INTELLIGENT MAINTENANCE MANAGEMENT IN A RECONFIGURABLE MANUFACTURING ENVIRONMENT USING MULTI-AGENT SYSTEMS

De Ville Weppenaar

Thesis submitted in fulfilment of the requirements for the

MAGISTER TECHNOLOGIAE:

ELECTRICAL ENGINEERING

in the

School of Electrical and Computer Systems Engineering

of the

Faculty of Engineering, Information and Communication Technology

at the

Central University of Technology, Free State

Supervisor:	**Prof. H.J. Vermaak**
Co-Supervisor:	**Prof. J.D.M. Kinyua**

Bloemfontein

July 2010

Declaration of Independent Work

I, **DE VILLE IAN WEPPENAAR**, identity number [redacted], and student number **20414927**, do hereby declare that this research project has been submitted to the Central University of Technology for the degree **MAGISTER TECHNOLOGIAE: ELECTRICAL ENGINEERING**, is my own independent work; and complies with the Code of Academic Integrity, as well as other relevant policies, procedures, rules and regulations of the Central University of Technology; and has not been submitted before by any person in fulfilment (or partial fulfilment) of the requirements for the attainment of any qualification

……………………………………………………… ……………………………………………..

SIGNATURE OF STUDENT **DATE**

Verklaring Ten Opsigte van Selfstandige Werk

Ek, **DE VILLE IAN WEPPENAAR**, identiteitsnommer **850903 5062 084**, en studentenommer **20414927**, verklaar hiermee dat hierdie navorsingsprojek ingedien is aan die Sentrale Universiteit van Tegnologie ter verwerwing van die kwalifikasie **MAGISTER TECHNOLOGIAE: Elektriese Ingenieurswese**, dat dit my selfstandige werk is; voldoen aan die kode van akademiese integriteit, so wel as ander relevante beleide, reëls en regulasies van die Sentrale Universiteit van Tegnologie; en is geensins in die verlede al ingedien ter verwerwing (of gedeeltelike verwerwing) van enige ander kwalifikasie nie.

………………………………………………………… ……………………………………………….

HANDTEKENING VAN STUDENT **DATUM**

Acknowledgements

I would like to thank the following individuals:

- God Almighty, for without His grace and love I would never have had the strength and perseverance needed in order to see this project through to completion.
- Prof. Johnson Kinyua, for his excellent guidance, words of wisdom and keen eye.
- Prof. Herman Vermaak, for encouragement, time and keeping the project (and me) on the right track.
- The Central University of Technology (CUT) for the opportunity to conduct this research project.
- The National Research Foundation (NRF) and specifically the Advanced Manufacturing Technology Strategy (AMTS), for without their funding (under project number AMTS-07-20-P) the completion of this study would not have been possible.
- My loving wife Liza-Marié, for believing in me and her constant words of encouragement.
- Friends and family for their continued love and support throughout the duration of this project.

Abstract

Traditional corrective maintenance is both costly and ineffective. In some situations it is more cost effective to replace a device than to maintain it; however it is far more likely that the cost of the device far outweighs the cost of performing routine maintenance. These device related costs coupled with the profit loss due to reduced production levels, makes this reactive maintenance approach unacceptably inefficient in many situations. Blind predictive maintenance without considering the actual physical state of the hardware is an improvement, but is still far from ideal. Simply maintaining devices on a schedule without taking into account the operational hours and workload can be a costly mistake.

The inefficiencies associated with these approaches have contributed to the development of proactive maintenance strategies. These approaches take the device health state into account. For this reason, proactive maintenance strategies are inherently more efficient compared to the aforementioned traditional approaches. Predicting the health degradation of devices allows for easier anticipation of the required maintenance resources and costs. Maintenance can also be scheduled to accommodate production needs.

This work represents the design and simulation of an intelligent maintenance management system that incorporates device health prognosis with maintenance schedule generation. The simulation scenario provided prognostic data to be used to schedule devices for maintenance. A production rule engine was provided with a feasible starting schedule. This schedule was then improved and the process was determined by adhering to a set of criteria. Benchmarks were conducted to show the benefit of optimising the starting schedule and the results were presented as proof.

Improving on existing maintenance approaches will result in several benefits for an organisation. Eliminating the need to address unexpected failures or perform maintenance prematurely will ensure that the relevant resources are available when they are required. This will in turn reduce the expenditure related to wasted maintenance resources without compromising the health of devices or systems in the organisation.

Abstrak

Tradisionele korrektiewe instandhouding is beide duur en oneffektief. In sekere omstandighede is dit meer koste effektief om ‘n apparaat te vervang as om dit te onderhou; maar dit is egter meer waarskynlik dat die koste van die apparaat heelwat meer is as die koste om roetine instandhouding te verrig. Die koste van apparate asook die wins verlies wat gepaard gaan met ’n vermindering in produksie uitsette, veroorsaak dat dié reaktiewe instandhoudingstegniek oneffektief geag kan word in meeste situasies. Blinde, voorspelbare instandhouding sonder om die teenswoordige, fisiese status van die hardeware in ag te neem is ’n verbetering, maar is nog steeds onvolmaak. Om net eenvoudig apparate te onderhou volgens ’n skedule, sonder om die werksure en -lading in ag te neem kan ’n duursame fout wees.

Die ondoeltreffendheid wat geassosieer word met hierdie tegnieke het bydrae gelewer tot die ontwikkeling van proaktiewe instandhouding strategieë. Die tegnieke neem die apparaat se toestand in ag. Om hierdie rede is proaktiewe instandhoudingstrategieë meer geskik in vergelyking met die voorgenoemde, tradisionele tegnieke. Deur die ontaarding van apparate te voorspel, word die verwagting van nodige hulpbronne en kostes aangaande instandhouding vergemaklik. Instandhouding kan ook geskeduleer word om produksie benodighede te akkommodeer.

Die studie verteenwoordig die ontwikkeling en simulasie van ’n sisteem wat die intelligente bestuur van instandhouding behartig; deur apparaat ontaarding te voorspel en ’n skedule van instandhouding te genereer. Die data wat gebruik is vir die skedulering van apparate vir instandhouding, is verskaf deur die simulasie. ’n Uitvoerbare, aanvanklike schedule was verskaf aan ’n produksie reël enjin. Die skedule is verbeter en die proses is bepaal deur aan ’n sekere aantal kriteria te voldoen. Vergelykings was getrek om die voordeel aan te dui van ’n aanvanklike skedule wat verbeter is en die resultate is voorgelê as bewys.

‘n Onderneming kan voordeel trek uit die verbetering van bestaande instandhoudingstegnieke. Om te verseker dat die relevante hulpbronne beskikbaar is wanneer dit benodig word, is dit van belang om die onverwagte vertragings of ontydige instandhouding, uit te skakel. Dit lei tot die gevolg dat

die uitgawes aangaande verkwiste onderhoud kostes verminder word, sonder dat die toestand van apparate of sisteme in die onderneming in gevaar gestel word.

Table of Contents

List of Acronyms

AB	–	Allen-Bradley
ADACOR	–	Adaptive and Cooperative Control Architecture for Distributed Manufacturing Systems
ADF	–	Agent Definition File
ADSL	–	Asymmetric Digital Subscriber Line
AGV	–	Autonomous Guided Vehicle
AI	–	Artificial Intelligence
AMTS	–	Advanced Manufacturing Technology Strategy
ANN	–	Artificial Neural Network
AOP	–	Agent Oriented Programming
AOSE	–	Agent Oriented Software Engineering
API	–	Application Program Interface
ATDW	–	Australian Tourism Data Warehouse
BDI	–	Belief-Desires-Intentions
BLiP	–	Business Logic integration Platform
BN	–	Bayesian Networks
BPMS	–	Business Process Management System
BRMS	–	Business Rules Management System
CMMS	–	Computerised Maintenance Management Systems
COM	–	Component Object Model
cRIO	–	CompactRIO
CSV	–	Comma Separated Value
CUT	–	Central University of Technology

DA	–	Data Access
DAG	–	Directed Acyclic Graph
DAI	–	Distributed Artificial Intelligence
DC	–	Direct Current
DCOM	–	Distributed Component Object Model
DCS	–	Distributed Control Systems
DHCP	–	Dynamic Host Configuration Protocol
DoL	–	Department of Labour
DWT	–	Discrete Wavelet Transform
eCAT	–	Continuous Agent Testing on Eclipse
EKF	–	Extended Kalman Filter
EMF	–	Eclipse Modelling Framework
ERD	–	Entity Relationship Diagram
ES	–	Expert System
FIFO	–	First-In-First-Out
FIPA	–	Foundation for Intelligent Physical Agents
FIS	–	Fuzzy Inference System
FLS	–	Fuzzy Logic System
FPGA	–	Field-Programmable Gate Array
FPGA	–	Field-Programmable Gate Array
GA	–	Genetic Algorithms
GEF	–	Graphical Editing Framework
GPL	–	GNU General Public Licence
GUI	–	Graphical User Interface

HMM	–	Hidden Markov Model
HSM	–	Hot Strip Mill
ICA	–	Independent Component Analysis
ICT	–	Information and Communication Technologies
IDE	–	Interactive Development Environment
IMMS	–	Intelligent Maintenance Management System
IO	–	Input-Output
IP	–	Internet Protocol
JDE	–	JACK Development Environment
KF	–	Kalman Filter
MAC	–	Media Access Control
MAS	–	Multi-Agent System
MaSE	–	Multi-agent Systems Engineering
MCSA	–	Motor Current Signature Analysis
MDA	–	Model-Driven Architecture
MFL	–	Magnetic Fux Leakage Signal
MPAEX	–	Multi-agent Pareto Appointment Exchanging
MRA	–	Multi Resolution Analysis
NAS	–	Network Attached Storage
NI	–	National Instruments
NUCREC	–	Need Urgency, Customer Rank and Equipment Criticality
OLE	–	Object Linking and Embedding
OMG	–	Object Management Group
OPC	–	OLE for Process Control

OWL	–	Web Ontology Language
PAS	–	Patient Admission Scheduling or Process Automation System
PASSI	–	Process for Agent Societies Specification and Implementation
PDT	–	Prometheus Design Tool
PLC	–	Programmable Logic Controller
POMAESS	–	Problem-Oriented Multi-Agent-Based E-Service System
POP	–	Post Office Protocol
PRT	–	Platinum Resistance Thermometer
PSO	–	Particle Swarm Optimisation
PTK	–	PASSI Toolkit
RAID	–	Redundant Array of Independent/Inexpensive Discs
RAS	–	Reconfigurable Assembly Systems
REBEL	–	The Roadmap Editor Built for Easy development
RFID	–	Radio Frequency Identification
RGEMS	–	Research Group in Evolvable Manufacturing Systems
RIME	–	Ranking Index for Maintenance Expenditures
ROADMAP	–	Role Oriented Analysis and Design for Multi-Agent Programming
RT	–	Real-Time
SCADA	–	Supervisory Control and Data Acquisition
SMTP	–	Simple Mail Transfer Protocol
SVM	–	Support Vector Machines
TAOM4E	–	Tool for Agent Oriented visual Modelling for the Eclipse
UKF	–	Unscented Kalman Filter
UML	–	Unified Modelling Language

UPS	–	Uninterruptible Power Supply
XMI	–	XML Metadata Interchange
XML	–	Extensible Mark-up Language

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Introduction

In the interest of successfully operating in the competitive global marketplace of today, it is vital for an organisation to optimise its operational costs. Costs relating to maintenance of complex industrial systems are one of the main contributors to the total operating costs of the enterprise and it is estimated that 18-30% thereof is wasted [1]. This statistic shows that there is room for improvement. It is therefore crucial that manufacturing firms optimise the maintenance function.

Traditional maintenance strategies can be classified into two main categories, of which the first is reactive, whereby the equipment is fixed or replaced after it fails. A blindly proactive strategy assumes a certain level of performance degradation, with no input from the machinery itself, and the equipment is serviced on a routine schedule whether service is actually required or not [2]. Both these scenarios are extremely wasteful and leave room for improvement. It is therefore crucial that maintenance be performed in an intelligent manner, whilst ensuring that equipment and systems are properly maintained. The traditional maintenance management systems can be converted to Intelligent Maintenance Management Systems (IMMS), enabling systems to achieve and sustain near-zero breakdown performance, and ultimately transform the traditional maintenance practices from a *fail* and *fix* to *predict* and *prevent* methodologies.

The nature of maintenance planning is changing rapidly with the uptake of condition based maintenance (CBM), integration and e-maintenance. The main idea of CBM is to utilise the device degradation information extracted and identified from on-line sensing techniques to minimise the system downtime by balancing the risk of failure and achievable profits. The decision-making process in CBM focuses on predictive maintenance. This is why many diagnostic tools and methods have been developed. E-maintenance, a new generation of maintenance, can be attributed to globalisation and the fast growth of the communication, computer and information technologies [1]. Distributed organisations can benefit from e-maintenance where people, expertise or data are physically separate or isolated. In addition, e-maintenance also facilitates the transition from mere "predictive maintenance" to intelligent "prognosis".

