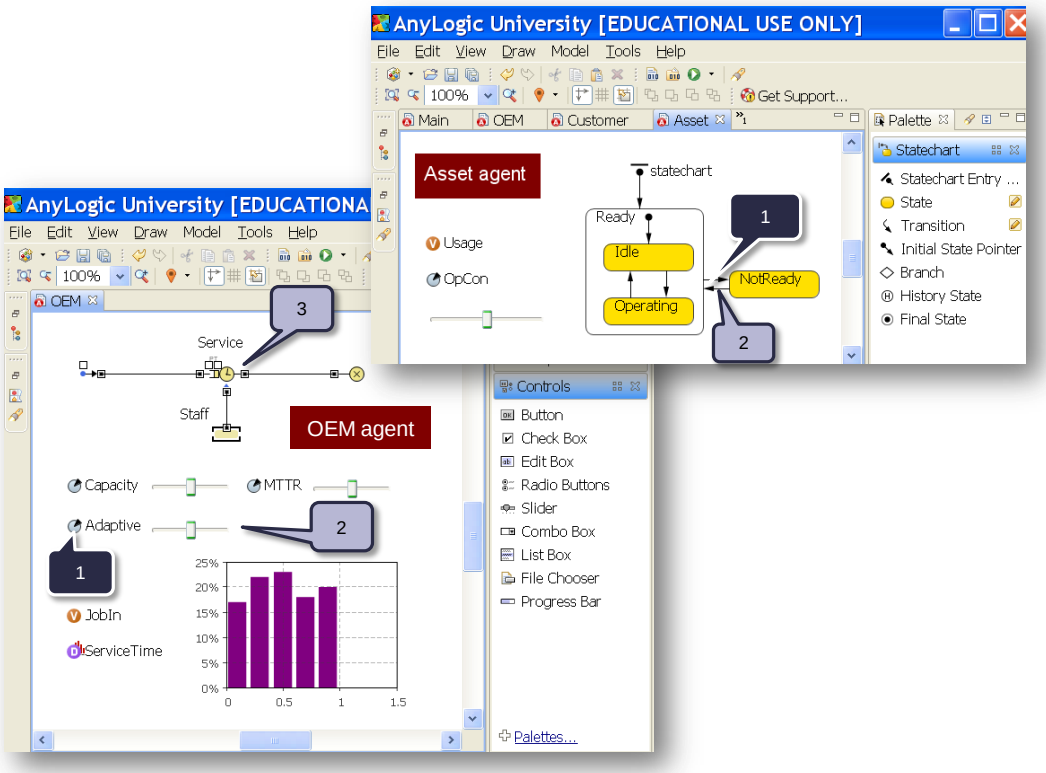


Figure L-4: A1 model in AnyLogic

Asset agent

1. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: "ToOEM!"

   Action:

   Info thisAsset = new Info();

   thisAsset.From=this;

   get_Customer().get_Main().OEM.enter.take(thisAsset);

2. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: "Approved!"

OEM agent

1. Drag and drop a *Parameter* from the *General* Pallete.

2. Drag and drop a *Slider* from the *Control* Pallete.

    Input Min and Max values.

    Default value: Adaptive

    Action code: set_Adaptive((int)value);

3. **Delay:**

    Service.queueSize()>Adaptive?normal(0.025*MTTR,MTTR/2):
    normal(0.05*MTTR,MTTR)

A2 model

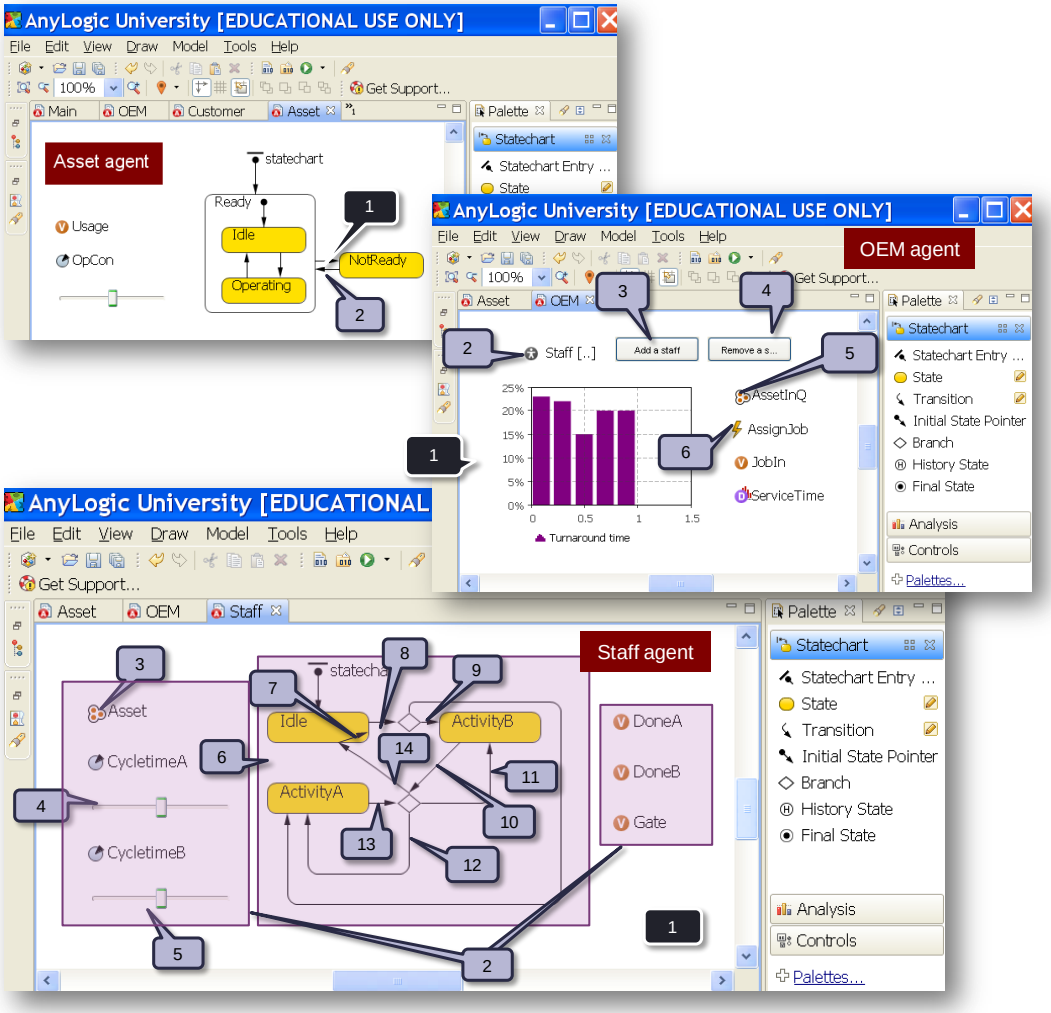


Figure L-5: A2 model in AnyLogic

Asset agent

1. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: "ToOEM!"

   Action:

   Info thisAsset = new Info();

   thisAsset.From=this;

send(thisAsset, get_Customer().get_Main().OEM);

2. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: “Approved!"

OEM agent

1. Go to the *Agent* tab of the *Properties* window.

   On message received: AssetInQ.add((Info)msg);

2. Create a Staff agent.

3. Drag and drop a *Button* from the *Control* Pallete.

   Action: add_Staff();

4. Drag and drop a *Button* from the *Control* Pallete.

   Action:

```
int x=0;
for (Staff s:Staff){
        if (s.Asset.size()==0){
                x = s.getIndex();}}
if (Staff.get(x).Asset.size()==0){
        remove_Staff(Staff.get(x));}
```

5. Drag and drop a *Collection* from the *General* Pallete.

   Element class: Info

6. **Trigger type:** Condition **Condition:** AssetInQ.size()>0

   Action code:

```
int s=0;
for (int i=0;i<Staff.size();i++){
```

```
            if (Staff.get(s).Asset.size()>Staff.get(i).Asset.size()){
                s=i;}}
        if (Staff.get(s).Asset.size()==0){
            send(AssetInQ.get(0), Staff.get(s));}
        AssetInQ.remove(0);
        AssignJob.restart();
```

Staff agent

1. Go to the *Agent* tab of the *Properties* window.

   On message received: Asset.add((Info)msg);

2. Create the elements from the *Contro*l Pallete.

3. **Element class:** Info;

4. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: CycletimeA

   Action code: set_ CycletimeA(value);

5. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: CycletimeB

   Action code: set_ CycletimeB(value);

6. Create the elements from the *Statechart* Pallete.

7. **Trigger by:** Condition

   Condition: (Asset.size()>0)&&(Gate==0)

   Action:

statechart.fireEvent("Work!");

Gate=1;

8. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: “Work!"

9. **Trigger by:** Condition **Condition:** randomTrue(0.5)

10. **Trigger by:** Timeout

    Timeout:

    (Asset.size()>2)?(int)normal(0.05*CycletimeB/2,CycletimeB/2):
    (int)normal(0.05*CycletimeB,CycletimeB)

    Action: DoneB=true;

11. **Trigger by:** Condition **Condition:** DoneB==false

12. **Trigger by:** Condition **Condition:** DoneA==false

13. **Trigger by:** Timeout

    Timeout:

    (Asset.size()>2)?(int)normal(0.05*CycletimeA/2,CycletimeA/2):
    (int)normal(0.05*CycletimeA,CycletimeA)

    Action: DoneA=true;

14. **Trigger by:** Condition

    Condition: (DoneA==true)&&(DoneB==true)

    Action:

    send("Approved!", Asset.get(0).From);

    get_OEM().ServiceTime.add(time()-Asset.get(0).Enter);

    Asset.remove(0);

    DoneA=false;

DoneB=false;

Gate=0;