Device degradation usually occurs over a measurable period of time. Human beings often do not perceive this degradation, and the subsequent failures are frequently misconceived as instantaneous occurrences. By measuring this degradation in some manner, it is possible to estimate when and where a failure of system machinery will occur. Predicting possible failure scenarios and preventing them can extend the life of equipment and reduce unscheduled downtime, which is a significant contributor to profit loss.

Current production machines contain increasingly sophisticated sensors and their computing performance continues to accelerate. It is now possible to rapidly and accurately sense performance indicators, and thus assess and predict system performance [3]. The vast amounts of plants' operational data available to operators can be overwhelming, and important events often go unnoticed. This information can be logged and analysed to provide deterioration trends. These in turn, can be used to determine device prognosis and generate optimised maintenance schedules, subsequently reducing the workload experienced by maintenance personnel. Valuable resources are freed and can be deployed elsewhere. The maintenance information can be readily available to other departments and maintenance tasks synchronisation with production schedules can be simplified. Multi-agent systems (MAS) can help alleviate some of the complexities arising from sensory overload experienced by maintenance personnel. It is also possible, given the nature of MAS, to easily distribute maintenance-related information to the right individuals, in a format they prefer and when it is required.

Recently, agent-oriented software development has been used as a paradigm for software engineering in a wide variety of applications [4]. The agent concept constitutes a powerful abstraction tool when dealing with software development. It is well suited for complex, open and distributed domains such as the Internet, requiring agents to act autonomously and to cooperate, coordinate, communicate, negotiate and even compete [5]. Agents are typically defined by considering the characteristics that they should have, including the following: autonomy, reactivity, sociability, mobility, and pro-activity [6]. In most cases, the solution to complex distributed problems developed using a MAS approach consists of multiple interacting agents acting as computing elements to achieve the systems goals. MAS have the traditional advantages of distributed and concurrent problem solving, with the additional advantage of sophisticated patterns of interactions such as cooperation, coordination and negotiation. The flexibility and high-level nature of these interactions distinguishes multi-agent systems from other forms of software, allowing agents to tackle the increasing complexity of the open software systems where integration, transparency and interoperation among heterogeneous components are essential. Methodologies to aid in the development process, tools and platforms that facilitate the implementation of agent systems have also been developed [7].

Monitoring and determining the prognosis of a device only forms part of the solution. It is certain that several devices will exist in any production or assembly system. Each of these devices will possess its degradation trend and prognostic outcome. In order to ensure that all of these devices and systems stay operational, it is vital to adopt some form of maintenance scheduling. Scheduling remains a great focus area in research, and the International Timetabling Competition (ITC) 2007 is evidence of this fact [8]. There are undeniable similarities between the processes required for maintenance task scheduling and the scenarios presented to the contestants of the ITC. These methods can be adapted to perform maintenance scheduling by providing the rules which govern the process and set the requirements.

1.2 Statement of the Problem

Expenses related to maintenance can potentially account for a large contribution of the total expenses in a company. Traditional corrective maintenance is both costly and ineffective. In some situations it is more cost effective to replace a device than to maintain it; however, it is far more likely that the cost of the device far outweighs the sum of performing routine maintenance. This device-related cost, coupled with the profit loss due to no production, makes this reactive maintenance approach unacceptably inefficient in many situations. With this approach it is difficult to anticipate when a failure will occur and therefore impossible to guarantee the continuous operation of a plant or system.

Blind predictive maintenance without considering the actual physical state of the hardware is an improvement, but is still far from ideal. The cost of performing maintenance tasks on some devices may be negligible when considered as a once-off, infrequent occurrence, but in an organisation where numerous devices are used this amount becomes more than noticeable. Simply maintaining these devices on a schedule without taking into account the operational hours and workload can be a costly mistake. Even when the cost of unnecessary maintenance tasks is ignored, the profit loss attributed to the halt in production is too significant to ignore. The associated wastefulness becomes even more prominent as the number of devices in the system increase.

The inefficiencies associated with these approaches have contributed to the development of proactive maintenance strategies. These approaches take the device health state as their basis. Because of this proactive maintenance strategies are inherently more efficient compared to the aforementioned, traditional approaches. Predicting the health degradation of devices allows for easier anticipation of the maintenance-related resource requirements and cost. Maintenance can also be scheduled to accommodate production needs.

1.3 Research Rationale

Improving on existing maintenance approaches will result in several benefits for an organisation. Eliminating the need to address unexpected failures or perform maintenance prematurely will ensure that the relevant resources are available when they are required. This will, in turn, reduce the expenditure related to wasted maintenance resources without compromising the health of devices or systems in the organisation.

Applying scheduling techniques to generate schedules will simplify a task that is considered tedious and error-prone. The responsible maintenance technician is then free to perform other tasks, further optimising the use of resources. If an existing schedule is rendered obsolete by some unforeseeable action or event, a new replacement schedule is generated with little or no effect on the maintenance processes.

1.4 Research Goal and Objectives

The primary goal of this research work is to improve on existing strategies and to develop a system for maintenance schedule generation based on the health prognosis of a device.

The objectives of this research to support the primary goal include:

- Evaluating artificial intelligence techniques for maintenance management.
- A study of techniques applied in the fields of maintenance and prognostics.
- Scheduling techniques and their application to maintenance scheduling.
- A comparison of methodologies associated with design and implementation of multi-agent systems.
- Assessing the feasibility of an intelligent maintenance management system by implementing a simulation scenario.

1.5 Research Methodology

The literature study was conducted by reviewing published and unpublished reports, articles, academic journals and other publications as well as the Internet to provide a background on the problem that is to be addressed. Previous related research in the respective fields of interest granted insight into prominent issues that should be considered.

Maintenance management was evaluated in order to determine what such a system should entail, and because it forms the basis of the work. E-maintenance, as an emerging technology, was reviewed and the implications it holds for the maintenance field. Several artificial intelligence techniques were studied to assess their feasibility in a system based on e-maintenance. The applications of multi-agent systems in maintenance were studied to determine the possible contribution of incorporating MAS in an intelligent maintenance management system. Since the aim of this work is to create a proactive approach to maintenance management, some techniques that have possible applications in maintenance prognosis were evaluated. Maintenance scheduling was studied as it forms a critical part of the system that is to be a result of this work. The techniques and the problems to which they are commonly applied were evaluated.

Multi-agent systems were studied in more detail, with specific focus on the methodologies used in the design and implementation of such systems. Several methodologies were reviewed and one was selected as the approach to be used in the design of the agents in the system. Some methodology restrictions were proposed in order to facilitate the automatic code generation feature of the interactive development environment (IDE) used by the selected methodology. It was decided to use the IDE to simplify the development of the agent system as opposed to the manual, by-hand alternative. The agents developed with the methodology could be implemented and run on different runtime environments, and for this reason an evaluation was conducted. After the methodology and tool were finalised, the design of the system agents followed.

After some failed attempts to obtain real-world maintenance data, it was decided to perform a simulation to generate the data required for testing the feasibility of the proposed system. The simulation was conducted by means of an induction motor regularly used for laboratory exercises by electrical engineering students at the Central University of Technology. The three-phase motor drives a generator, which in turn, powers individually switchable rows of bulbs. These bulbs function as different load levels; more bulbs mean more power and, in turn, higher resistance from the generator. The measurements from this system were used to simulate several motors, each operated at different and varying load levels during a set period of time. These motors were then used to generate a maintenance schedule and the process was governed by a set of rules that determined the scheduling order. Maintenance technicians were assigned to each of the maintenance tasks and the labour laws of South Africa were incorporated in their scheduling process. The schedules were generated using different settings and starting solutions and the results were compared.

1.6 Organisation of the Dissertation

This dissertation consists of eight chapters. The chapters comprise four sections, which consist of the literature review, an empirical study, the results of the study and conclusions with some recommendations.

The first chapter indicates the importance of improving existing maintenance strategies and highlights the research problem and objectives of the study. Chapter two provides an overview of the component handling platform where the proposed system will be implemented. Chapter three focuses on the importance of maintenance management and the advancement thereof, technologies and techniques used for maintenance prognosis and concludes with an evaluation of maintenance scheduling. Chapter four presents multi-agent systems and how these relate to the study. In this chapter, some of the methodologies associated with the development of multi-agent systems are evaluated and one is selected for use in this study. The tools associated with the selected methodology, as well as the runtime environment are discussed. Chapter five presents the design of the intelligent maintenance management system using the selected methodology. Chapter six discusses the simulation scenario which aims to assess the feasibility of the proposed system and chapter seven presents the results of the simulation scenario. The conclusion is given in chapter eight.

Chapter 2 The Component Handling Platform

2.1 Introduction

The Department of Electrical and Computer Systems Engineering at the Central University of Technology (CUT), Free State [9] has a scaled-down assembly system in one of its labs. The lab was setup and is managed by the Research Group in Evolvable Manufacturing Systems (RGEMS) [10]. The assembly system can be utilised by students and researchers alike for work pertaining to ongoing research endeavours. The assembly system provides a convenient onsite test-platform. Industrial standards are used as far as possible and several cross-discipline research activities are ongoing. Figure 2-1 shows the lab (as it is at the time of writing) in which the automation test platform is situated. It is important to note that this image was taken while ongoing alterations and expansion were in progress.

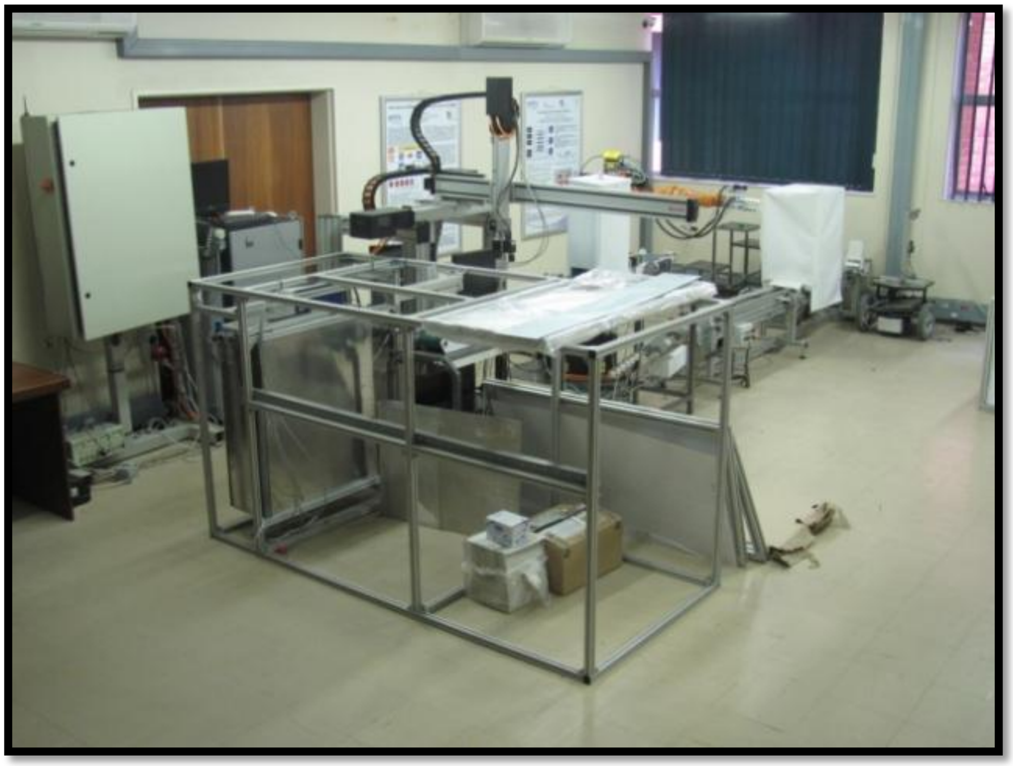


Figure 2-1: Lab 120 at the BHP Billiton building of the Central University of Technology

The industrial test platform is an assembly system and will, upon completion, have several stations, each performing a specific step in the assembly process. Some of the stations can consist of human beings performing tasks that are simply too complex or cost ineffective for machines to perform. The focus of RGEMS is *reconfigurable* manufacturing where the assembly system should be able to assemble products that vary slightly from a base model. The differences in the assembled products should not require a system-wide reconfiguration, but rather slight alterations in the production route or isolated reconfigurations of stations.

Almost all physical installation and configuration is done by students under the supervision of senior students or CUT staff. The majority of the work conducted on or aided by the test platform is for post-graduate studies. Students perform tasks relating to their specific research work or project and coordinate with other students working on different projects.

2.2 Network Layout

An important part of the assembly system is the network that facilitates communication between the different components, systems and entities. Security is also an important consideration in preventing unauthorised access and control of devices and servers as this could have undesirable consequences. The network of the RGEMS assembly system consists of two main divisions. The *IT Core* network provides the infrastructure for day-to-day activities of the lab and the group members; whereas the *Factory* network comprises all the physical hardware devices that form the assembly system. Figure 2-2 illustrates of the network layout and some of the connected devices [11]. The following sub-sections provide an in-depth discussion of the network.

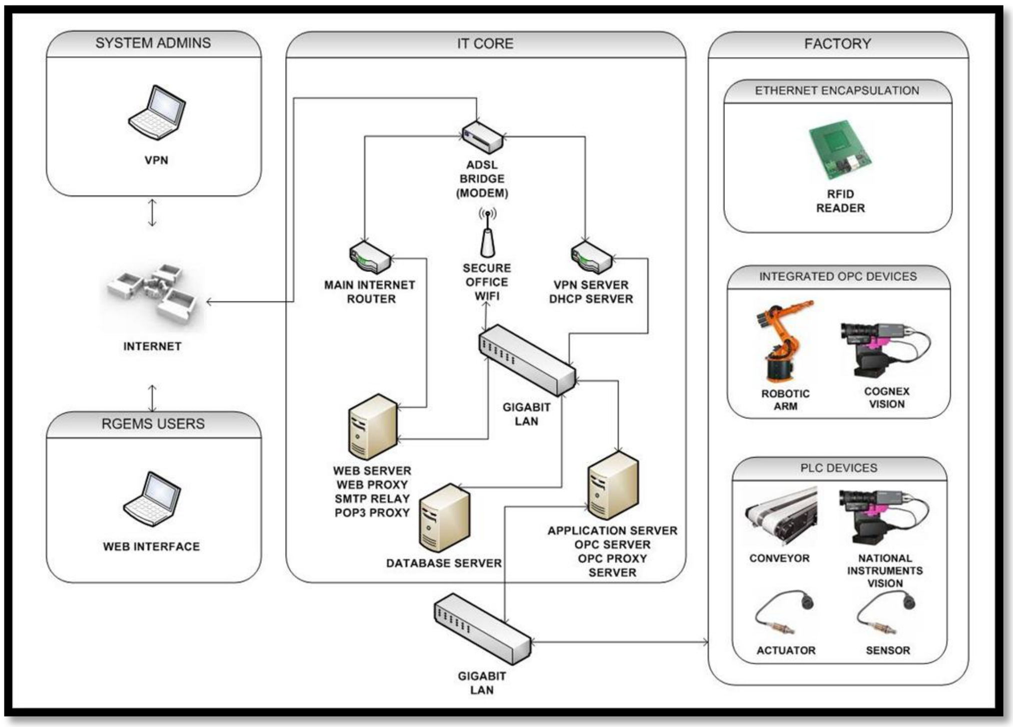


Figure 2-2: RGEMS network layout

Several Gigabit (802.3ab) [12] switches (both managed and unmanaged) form the basis for the wired portion of the network. Wireless N (802.11n – backwards compatible with 802.11b and 802.11g) [13] is used as the standard for wireless communication. Internet connectivity is provided by an asymmetric digital subscriber line (ADSL) through a router.

It is important to note that the RGEMS network has no connection to any CUT network (staff or student), be it wired or wireless. This is for security reasons and to protect the interests of RGEMS and CUT. Both entities have sensitive information and network resources which should be protected. The network seclusion also provides a more controlled environment for monitoring and testing purposes.

2.2.1 IT Core Network

As described in the previous section, the IT Core network enables RGEMS members to perform day-to-day tasks and provides access to the assembly system. As the network consists of wired, wireless and Internet connectivity, special care is taken to improve security. The router providing the Internet connectivity (modem) is not connected directly to the switches for the factory devices or switches for the servers and workstations. This is clearly visible in Figure 2-2. This separation is purposely done for increased security. In the unlikely event that unauthorised access is gained through the Internet. Only certain, non-critical, systems will be visible and no devices on the factory network will be accessible.

An Internet Protocol (IP) address is assigned to all servers, workstations and hardware devices through a hardware Dynamic Host Configuration Protocol (DHCP) server. Entries in a table containing the physical or Media Access Control (MAC) addresses of the hardware devices relative to the desired IP address are used to assign addresses to devices. This ensures that the devices maintain the same hardware address on the network between cold starts, but can easily be managed or altered by adjusting the settings of the DHCP server.

There are three servers in the RGEMS core network that provide certain functionalities and services to users and devices in the network. Figure 2-3 depicts the server cabinet with the three servers. These are (from top to bottom):

- **Database Server** – The database server provides all database functionalities required by the devices and users. The database software used is MySQL 5.1 [14], which was recently acquired by Sun Microsystems [15]. The community edition of the server software is used, which is available under the GNU General Public Licence (GPL) [16]. A

Redundant Array of Independent/Inexpensive Discs (RAID 10) configuration is used to ensure that the data is always secure in the event of a hard disk failure. This is facilitated via a dedicated RAID controller.

- **Application Server** – The application server acts as an OLE for Process Control (OPC) server and Proxy. OPC and how it is implemented in the test platform is further discussed in section 2.3. This server is also connected to two separate networks: first to the server and workstation network and, secondly, to the factory devices network.
- **Web Server** – The web server is the only device that has direct access to the Internet. An Internet proxy server running on the web server provides access, in a controlled manner, to all devices and users requiring web access in the network. The RGEMS website [10] is also hosted on this server. This server also acts as a Simple Mail Transfer Protocol (SMTP) relay and Post Office Protocol (POP3) proxy server for e-mail. It is important to note that the web server has two network connections: one is connected to the ADSL network and the other to the switch connected to the remainder of the servers and workstations. These connections are not bridged for security reasons. Therefore, the web server is effectively connected to two networks.

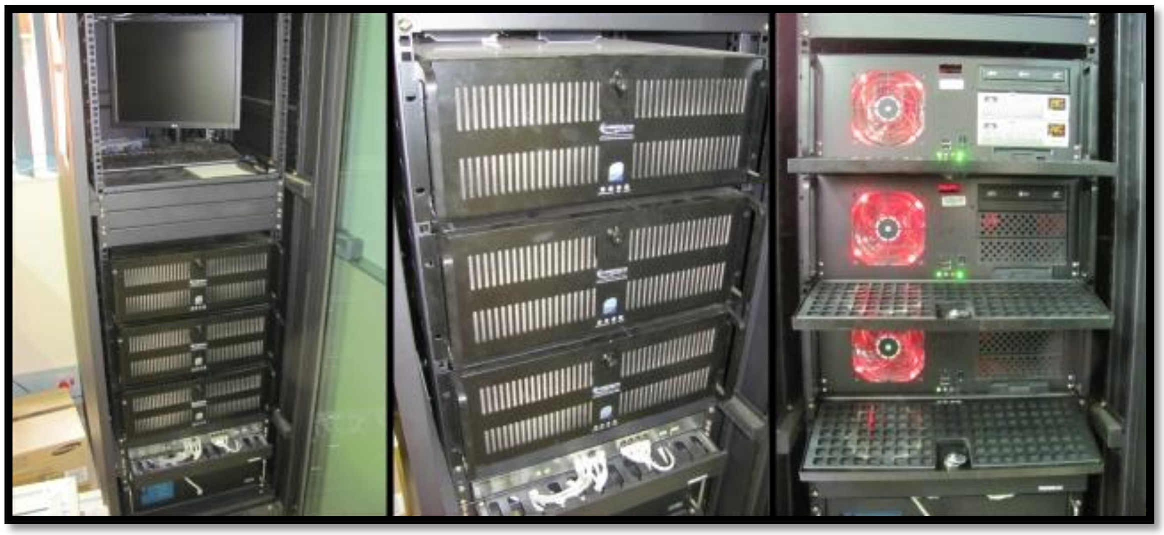


Figure 2-3: RGEMS server cabinet

A 3kVA Uninterruptible Power Supply (UPS), from which all the servers, routers and other miscellaneous devices in the server cabinet are powered, provides backup power in the event of a power failure. A four-terabyte (three-terabyte available storage) RAID 5 Network Attached Storage (NAS) serves as a file repository for RGEMS members as well as storage for daily server backups and camera footage. Each of the three servers is backed up for 90 days. Three IP cameras are set to record on motion-detection to the NAS for security purposes. Each camera has 100 gigabytes of storage allocated to it for recordings.

2.2.2 Factory Network

The factory network consists of all the hardware devices that form the assembly test platform. These devices are accessible through the Application Server via OPC. As there is no connection between the switches for the factory network and the IT core network, only OPC can be used to connect to the devices on the factory network. The OPC configuration is discussed in section 2.3. The devices can be categorised into three groups as follows:

- **Integrated OPC Devices** – These devices have their own OPC integration via a server, client or both. They are directly accessible via OPC and the Application Server that fulfils the role of OPC proxy when accessing these devices.
- **PLC Devices** – These devices (sensors, actuators, etc.) are not connected directly to the network, but are instead accessible through a Programmable Logic Controller (PLC) connected to the network. The PLC is accessed through OPC and, in turn, all the sensors and devices connected to the PLC.
- **Ethernet Encapsulation** – Some devices, such as Radio Frequency Identification (RFID) readers, do not possess native Ethernet connectivity. In this instance, an Ethernet encapsulation [17] device is used to encapsulate the RS232/RS485 serial communication and to make the device available over Ethernet. Migrating from serial communications to Ethernet simplifies the process of referencing the device through OPC.

2.3 OPC

2.3.1 Background and History

OLE for Process Control (OPC) was originally known as Object Linking and Embedding (OLE) for Process Control. OPC is a standards specification that resulted from the collaboration of an industrial automation industry task force in 1996. The task force consisted of a number of leading worldwide automation suppliers working in conjunction with Microsoft [18]. It was originally based on Microsoft's OLE Component Object Model (COM) and Distributed Component Object Model (DCOM) technologies and known simply as the "OPC Specification". The goal was to define a standard set of objects, interfaces and methods for use in process control and manufacturing automation applications to facilitate interoperability. As OPC defines a series of standards, the name was later changed to the Data Access (DA) specification and is now commonly known as Data Access.

The OPC Foundation was created after the initial release to maintain the standard. "The OPC Foundation is dedicated to ensuring interoperability in automation by creating and maintaining open specifications that standardize the communication of acquired process data, alarm and event records, historical data, and batch data to multi-vendor enterprise systems and between production devices [19]." Several standards have been added since then, and some names have changed. OPC is no longer an acronym and the technology is simply known as "OPC". This is the official stance of the OPC Foundation.

An OPC Server provides a method for different software packages to access data from process control devices, such as Programmable Logic Controllers (PLC) or Distributed Control Systems (DCS). These software packages then function as OPC Clients and the interaction is then client-server based. OPC enables the definition of a common interface that is written once and then reused as the need arises without redefining or redeveloping the interface. It is then possible for several applications to communicate with several OPC servers via a single OPC interface. Figure 2-4 illustrates this [20].

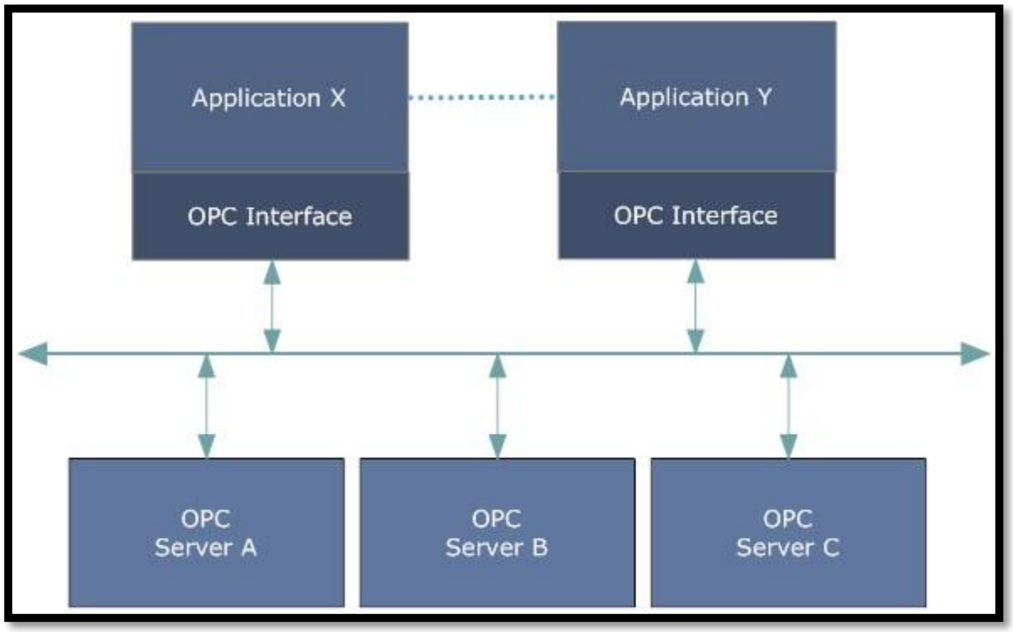


Figure 2-4: Applications working with various OPC servers

It is important to consider that OPC is a published specification, and therefore no company or organization has ownership of OPC. Any individual is allowed to develop an OPC server and membership of the OPC Foundation is not a prerequisite. The same applies to OPC integrators and it is the sole responsibility of every company to obtain certification of their products and verify the training of their system integrators.