B0 model

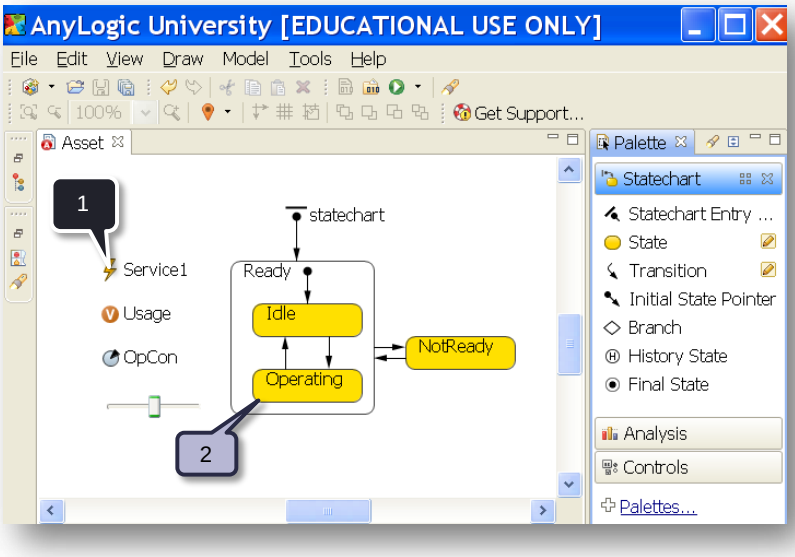


Figure L-6: B0 model in AnyLogic

Asset agent

1. Drag and drop an *Event* from the *General* Pallete.

   Trigger type: Timeout **Mode:** Cyclic

   First occurrence time: Input the time between services

   Recurrence time: Input the time between services

   Action code: statechart.fireEvent("ToOEM!");

2. **On exit:** Usage=Usage+(int)normal(0.05*OpCon, OpCon);

B1 model

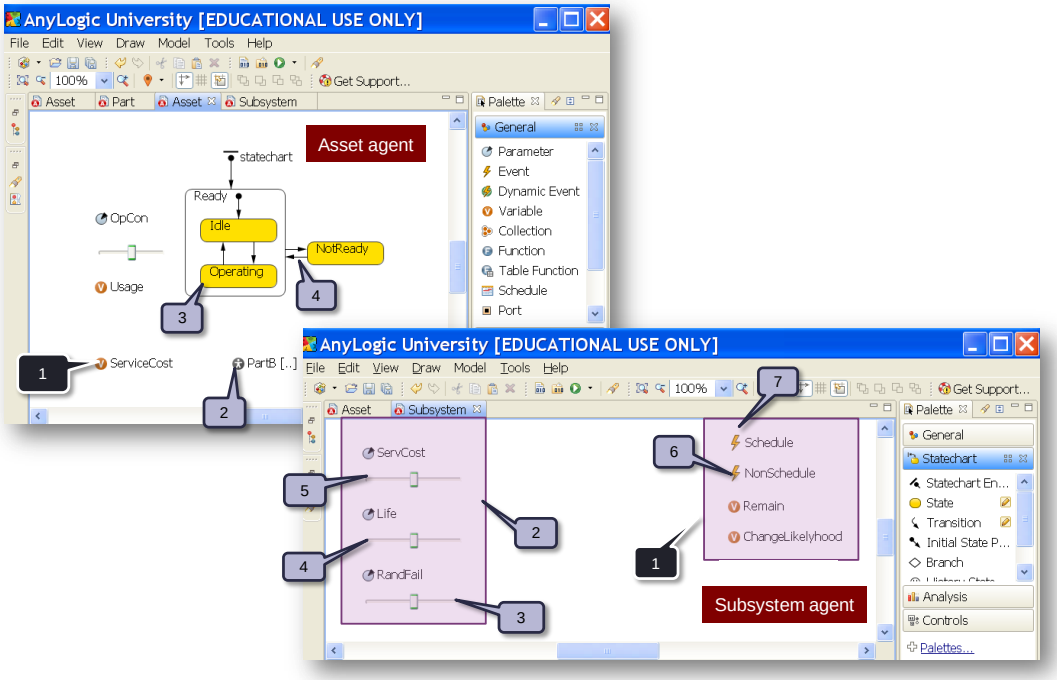


Figure L-7: B1 model in AnyLogic

Asset agent

1. Create the variables.
2. Create the Part agent.
3. **On exit:**

   int m= (int)normal(0.05*OpCon, OpCon);

   Usage=Usage+m;

   for (Subsystem p:PartB){

   p.Remain=p.Remain-m;}

4. Add action

   Action:

   for (Subsystem p:PartB){

p.Gate=0;}

Subsystem agent

1. Create the elements from the *Control* Pallete.
2. Create the elements from the *Control* Pallete.
3. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: RandFail

   Action code: set_RandFail(value);
4. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: Life

   Action code: set_Life((int)value);
5. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: ServCost

   Action code: set_ServCost(value);
6. **Trigger type:** Rate**Rate:** RandFail

   Action code:

   if (Gate==0){

   Gate=1;

   send("ToOEM!", get_Asset());

   Remain=(int)normal(5,Life);

   get_Asset().ServiceCost=get_Asset().ServiceCost+ServCost;}

7. **Trigger type:** Timeout **Mode:** Cyclic

First occurrence time: 0 **Recurrence time:** 1

Action code:

```
ChangeLikelyhood=1000/Remain;
if
((ChangeLikelyhood>100)&&(get_Asset().statechart.isStateActive(NotReady))){
        Gate=1;
        Remain=(int)normal(5,Life);
        get_Asset().ServiceCost=get_Asset().ServiceCost+ServCost;}
if ((Remain<=0)&&(Gate==0)){
        Gate=1;
        get_Asset().statechart.fireEvent("ToOEM!");
        Remain=(int)normal(5,Life);
        get_Asset().ServiceCost=get_Asset().ServiceCost+ServCost;}
```