2.3.2 Configuration Overview

OPC forms the communication platform for software systems to communicate with factory floor and hardware devices. These devices are connected to the factory network explained in section 2.2.2. The application server is the only server connected to both the factory and IT core networks, and is thus the server that hosts all OPC Servers used by any client applications to access devices via OPC. In an attempt to unify how devices are accessed via OPC, one OPC server is set up as an OPC proxy for connecting to other OPC Servers. This architecture allows one to tighten security for clients and servers, allowing clients to access all factory devices through one OPC Server. This reduces code and complexity as only one connection to one OPC server is required to access all devices. Figure 2-5 shows the OPC Server and client configuration.

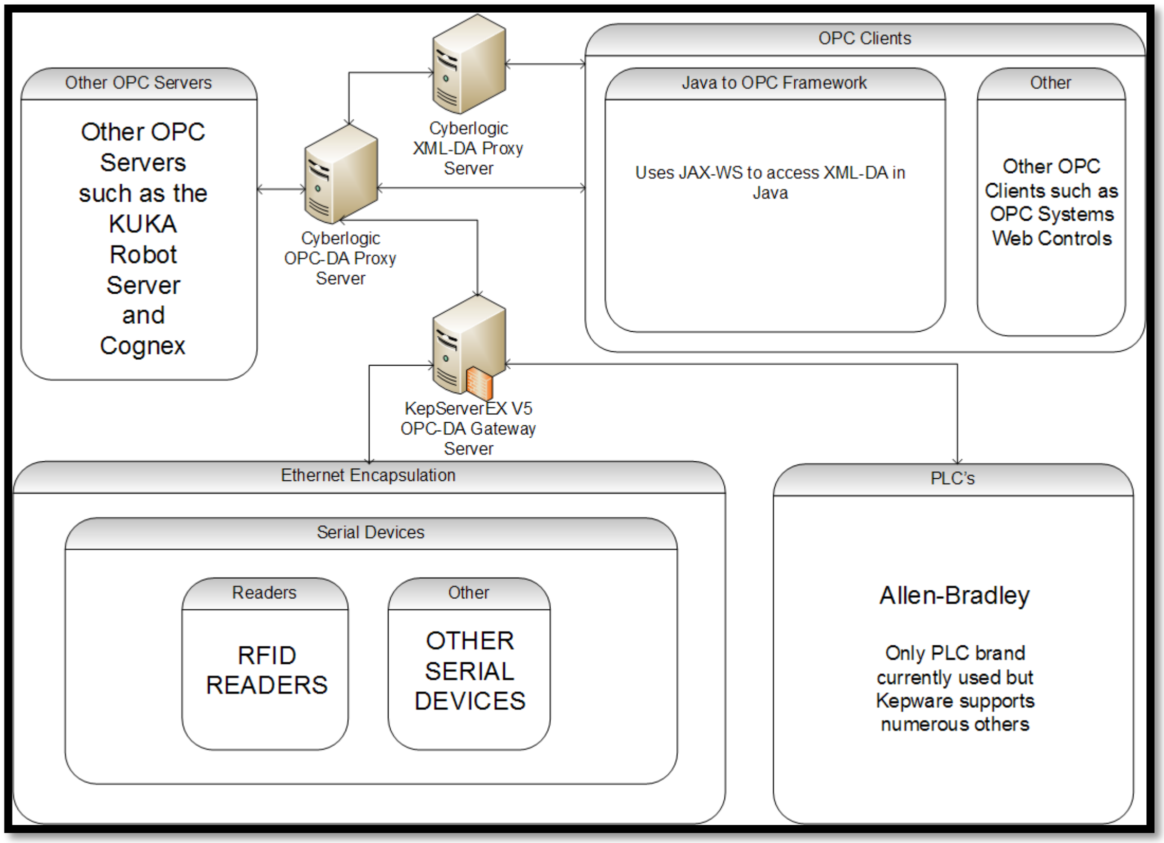

Other OPC Servers
Other OPC Servers such as the KUKA Robot Server and Cognex
Cyberlogic XML-DA Proxy Server
Cyberlogic OPC-DA Proxy Server
OPC Clients
Java to OPC Framework
Uses JAX-WS to access XML-DA in Java
Other
Other OPC Clients such as OPC Systems Web Controls
KepServerEX V5 OPC-DA Gateway Server
Ethernet Encapsulation
Serial Devices
Readers
RFID READERS
Other
OTHER SERIAL DEVICES
PLC's
Allen-Bradley
Only PLC brand currently used but Kepware supports numerous others

Figure 2-5: Proposed OPC configuration

Some of the devices such as cameras from Cognex [21] and robotic arms from Kuka [22] have their own OPC servers that are developed by the manufacturers themselves. These devices can easily be accessed with various OPC clients. Unfortunately not all devices have this feature and some steps must be followed to enable OPC connectivity. Kepware [23] supplies an OPC server that has drivers for numerous hardware devices. As shown in Figure 2-5, an Allen-Bradley [24] PLC from Rockwell Automation [25] is visible through the OPC driver provided by Kepware. Kepware also possesses a driver for Ethernet encapsulation to enable OPC connectivity for several serial devices such as RFID readers.

As mentioned previously, clients access only one OPC server. The server used is provided by Cyberlogic [26] and all connections to other OPC servers occur through this server. This server then assumes the role of OPC Proxy for all other servers required by clients. The Cyberlogic OPC proxy server connects to the device specific OPC servers provided by the manufacturer, such as those provided by Kuka and Cognex. In this situation the proxy OPC server assumes the role of client and the OPC server to which it is connected, fulfils the role of server. The proxy OPC server also connects to the Kepware server in the same manner. This security of the servers is configured to allow only connections from authorised clients. The Kepware and device specific-servers only allow connections from the Cyberlogic server, and these servers are invisible to other clients. By allowing clients to only connect to a single server (proxy), security complexity is reduced and overall management is simplified.

Figure 2-5 also shows how the Cyberlogic OPC server acts as a proxy for XML-DA communications. XML-DA allows for OPC communications via Extensible Markup Language (XML) and web services, subsequently eliminating the need for using DCOM. This allows normal DA clients and XML-DA clients to connect to the OPC proxy server and gain access to all factory devices.

2.4 Key Hardware Devices

With the industrial test platform modelled on an assembly system, the majority of the devices usually associated with an assembly system are present. These devices include, but are not limited to:

- Sensors for proximity determination (Figure 2-6 shows some examples), temperature measurements, speed and frequency calculations, etc.
- Switches for stopping and starting, as well as emergency stop switches.
- Pneumatic systems for suction cups, rams, etc.
- Conveyor belts for general purpose routing of parts and components.

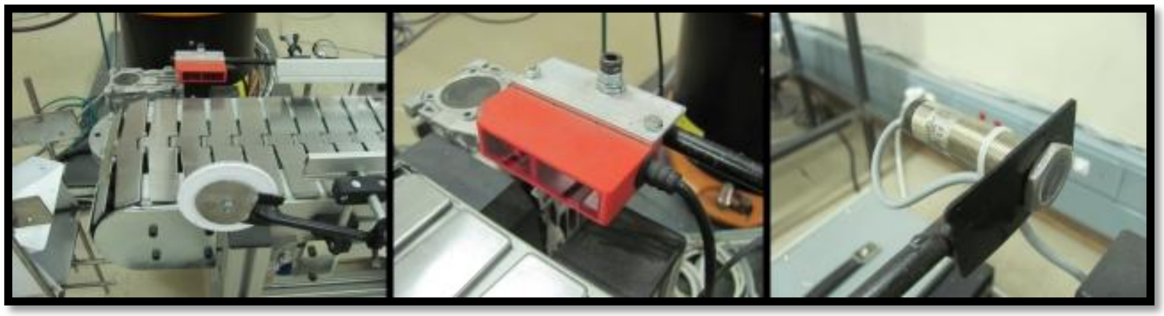


Figure 2-6: Proximity sensors

2.4.1 Programmable Logic Controllers

In addition to the more common devices present in the system, some more specialised devices are worth mentioning. A ML1500 programmable logic controller (PLC) from Allen-Bradley [24] (AB) forms the "heart" of the assembly system's control logic. Analogue and digital Input/Output (I/O) expansion cards provide additional slots for sensors, switches, etc. The PLC controls the majority of sensors, actuators and other end-effectors (pneumatic or mechanical). The PLC also has a Serial-to-Ethernet module which provides accessibility flexibility for controlling and programming the device. Figure 2-7 shows the PLC, expansion slots and some motor drives.

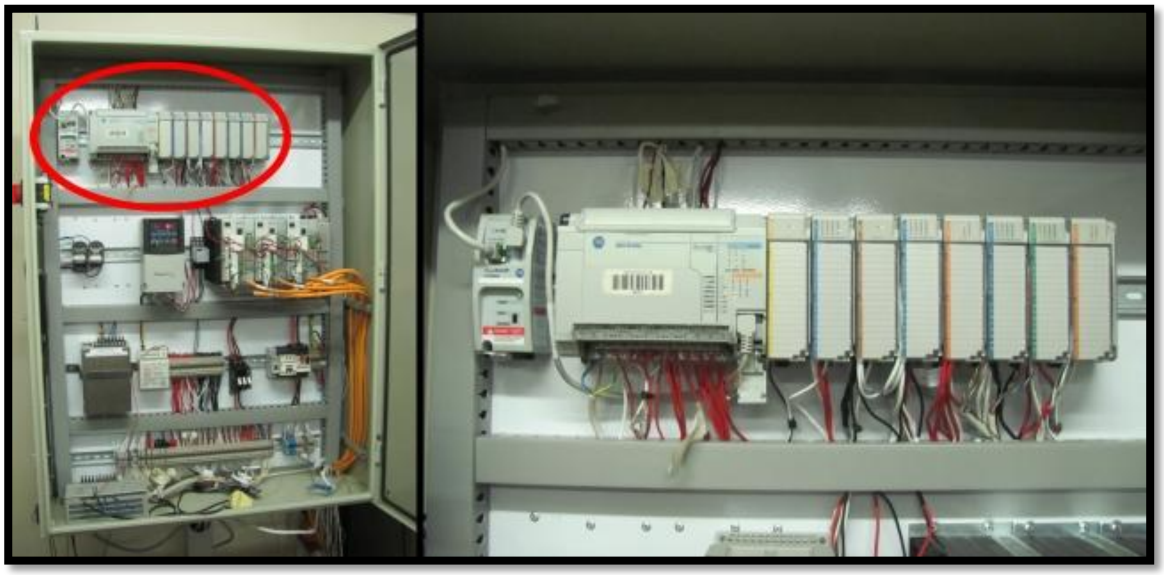


Figure 2-7: PLC with input and output expansion cards

2.4.2 Robotic Arms and Vision Systems

Two robotic arms aid in sorting and picking and placing of products. One is a KR 6-2 6-axis, industrial, low payload robot from Kuka [22]. This robot is situated between the first and the second conveyor in the assembly line. For component handling the robot makes use of a pneumatic 3-finger gripper and a suction cup. The work envelope also possesses a camera from Cognex [21], which is discussed in more depth later. Figure 2-8 depicts the Kuka robotic arm with the attached camera from Cognex, the gripper and suction cup. The second arm is a 3-axis Cartesian robot from Bosch Rexroth AG [27], illustrated in Figure 2-9. It fulfils similar functionalities as the Kuka robotic arm. Another camera is also positioned on the Cartesian robot for robot vision applications.

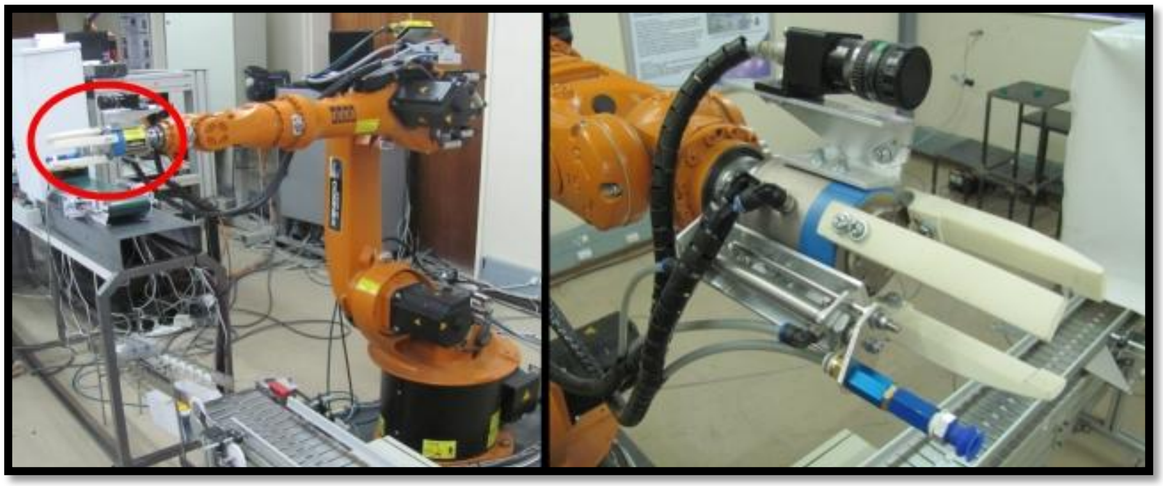


Figure 2-8: Kuka robotic arm with Cognex camera, gripper and suction cup

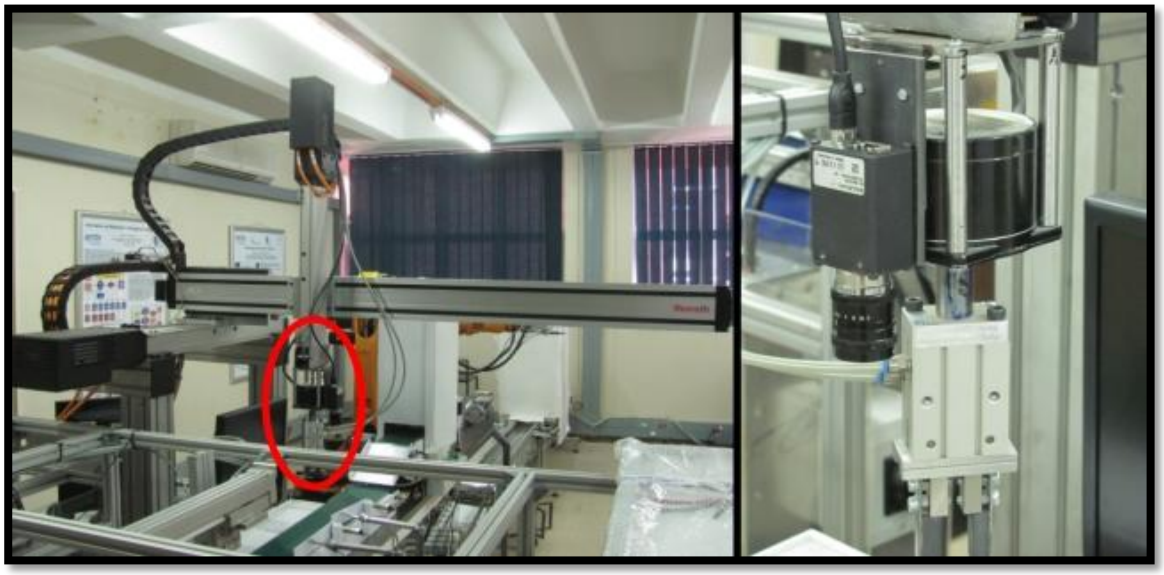


Figure 2-9: Cartesian robot and attached camera

In addition to the cameras mounted on the robotic arms, two other machine vision quality control systems also form part of the assembly platform. The first is mainly used for inspecting a 12-position palette of either grey objects or white objects. This system is illustrated in Figure 2-10. An image of the palette is acquired to determine whether objects are missing or misplaced. Once this has been established, one of the robotic arms can be used to replace missing or incorrect parts.

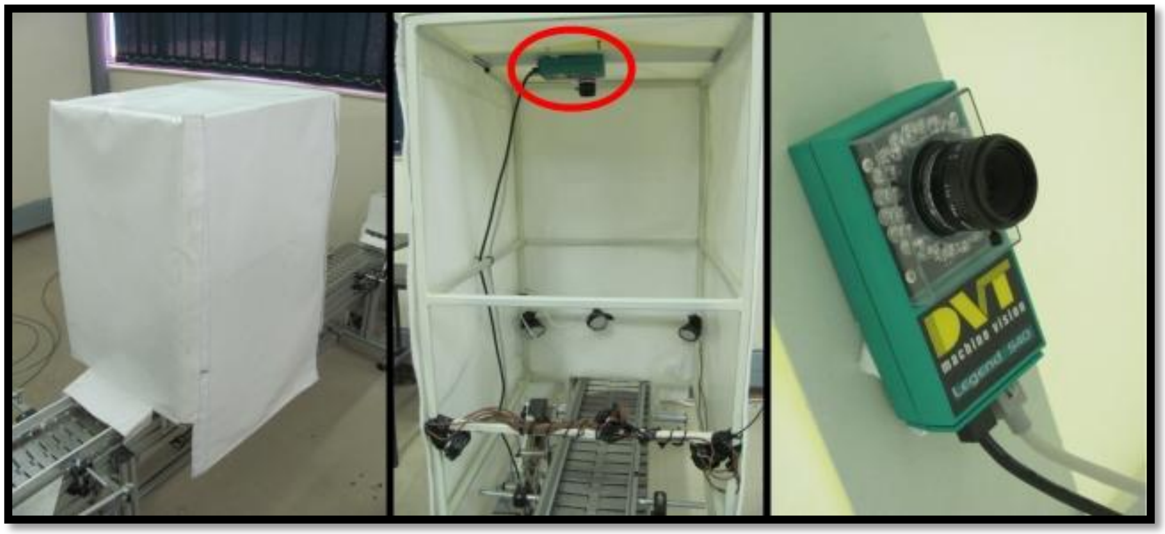


Figure 2-10: DVT machine vision inspection system

The second system makes use of Artificial Neural Networks for a machine vision grading system. It was originally trained to grade oranges or potatoes. It is mainly used in the assembly system to grade partially assembled parts. Figure 2-11 depicts the camera aided grading system. It is important to note that both vision systems are enclosed with their own illumination. The reason for this is that the 50 hertz frequency of the fluorescent lamps in the lab interferes with the image acquisition. The metallic finish of the conveyor also necessitates multiple lamps at different angles to eliminate reflection and shadow casting on the acquired image.

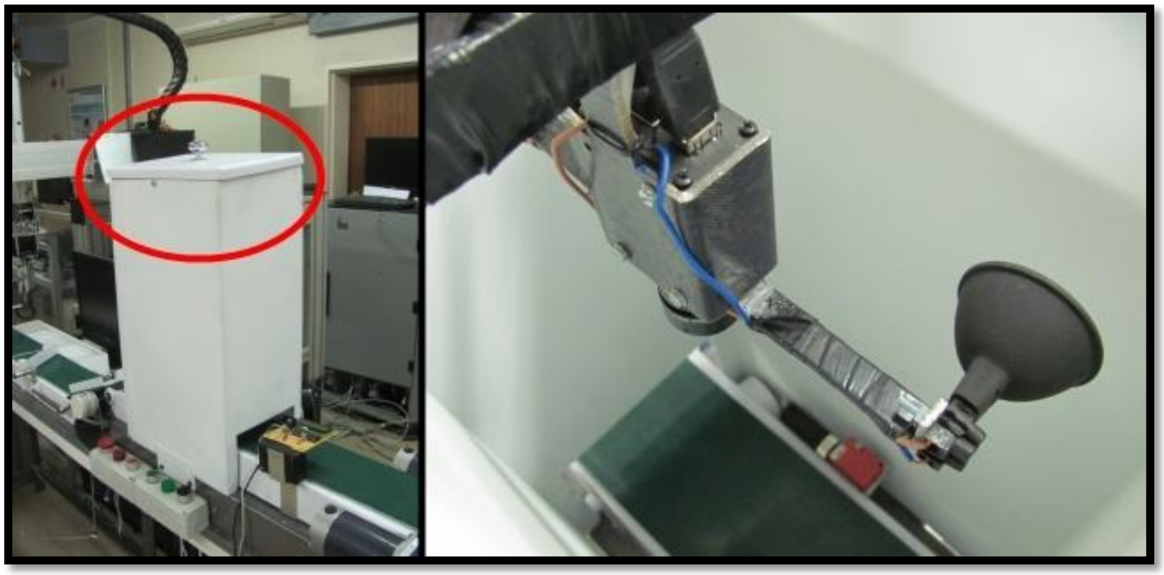


Figure 2-11: Artificial intelligence machine vision grading system

2.4.3 Radio Frequency Identification

Some Radio Frequency Identification (RFID) readers are also present on the system. Figure 2-12 illustrates one of the readers used for product tracking and identification purposes. Although the readers appear crude at the time of writing, special housings will be constructed for each reader once their respective applications have been finalised. RFID tags are embedded or attached to the components progressing through the system. RFID is a good addition to an assembly system for component tracking and identification, but cannot replace vision based quality control systems.

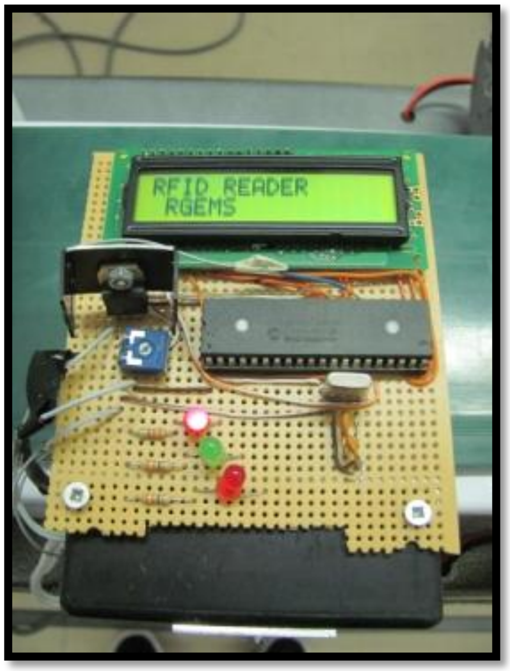


Figure 2-12: RFID reader

2.4.4 Serial-to-Ethernet Devices

Several Ethernet encapsulation devices also form part of the network. These devices provide access to serial devices via the OPC protocol. The devices are all accessed homogeneously via an IP address. The Ethernet encapsulation is done by Multenet [28] PocketPAD4 serial to Ethernet device servers. The PocketPAD4 comes in two variations: one provides four RS 232 ports and the other has three RS 232 ports, and one port can be configured to be RS 232, RS 422 or RS 485, depending on the requirement. The configuration of the port is done via jumper settings on the printed circuit board (PCB). Each of these ports can be individually assigned an IP address and port number for applications to connect to. The individual ports also have their own serial communication settings, such as baud rate, data bits, stop bits, parity, etc. The use of Ethernet encapsulation devices, such as the PocketPAD4, considerably simplifies the connection of serial devices to the industrial Ethernet network and OPC.

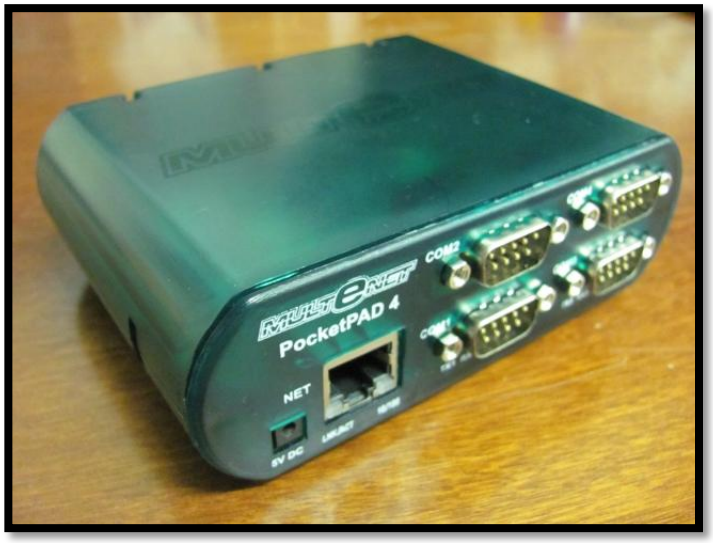


Figure 2-13: Multenet PocketPAD4 serial to Ethernet device server

2.4.5 Autonomous Guided Vehicles

An autonomous guided vehicle (AGV) allows components to enter and exit the system at arbitrary points. The AGV can be programmed to perform numerous tasks in the assembly system. It features a camera capturing images from a dome-shaped mirror to facilitate omni-directional vision. The image is processed and proximity sensors around the front of the AGV aid in determining distance from objects. The camera and proximity sensors aid in navigation and object avoidance.