C1 model

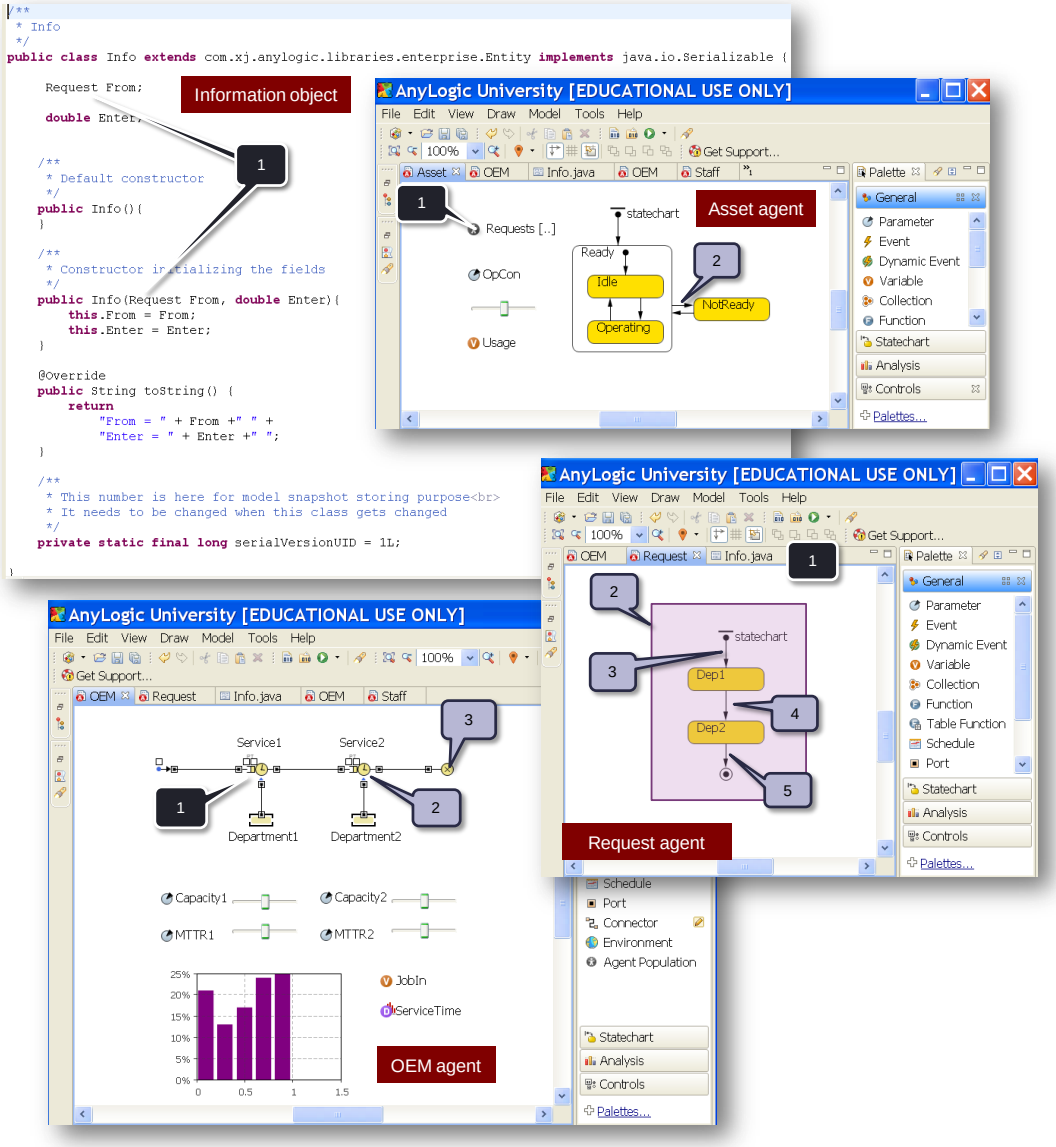


Figure L-8: C1 model in AnyLogic

Information object

1. Change to Request.

Asset agent

1. Create the Request agent.

2. Change the action code.

Action: add_Requests();

Request agent

1. Go to the *Agent* tab of the *Properties* window.

   On message received: statechart.receiveMessage((String)msg);

2. Create the elements from the *Statechart* Pallete.

3. **Action:**

   Info thisAsset = new Info();

   thisAsset.From=this;

   get_Asset().get_Customer().get_Main().OEM.enter.take(thisAsset);

4. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: “Done1!"

5. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: “Done2!“

   Action:

   get_Asset().statechart.fireEvent("Approved!");

   get_Asset().remove_Requests(this);

OEM agent

1. **On exit:** send("Done1!", entity.From);

2. **On exit:** send("Done2!", entity.From);

3. Remove the code

   send("Approved!", entity.From);