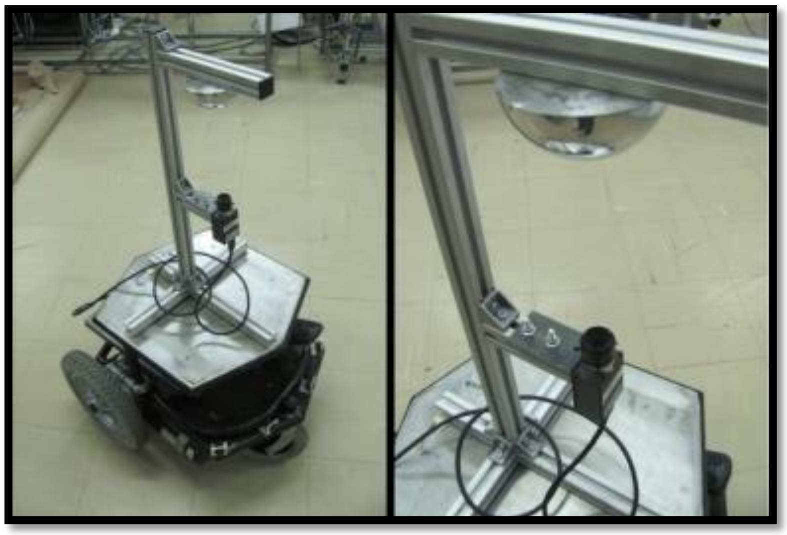


Figure 2-14: Autonomous guided vehicle

2.5 Conclusion

This chapter focussed on the component handling platform of RGEMS. Several vision-, robotic- and infrastructure related devices were discussed along with the constituent devices. Devices from several different vendors and manufacturers are present in the system. A discussion of the network infrastructure was followed by a brief introduction to and background information on the OPC communications protocol. It was also shown how OPC was selected as the universal communication protocol between the different hardware devices. All communication takes place via Ethernet and the components that make up the system each have their own address, be it directly via an IP address, or indirectly via a PLC or similar device. The network layout was discussed, with some mention of the servers and their specific function. Some of the devices can be potentially dangerous, thus special care was taken in securing the network and all the hardware components from unauthorised access.

Chapter 3 Maintenance Management

3.1 Introduction

The cost of operating an enterprise is influenced by critical factors such as the cost of maintaining complex industrial systems. It is estimated that 18-30% of this is wasted, which clearly highlighting the importance of optimising maintenance-related functions [29]. Failing to adequately perform maintenance can result in unplanned asset failure, and these failures can add many subsequent costs to the organisation. The majority of the maintenance is either reactive (fixing or replacing equipment as it fails) or blindly proactive (servicing equipment on a routine schedule after assuming a certain level of performance degradation has occurred) [2]. To the human eye machine failure often seems instantaneous, without any warning but the reality is that in most cases a measurable process of deterioration has taken place before a failure occurs.

Various technologies are available to aid in the monitoring of device degradation, such as, but not limited to vibration analysis, infrared thermography, oil analysis and tribology, ultrasonic sensing, motor current analysis, performance monitoring and visual inspection. "Many computerised maintenance management systems (CMMS) use condition monitoring alarm levels to trigger maintenance activities [29]". Predefined thresholds are used to compare incoming condition-based data. Exceeding the threshold level triggers an alarm to highlight the event. Maintenance personnel frequently struggle to cope with the vast number of alarms that are triggered on a daily basis. This is to a large extent due to the considerable number of events associated with the condition monitoring activity and the limitations on the setting of alarm levels. Prioritising which alarms to handle first can be troublesome if one considers the number of alarms and the fact that the logical assumption is that an alarm is legitimate until proven otherwise. This can be a difficult and time consuming procedure. The operator's experience is usually the prominent factor that dictates how efficiently alarms are handled.

3.2 E-maintenance

3.2.1 Introduction

The concept of e-maintenance refers to the integration of the information and communication technologies (ICT) within the maintenance strategy and/or plan. Muller *et al.* define e-maintenance as the "maintenance support which includes the resources, services and management necessary to enable proactive decision process execution. This support includes e-technologies (i.e. ICT, Web-based, tether-free, wireless, infotronic technologies) and e-maintenance activities (operations or processes) such as e-monitoring, e-diagnosis, e-prognosis, etc. [30]." Figure 3-1 illustrates the relationship between e-maintenance, e-manufacturing and e-business (adapted from [31]).

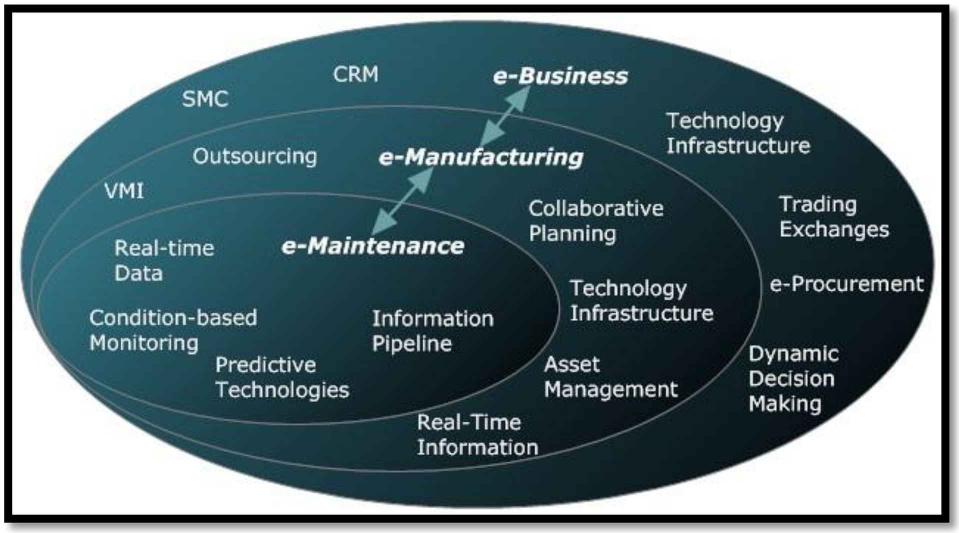


Figure 3-1: Relationship between e-maintenance, e-manufacturing and e-business

The emergence of e-maintenance can be attributed to two main factors:

- The utilisation of e-technologies; allowing for the increase of maintenance efficiency, speed, proactivity, etc.
- The need to integrate business performance, which enforces openness, integration, and collaboration with the services of the e-enterprise.

One of the greatest advantages of e-maintenance is the ability to connect field systems with remote expertise centres. This enables remote maintenance decision-making, which, in turn, provides added value to the top line, trims expenses and reduces waste. According to Han and Yang, an e-maintenance platform introduces an unprecedented level of transparency and efficiency into the entire industry, and it can be an adequate support of business process integration [32]. Implementing conventional maintenance and e-maintenance is compared in Figure 3-2 (adapted from [32]).

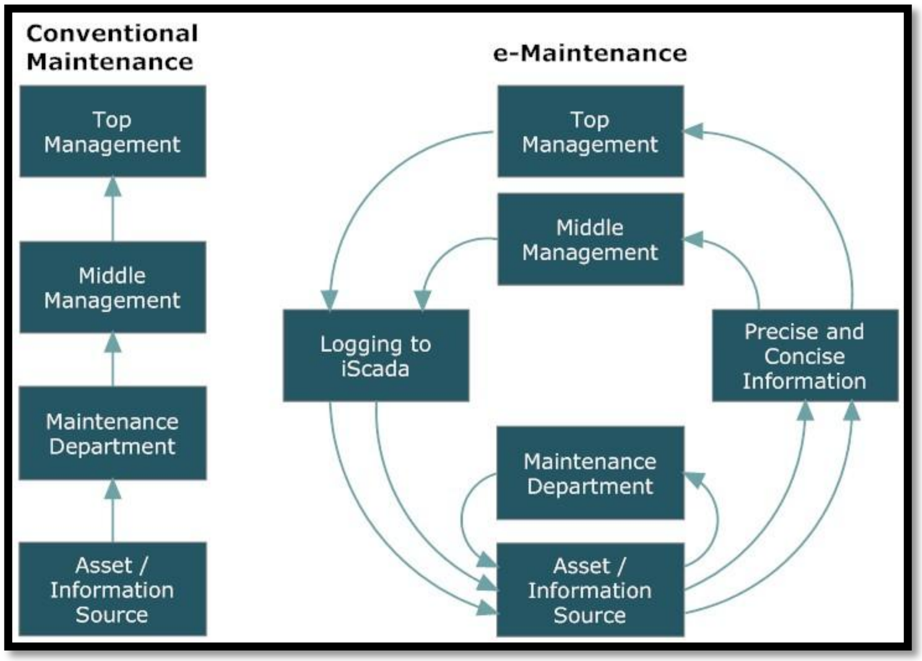


Figure 3-2: Implementing e-maintenance

E-maintenance can be viewed as more than the implementation of a maintenance strategy, a maintenance plan, or a maintenance type. It can be defined as a maintenance *revolutionary* change, rather than a maintenance *evolutionary* change. It can be compared to e-business a few years ago; the impact of e-maintenance is overestimated in the short run, but underestimated in the long run [32].

3.2.2 Artificial Intelligence Techniques

Several artificial intelligence (AI) approaches were evaluated, with a focus on implementations in the e-maintenance domain. This would assist in evaluating the feasibility of using AI techniques in e-maintenance, and more specifically, intelligent maintenance management. The aim of this study was to determine the viability of using the proposed AI technique based on previous implementations, as well as the current state of research in the applicable area.

3.2.2.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way in which biological nervous systems process information. This can be closely compared to the brain. The key element of this paradigm is the novel structure of the information-processing system. It is composed of a large number of highly interconnected processing elements, or neurons, working in unison to solve specific problems. ANNs, like people, learn by example [33]. The architecture of an AAN depends on:

- Number of inputs and outputs of the network
- Number of layers
- How the layers are connected to each other
- The transfer function of each layer
- Number of neurons in each layer

An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. As is the case with multi-agent systems, a neural network is a multitude of cooperative “entities” designed to perform a specific task. An important aspect of neural networks is that they have to learn by example, much like human beings do [34].

An important characteristic to take note of is that neural networks have layers: a perception or input layer, a number of processing layers, each with a transfer function to the next layer, and finally an output layer. In most cases, each node or neuron is linked to every node of the previous and next layer. There are cases where all the neurons are not interconnected in such a manner, but these are not that common. An example of a layered structure of a neural network is shown in Figure 3-3 (adapted from [34]).

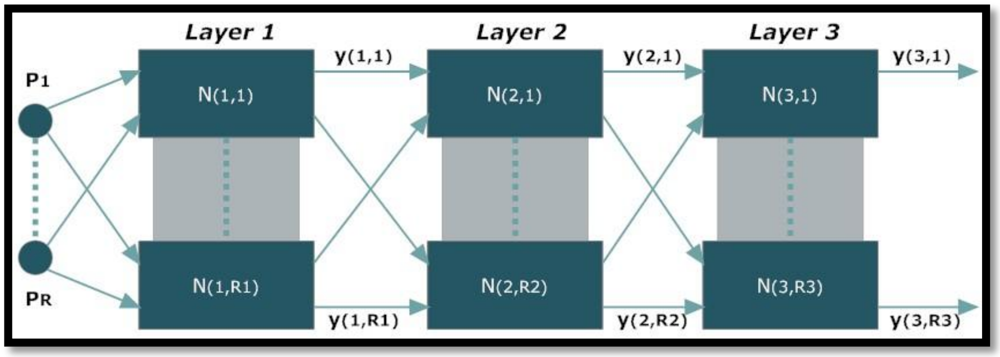


Figure 3-3: Feed-forward architecture with three layers

ANNs have been used for signal processing, data analysis, fault identification, and degradation prediction. They have also been used along with other advanced techniques, such as expert systems, fuzzy logic systems and genetic algorithms to correctly interpret fault data. Han and Yang propose an e-maintenance system that incorporates advanced techniques, such as ANN, for the maintenance of induction motors [32]. The focus lies with induction motors because they are usually the prime movers in an industrial plant. Failure of these motors can greatly affect productivity and even cause the shutdown of the complete production line.

Manikandan and Devarajan propose using an AAN in the classification of unknown faults in linear dynamical systems. The neural network is trained using a back propagation algorithm. This enables the bias and weights of the ANN to be automatically updated, based on the difference between the actual output of the network and the target output. A number of known faulty conditions are used to train the network. Once the ANN is trained, it does fault classification of unknown fault conditions. Figure 3-4 (adapted from [35]) presents a schematic of the pre-trained ANN for fault classification.