C2 model

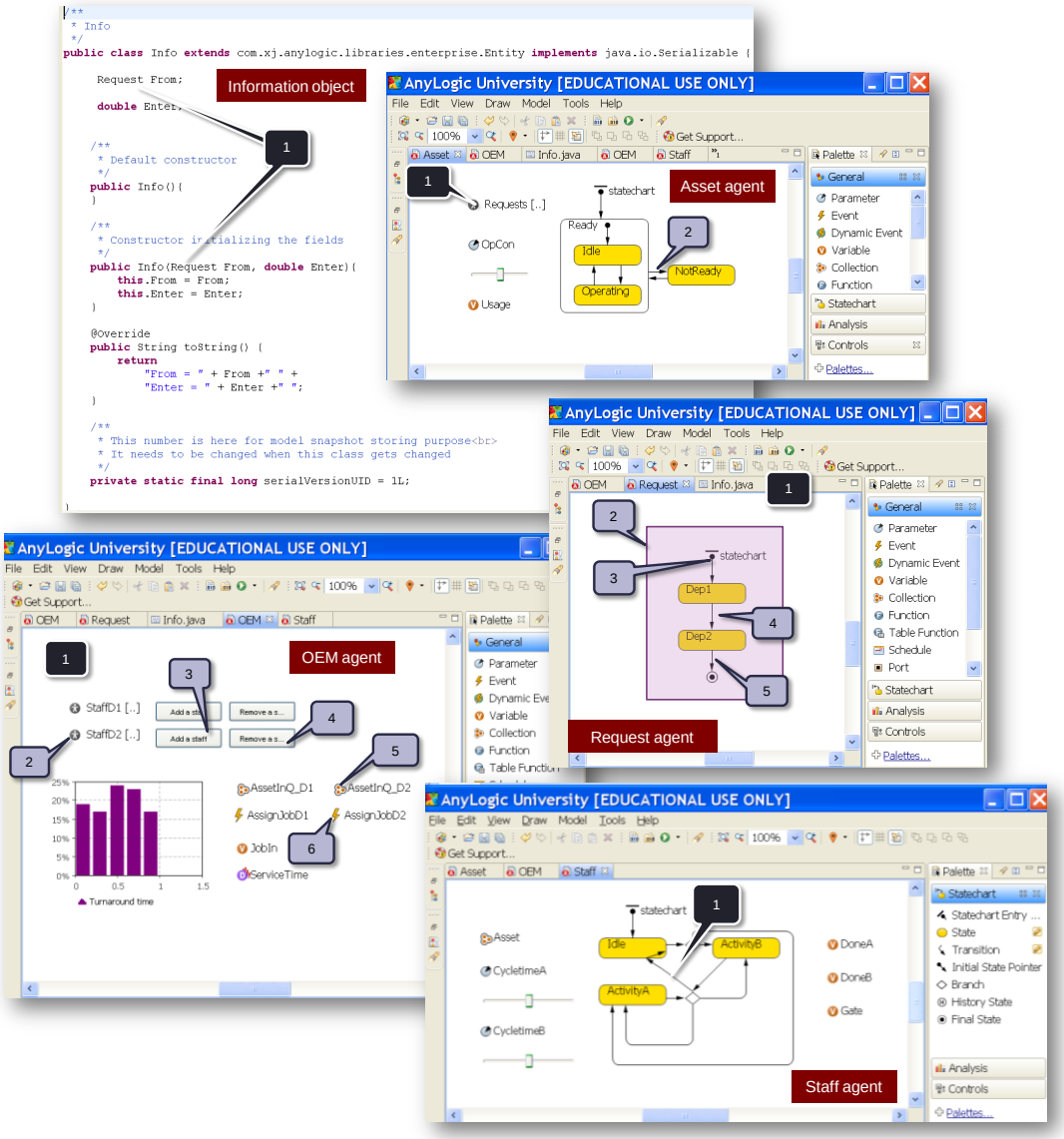


Figure L-9: C2 model in AnyLogic

Information object

1. Change to Request

Asset agent

1. Create a Request agent.

2. Change the action code.

Action: add_Requests();

Request agent

1. Go to the *Agent* tab of the *Properties* window.

   On message received: statechart.receiveMessage((String)msg);

2. Create the elements from the *Statechart* Pallete.

3. **Action:**

   Info thisAsset = new Info();

   thisAsset.From=this;

   get_Asset().get_Customer().get_Main().OEM.AssetInQ1.add(thisAsset);

4 **Trigger by:** Message **Message type:** String

   Fire transition if message equals: “Done1!"

5 **Trigger by:** Message **Message type:** String

   Fire transition if message equals: “Done2!“

   Action:

   get_Asset().statechart.fireEvent("Approved!");

   get_Asset().remove_Requests(this);

OEM agent

1. Go to the *Agent* tab of the *Properties* window and remove *On message received* code.

2. Create a StaffD2 agent.

3. Drag and drop a *Button* from the *Control* Pallete.

   Action: add_StaffD2();

4. Drag and drop a *Button* from the *Control* Pallete.

   Action:

int x=0;

for (Staff s:StaffD2){

if (s.Asset.size()==0){

x = s.getIndex();}}

if (StaffD2.get(x).Asset.size()==0){

remove_StaffD2(StaffD2.get(x));}

5. Drag and drop a *Collection* from the *Genera*l Pallete.

Element class: Info

6. **Trigger type:** Condition **Condition:** AssetInQ2.size()>0

Action code: int s=0;

for (int i=0;i<Staff.size();i++){

if (Staff.get(s).Asset.size()>Staff.get(i).Asset.size()){

s=i;}}

if (Staff.get(s).Asset.size()==0){

send(AssetInQ2.get(0), Staff.get(s));}

AssetInQ2.remove(0);

AssignJob2.restart();

Staff agent

1. Change the action code to

Action:

if (Asset.get(0).Stage==2){

send("Done2!", Asset.get(0).From);

get_OEM().ServiceTime.add(time()-Asset.get(0).Enter);

```
}else{
        send("Done1!", Asset.get(0).From);
        get_OEM().AssetInQ_D2.add(Asset.get(0));}
Asset.remove(0);
DoneA=false;
DoneB=false;
Gate=0;
```

D1 model

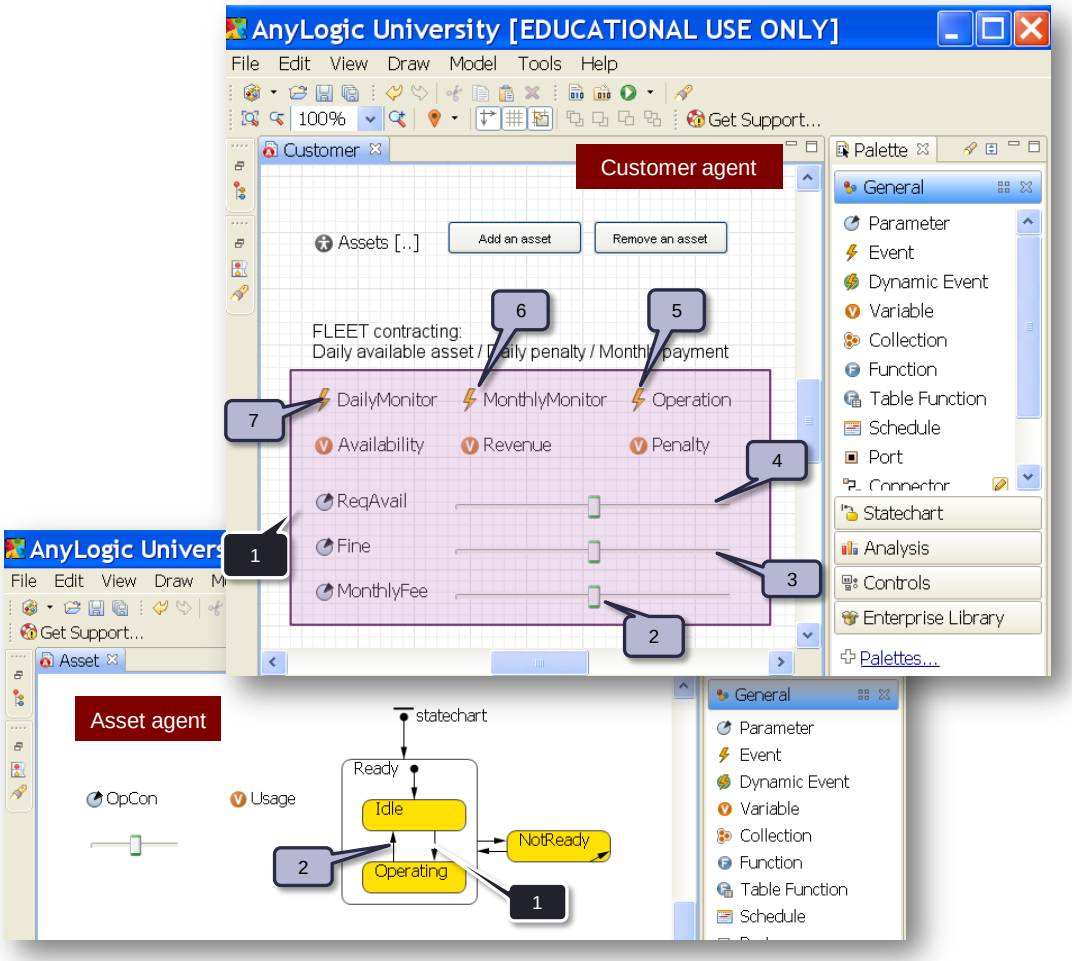


Figure L-10: D1 model in AnyLogic

Customer agent

1. Create these elements from the *General* Pallete.

2. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: MonthlyFee

   Action code: set_MonthlyFee(value);

3. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: Fine

   Action code: set_Fine(value);

4. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: ReqAvail

   Action code: set_ReqAvail((int)value);

5. Input frequency of asset operations.

   Action code:

```
int m=0;
if (Assets.size()>0){
        for (Asset s: Assets){
                if (s.statechart.isStateActive(Asset.Idle)){
                        m=s.getIndex();}}
        if (Assets.get(m).statechart.isStateActive(Asset.Idle)){
                Assets.get(m).statechart.fireEvent("Work!");}}
```

6. **Trigger type:** Timeout **Mode:** Cyclic

First occurrence time: 720

Recurrence time: 720

Action code:

```
if (Assets.size()>0){
        Revenue=Revenue+MonthlyFee;}
```

7. **Trigger type:** Timeout **Mode:** Cyclic

First occurrence time: 24 **Recurrence time:** 24

Action code:

```
int m=0;
for (Asset s:Assets){
        if (s.statechart.isStateActive(Asset.Ready)){
                m++;}}
if ((m<ReqAvail)&&(Assets.size()>0)){
        Penalty=Penalty+Fine;
        Availability=100*m/ReqAvail;
}else{
    Availability=100;}
```

Asset agent

1 **Trigger by:** Message **Message type:** String

Fire transition if message equals: “Work!"