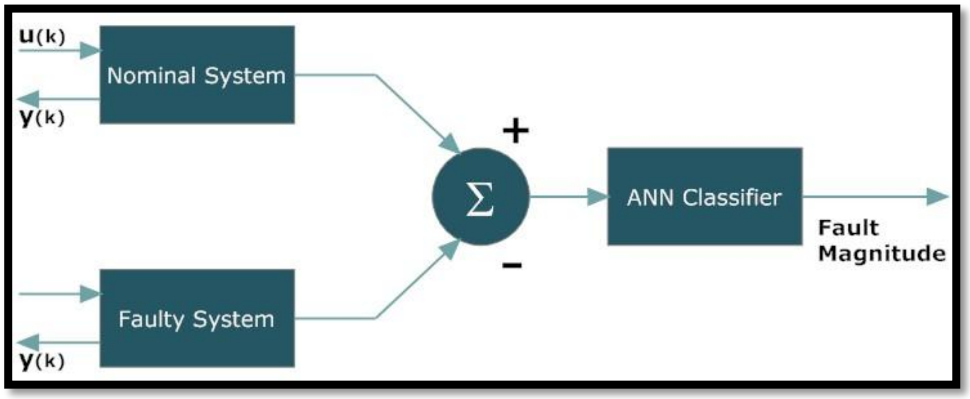


Figure 3-4: Extracting fault residues

3.2.2.2 *Expert Systems*

Expert systems (ES) are decision-making and problem solving tools that represent the knowledge of an expert in a particular field. "An expert system consists of computer software programs that emulate or clone the reasoning of a human expert in a problem domain. A problem domain is a special subject area in which the expert can solve problems very well [36]." ES bring expert knowledge directly to the fingertips of a novice. This knowledge is then made available to a novice user via the expert system. In a sense, the expert system is the middleman between the novice and the expert. Figure 3-5 shows the key components of an ES (adapted from [36]). This allows the novice user full access to the knowledge of the specialist, without directly consulting him/her.

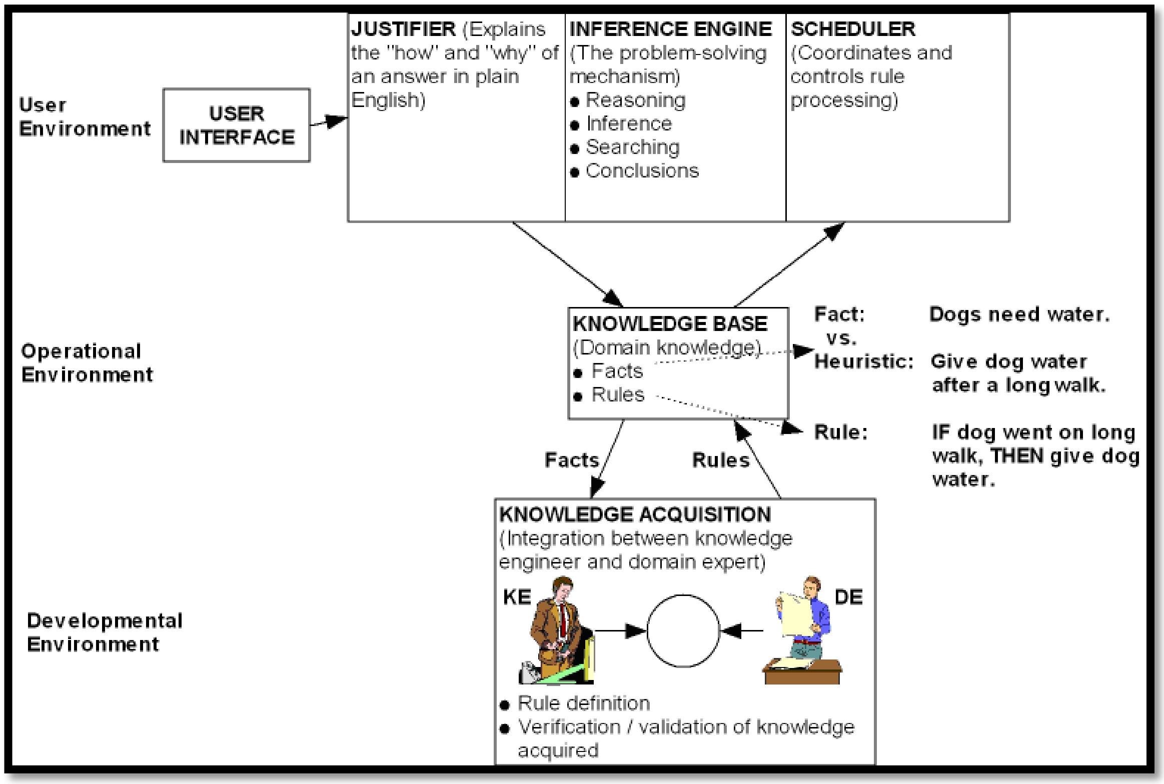

User Environment
USER INTERFACE
JUSTIFIER (Explains the "how" and "why" of an answer in plain English)
INFERENCE ENGINE (The problem-solving mechanism)
● Reasoning
● Inference
● Searching
● Conclusions
SCHEDULER (Coordinates and controls rule processing)
Operational Environment
KNOWLEDGE BASE (Domain knowledge)
● Facts
● Rules
Fact: Dogs need water.
vs.
Heuristic: Give dog water after a long walk.
Rule: IF dog went on long walk, THEN give dog water.
Facts
Rules
Developmental Environment
KNOWLEDGE ACQUISITION (Integration between knowledge engineer and domain expert)
KE
DE
● Rule definition
● Verification / validation of knowledge acquired

Figure 3-5: Key components of an expert system

Research on fault diagnosis in the Artificial Intelligence community was initially focused on ES or knowledge-based approaches. This was based on applying heuristics to explicitly associate symptoms with fault hypothesis. The short comings of a pure ES approach led to the development of model-based approaches [35]. The majority of the systems using knowledge-based methods work with models of the system, despite the presence of faults. This implies that, theoretically, there needs to be a model for each possible fault which is impossible for most real-world situations. Some faults may damage the system; others may be dangerous or impossible to simulate.

An ES is only as good as the knowledge represented in the computer; the knowledge is not easily defined, and experts are not always available. Another demerit of ordinary, rule-based ES is that they cannot handle new situations not covered explicitly in their knowledge bases. Adapting to changes in the system poses a problem and calls for a considerable amount of reconfiguration work [36]. These points raise important questions concerning the applicability of expert systems in an intelligent maintenance management system, and should be carefully considered.

3.2.2.3 Fuzzy Logic Systems

Fuzzy logic systems (FLS) are aimed at formalising modes of reasoning which are approximate rather than exact and are much more general than traditional logical systems. The greater generality of fuzzy logic facilitates the solving of complex problems in the realms of search, question-answering decision and control [37]. Fuzzy logic systems also provide a foundation for developing tools for dealing with natural languages and knowledge representation. FLS can be applied in a variety of domains such as industrial systems, automotive, decision analysis, medicine, geology, pattern recognition, robotics, aircraft control, cruise control, to name but a few [38], [39]. FLS can be merged with other techniques such as expert systems, as discussed in [40], to make them more robust and practical.

Induction machine stator faults can be identified with the aid of FLS [40]. For fault diagnostics, the machine condition is described by linguistic variables. Fuzzy subsets and the corresponding membership functions describe stator current amplitude. A knowledge base is constructed consisting of rules and databases. This knowledge base aids the fuzzy inference. Figure 3-6 (adapted from [40]) shows the block diagram of FLS for fault diagnostics of induction machines. Fuzzy rules and membership functions are constructed by observing the data set.

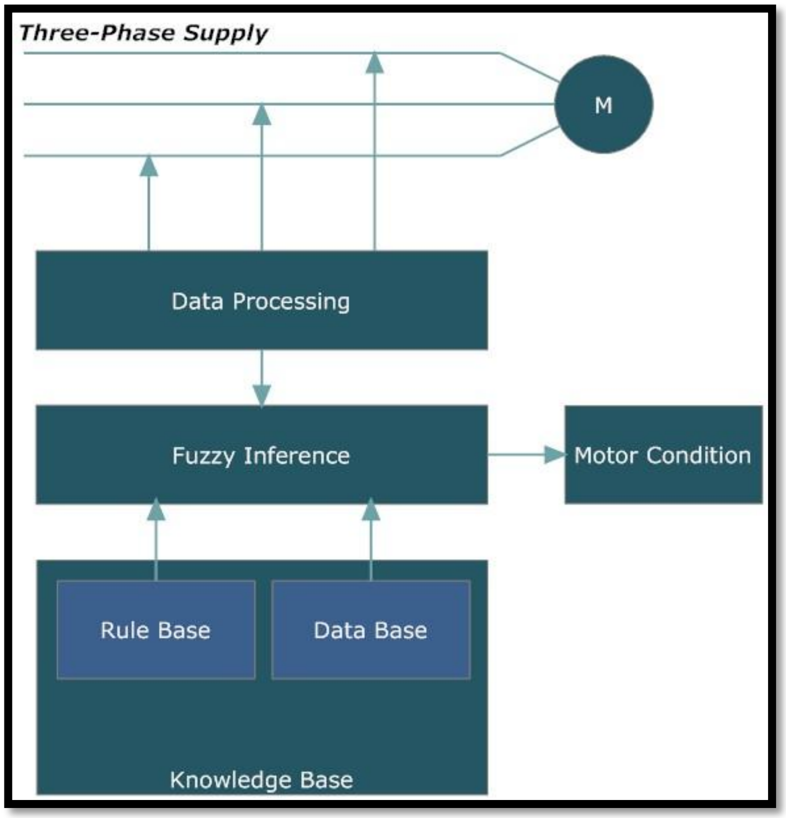


Figure 3-6: Fuzzy logic based induction machine diagnostics

According to Moore and Starr [29], FLS can be applied where rules or heuristics are available. They propose a system where maintenance jobs arising from condition-based maintenance are automatically prioritised using a strategy called cost-based criticality. The strategy does not change the strategic plan for maintenance; it only addresses prioritisation.

In practice many engineers prefer auditable decision-making of FLS as opposed to techniques such as Artificial Neural Networks (ANN). "Any method used must generate trust in its users or it will be shelved [29]." Some suggest merging FLS and ANN to produce a more robust and flexible solution. By merging FLS and ANN techniques, a neural-fuzzy fault detector is obtained, as proposed in [40]. This fault detector learns the stator faults and the conditions under which they occur through an inexperienced and non-invasive procedure. The neural-fuzzy system is ANN structured using FLS principles. This enables the system to provide qualitative descriptions of the machine condition and the fault detection process.

3.2.2.4 Genetic Algorithms

Genetic algorithms (GA) can be defined as a heuristic search method that imitates the principles of biological adaptation. This is based on the mechanics of the Darwinian survival-of-the-fittest theory [41]. According to Magnus [42], “genetic algorithms use the biological metaphor of an evolving population to explore large-dimensional problem spaces.”

The input to a GA is a set of potential solutions to the problem at hand. These solutions are encoded in some fashion and a metric, called the fitness function, is used to quantitatively evaluate each candidate. The candidates may be solutions already known to work, to which the GA can be applied for improvement, but more often they are generated at random. The GA then evaluates each candidate according to the fitness function. In a pool of randomly generated candidates, most will not work at all, and these are deleted. However, purely by chance, a few may hold promise - they may show activity, even if only weak and imperfect activity, toward solving the problem [43]. These promising candidates are kept and allowed to reproduce. Multiple copies are made, introducing random changes during the copying process. Figure 3-7 (adapted from [43]) illustrates some of the random changes that can be made during the copy process and the effects these changes can have. The digital offspring, obtained from reproduction and copying, then go on to the next generation, forming a new pool of candidate solutions, and are subjected to a second round of fitness evaluation. The candidates that show a reduced or exactly the same fitness function are deleted. The remaining candidates are then better possible solutions to the problem at hand and repeating this process a hundred or thousand times can provide very good solutions to this problem.

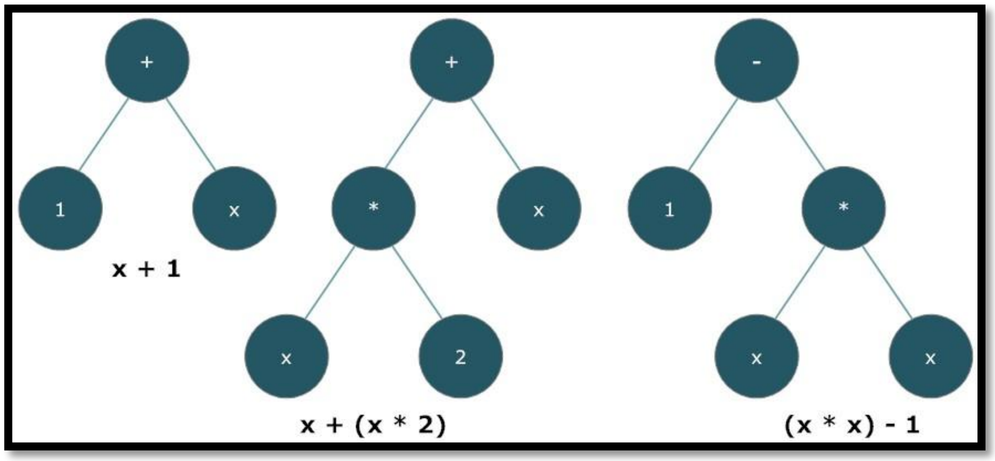


Figure 3-7: Three simple program trees of the kind normally used in genetic programming

GA have been applied to a broad variety of problems, which include but are not limited to acoustics, chemistry, electrical engineering, robotics, financial markets and routing [43]. In one instance a genetic algorithm was used to create a voice recognition circuit using a field-programmable gate array (FPGA). The solution that the GA came up with used only 37 gates, of which five were only connected to the power supply and not the rest of the circuit. These gates, however, were required for the operation of the circuit. This implies that the GA exploited the subtle electromagnetic effect of the cells to come up with the solution [43].