2. **Trigger by:** Timeout

Timeout: Input asset operating duration

D2 model

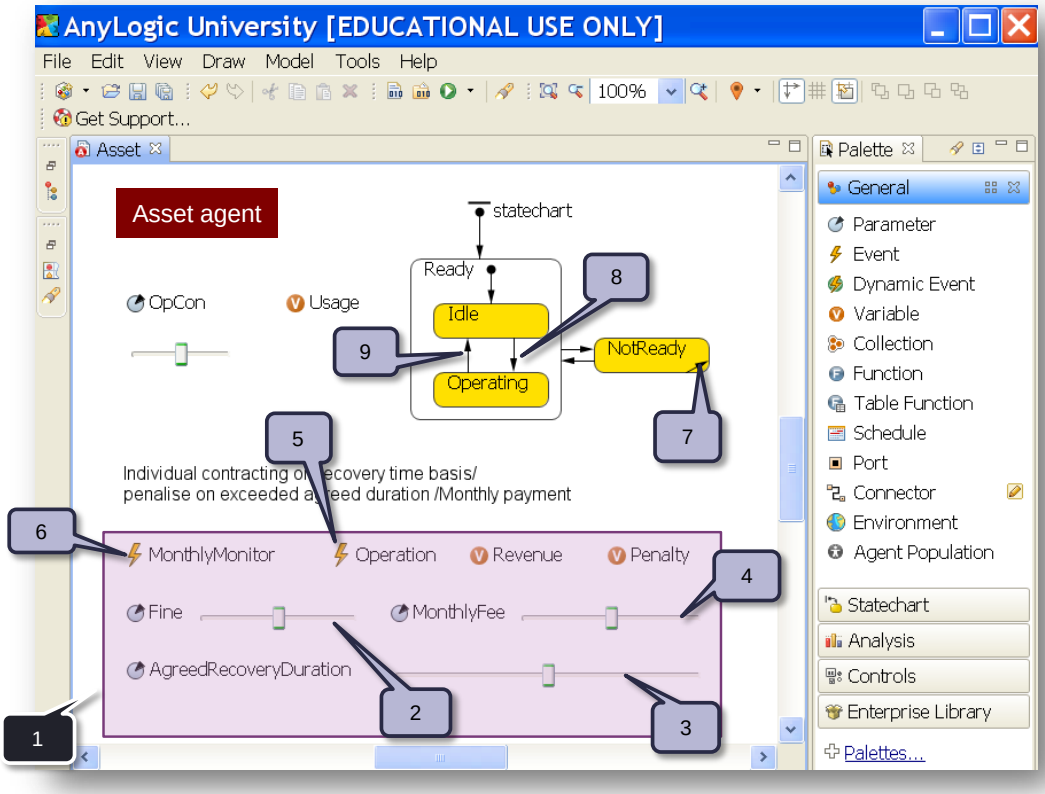


Figure L-11: D2 model in AnyLogic

Asset agent

1. Create these elements from the *General* Pallete.

2. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: Fine

   Action code: set_Fine(value);

3. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: AgreedRecoveryDuration

   Action code: set_ AgreedRecoveryDuration (value);

4. Drag and drop a *Slider f*rom the *Control* Pallete.

   Input Min and Max values.

   Default value: MonthlyFee

   Action code: set_MonthlyFee(value);

5. Input frequency of asset operation.

   Action:

   if (statechart.isStateActive(Idle)){

   statechart.fireEvent("Work!"); }

6. **Trigger type:** Timeout **Mode:** Cyclic

   First occurrence time: 720

   Recurrence time: 720

   Action code: Revenue=Revenue+MonthlyFee;

7. Drag and drop a *Transition* from the *Statechart* Pallete.

   Trigger by: Timeout

   Timeout: AgreedRecoveryDuration

   Action code: Penalty=Penalty+Fine;

8. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: “Work!"

9. **Trigger by:** Timeout

   Timeout: Input asset operating duration

D3 model

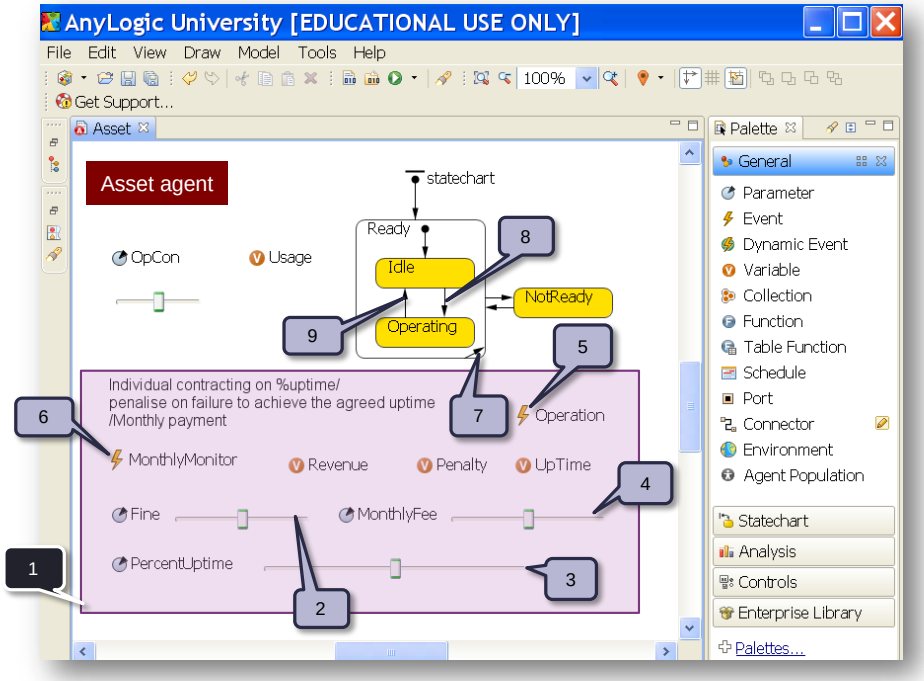


Figure L-12: D3 model in AnyLogic

Asset agent

1. Create these elements from the *General* Pallete.

2. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: Fine

   Action code: set_Fine(value);

3. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: PercentUptime

   Action code: set_PercentUptime(value);

4. Drag and drop a *Slider* from the *Control* Pallete.

Input Min and Max values.

Default value: MonthlyFee

Action code: set_MonthlyFee(value);

5. Input frequency of asset operation.

Action:

```
if (statechart.isStateActive(Idle)){
        statechart.fireEvent("Work!"); }
```

6. **Trigger type:** Timeout **Mode:** Cyclic

First occurrence time: 720

Recurrence time: 720

Action code:

```
Revenue=Revenue+MonthlyFee;
if (UpTime*100/720<PercentUptime){
        Penalty=Penalty+Fine;}
UpTime=0;
```

7. Drag and drop a *Transition* from the *Statechart* Pallete.

Trigger by: Timeout

Timeout: 1

Action code: UpTime++;

8. **Trigger by:** Message **Message type:** String

Fire transition if message equals: “Work!"

9. **Trigger by:** Timeout

Timeout: Input asset operating duration

D4 model

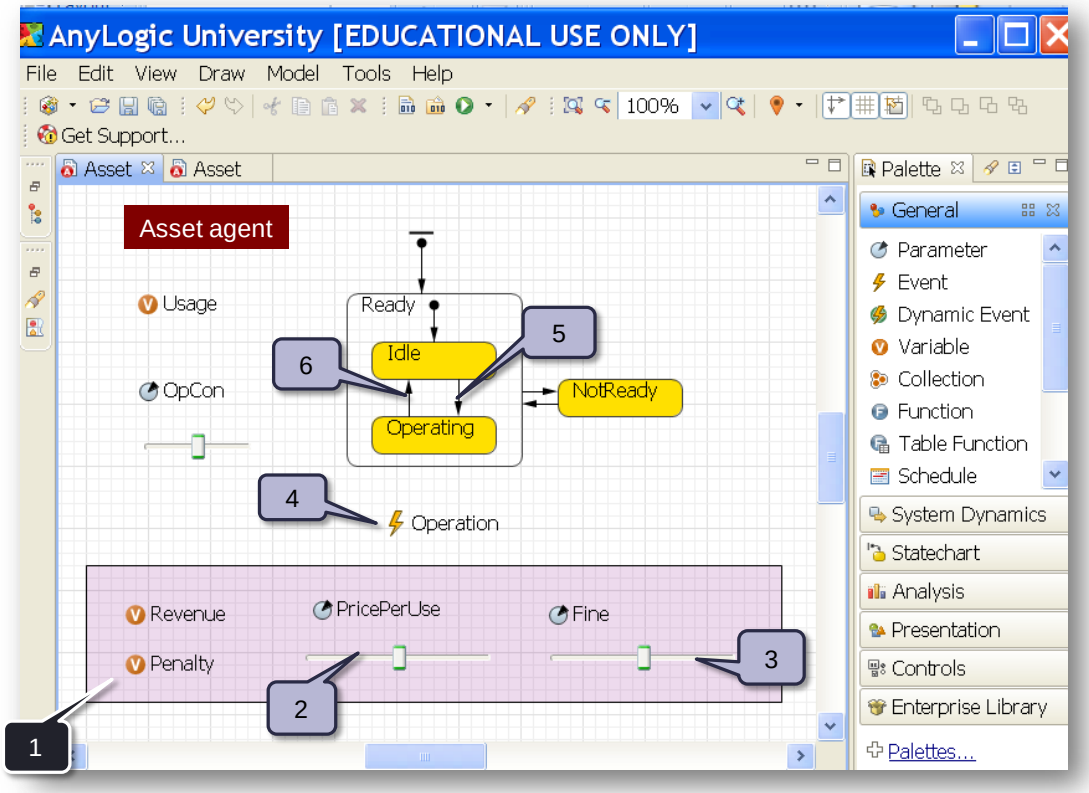


Figure L-13: D4 model in AnyLogic

Asset agent

1. Create these variables and parameters from the *General* Pallete.

2. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: PricePerUse

   Action code: set_PricePerUse(value);

3. Drag and drop a *Slider* from the *Control* Pallete.

   Input Min and Max values.

   Default value: Fine

   Action code: set_Fine(value);

4. Drag and drop an *Event* from the *General* Palette.

   Input frequency of asset operation.

   Action:

   if (statechart.isStateActive(Idle)){

   statechart.fireEvent("Work!"); }

5. **Trigger by:** Message **Message type:** String

   Fire transition if message equals: “Work!"

   Action:

   Penalty=Penalty+Fine;

   Revenue=Revenue+PricePerUse;

6. **Trigger by:** Timeout

   Timeout: Input asset operating duration

   Action:

   Penalty=Penalty-Fine;