The random nature of GA allows for the consideration of a vast number of possible solutions, some of which contradict popular beliefs and expert opinions. However, these solutions do result in some of the most interesting solutions to everyday problems imaginable.

3.2.2.5 Multi-Agent Systems

An agent is defined as a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its designed objectives [6]. A multi-agent system (MAS) consists of several agents, each having a specialised functionality in the agent “community”. This entire system is designed to fulfil a common purpose. Intelligent software agents can be integrated into the performance assessment, prediction and diagnosis functions of the maintenance application, as stated in [1]. Several standards for MAS have emerged in recent years. The Foundation for Intelligent Physical Agents (FIPA) is an IEEE Computer Society standards organisation that promotes agent-based technology and the interoperability of its standards with other technologies [44].

Solutions based on intelligent software agents or MAS have been used to solve problems in a wide variety of domains such as business process management systems (ADEPT), personal information agents (MAXIMS), information retrieval systems, electronic commerce, health care, human-computer interfaces, and even social simulations [6], [45]. Pirttioja et al. proposed an agent-based architecture for information handling. The main reason for this work was to highlight the possibility of applying software agents to help solve the problem of information overload for the human operator [46]. This agent-augmented architecture which is further discussed in related work by the same authors [37] is illustrated in Figure 3-8.

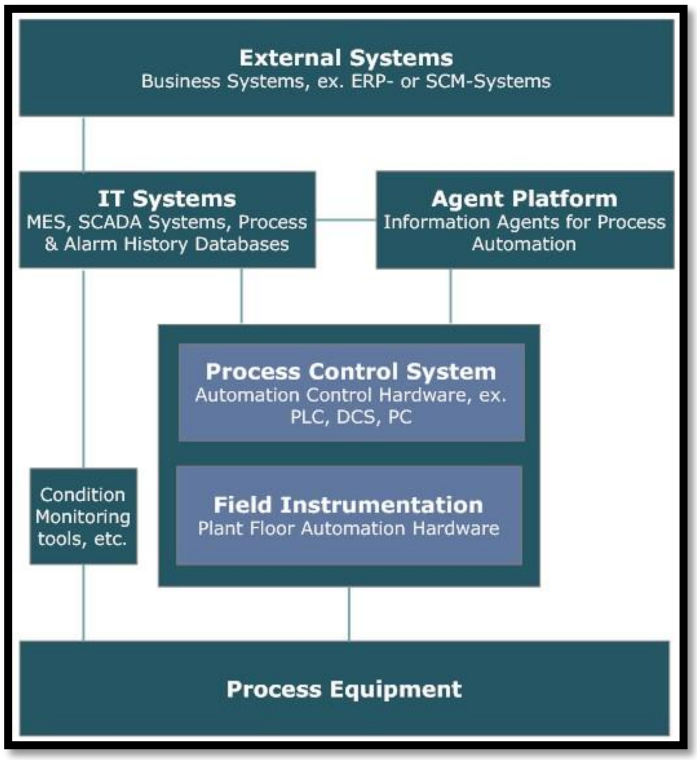

External Systems
Business Systems, ex. ERP- or SCM-Systems
IT Systems
MES, SCADA Systems, Process & Alarm History Databases
Agent Platform
Information Agents for Process Automation
Process Control System
Automation Control Hardware, ex. PLC, DCS, PC
Field Instrumentation
Plant Floor Automation Hardware
Condition Monitoring tools, etc.
Process Equipment

Figure 3-8: Agent-augmented architecture

The MAS discussed in this paper is designed to be an extension of the ordinary process automation system. "It is a separate layer on top of the physical control system, at the same level with present-day supervisory control software, and below plant management and business systems [37]." Both human beings and computational systems can utilise the information access services. The agent system can generate a user interface to enable interaction with human operators; or by using common process automation Supervisory Control and Data Acquisition (SCADA) software. This architecture also provides functionality to format and relay information from the process control system to external computational systems; greatly increasing the potential for enterprise level integration.

One distinct demerit of MAS is the lack of suitable ontological definitions and there is no evidence to show that sufficient effort has been made towards creating semantically rich ontologies for the industrial automation and maintenance domain [37]. This is an important point to consider when using MAS for any implementation.

3.2.3 Examples of Agents in e-Maintenance

This section examines some of the Multi-Agent Systems-based approaches in e-maintenance. Zhang proposes a framework of multi-agent systems to realise knowledge management in terms of information [47]. The author models all the information sources in the industrial automation system as agents. The relationships between these sources are emphasised in the context of maintenance, especially proactive maintenance.

Yu *et al.* discuss a combination of modern information-processing and communication tools, commonly referred to as Tele-service, and how this approach offers the technical support required to implement remote service maintenance [48]. The authors conclude that this technical support is insufficient to deal with remote maintenance decision-making which requires not only informational exchanges between customers and suppliers but also co-operation and negotiation based on the sharing of different complementary and/or contradictory knowledge. An evolution is proposed from Tele-service to e-service and e-maintenance, in particular where the maintenance decision-making results from collaboration of maintenance processes and experts to form a Distributed Artificial Intelligence (DAI) environment. For this purpose, the authors introduce a Problem-Oriented Multi-

Agent-Based E-Service System (POMAESS). The negotiation protocol and the case-based-reasoning decision support function within this system focuses on service maintenance problem-solving.

Passadore and Pezzuto present a European project named E-Support, which is aimed at the maintenance companies with technicians in the field and away from the central headquarters [49]. The main goal of E-Support is to help field engineers and technicians access the knowledge repository of the company. They connect to remote servers via mobile devices in order to gain access to information regarding vendors, customers, plants, parts and download technical documents. The authors suggest an implementation based on a multi-agent platform running agents on mobile devices and server agents that provide the services.

Naedele *et al.* motivate the use of a multi-agent system (MAS) based asset management application for manufacturing plants [50]. The authors provide some examples of situations where collaboration of intelligent software agents may give better results compared to fully deterministic systems. A layered reference model, previously used to compare various research approaches and systems in the area of MAS for plant asset management, is mapped to the ISO 13374 standard prescribing high-level system architecture for condition monitoring systems. They go on to show that both reference models are compatible and complement each other well, and suggest adding an agent communication language to the (not normative) list of communication mechanisms between the different function blocks in the standard.

3.3 Maintenance Prognosis

3.3.1 Introduction

This section considers four prominent techniques or algorithms used in maintenance to determine device prognosis and/or aid in the maintenance process. A brief overview of each method is followed by a discussion of some proposed applications.

3.3.2 Bayesian Networks

“Bayesian networks (BN) are directed acyclic graphs that are constructed by a set of variables coupled with a set of directed edges between the variables [51].” Bayesian networks can be applied to reason between variables via conditional probabilities. A Bayesian network is a graphical model that carries probabilistic information about its nodes. It has a set of variables $U = (A_1, A_2, \cdots, A_n)$ and a network structure represented by a graph of conditional dependencies among the variables in U. A conditional dependency connects a child variable with a set of parent variables. This dependency is represented by a table of conditional distributions of the child variable, given each combination of the values of the parent variables. The properties of BNs can be summarised as follows [51]:

- BNs employ a set of variables and a set of directed edges between variables.
- Each variable contains a finite set of mutually exclusive states.
- The variables coupled with the directed edges construct a directed acyclic graph (DAG).
- Each variable A with parents $B_1, B_2, \cdots, B_n$ has a conditional probability table $P(A|B_1, B_2, \cdots, B_n)$ associated with it.

Van Noortwijk *et al.* propose using a BN to determine a new failure model for hydraulic structures in structural engineering [52]. The failure model employed in this work is based on observable deterioration characteristics. The majority of failure models are based on lifetime distributions or Markovian deterioration. These approaches give rise to problems in practice, such as the difficulty in getting data for either a lifetime distribution or the transition probabilities of a Markov chain.

Juang and Anderson propose a Bayesian theoretic approach to determine an optimal adaptive preventive maintenance policy with minimal repair [53]. The mathematical formulas of the expected cost per unit time are obtained by incorporating minimal repair, major repair, planned replacement, unplanned replacement and periodic scheduled maintenance in the model. When the failure density has a Weibull distribution [54] with uncertain parameters, a Bayesian approach is established to formally express and update the uncertain parameters for determining an optimal adaptive preventive maintenance policy. This model incorporates five possible maintenance actions: minimal repair, major repair, planned replacement, unplanned replacement and periodic scheduled maintenance. Scheduled maintenance is carried out as soon as *T* time units have elapsed since the last major maintenance action, which includes a system replacement, major repair or previous scheduled maintenance. At the *Nth* scheduled maintenance, the system is replaced rather than maintained. Furthermore, when the system fails before age *T*, it either receives a major repair (or replaced after $(N - 1)$ maintenance events) or is minimally repaired depending on the random repair cost of the failure. The objective is to determine the optimal plan (in terms of *N* and *T*) which minimises expected cost per unit of time. When the failure density has a Weibull distribution, a Bayesian approach is proposed to derive the optimal adaptive preventive maintenance policy with minimal repair such that the expected cost per unit time is minimised.

Morales et al. propose a Bayesian approach based on a queuing system with a set of machines which are subject to failures [55]. The machines are maintained by repair crews and can be replaced by spare machines when broken down. Several Bayesian performance criteria for the system design are discussed, each of them guaranteeing a required operational capacity in a different sense. The decision as to which criterion to use depends on the special characteristics of the system and the particular objectives of the operator. Predictive criteria imply a satisfactory expected performance. Posterior criteria are more demanding and require obtaining the desired capacity with great confidence. The authors note that the predictive approach is conceptually easier and more comprehensible, but posterior criteria can be more appropriate if the reliability requirements are very strong. A Bayesian decision approach would provide its full benefits if it is possible to specify costs, not only for spares and repair crews, but also penalties when the operational capacity requirements are not met.

The work conducted by Oukellou et al. exploits the ability of Bayesian networks to model complex systems [56]. The study investigates the possibility of using them to learn the spatial relationship between singular points. A global diagnosis system is developed for broken rail detection that combines two information sources: local- and global data. The local data is provided by a dedicated sensor that detects any modification in the geometry and/or electromagnetic characteristics of the rail. The global data is provided by track-setting rules that constrain the position of singular points of the track while real defect are not spatially dependant. The global statistical model uses a Bayesian network to take account of these spatial dependencies and independencies. This study investigates the possibility of combining these two approaches in order to distinguish real defects from singular points under difficult operating conditions.

Sahin *et al.* present a fault diagnosis system for airplane engines using Bayesian networks (BN) and distributed particle swarm optimization (PSO) [51]. This implementation has the advantages of being general, robust, and scalable as opposed to more traditional BN-based fault diagnosis methods. Neither expert knowledge nor node ordering is necessary prior to the Bayesian network discovery. The raw datasets obtained from airplane engines during actual flights are pre-processed using an equal frequency binning histogram and used to generate Bayesian networks for engine fault diagnosis. The authors studied the performance of the distributed PSO algorithm and generated a BN that can detect faults in the test data successfully. Main components of the implementation are data pre-processing, a search algorithm for finding the Bayesian network that represents the data best using particle swarm optimisation (PSO), and a parallel computing implementation of the PSO algorithm using the message passing interface on a Linux computer cluster.

3.3.3 Kalman Filters

Among the numerous techniques that have been investigated to solve a diagnosis problem, the popular Kalman Filter (KF) has received special attention. This recursive, minimum-variance algorithm has proven its capability to track gradual deterioration such as wear with good accuracy [57]. Indeed, the Kalman filter embeds a transition model that describes a *relatively slow* evolution of the health parameters. On the other hand, the response of the Kalman filter to short-time-scale variations in the condition of engine is either a long delay in recognising the fault, or/and a spread of the estimated fault over several components which is termed the *smearing* effect.

“Most estimation methods which have been recently developed are computationally unmanageable, and require special assumptions on the form of the process and observer models which should be satisfied in practice [58].” For these reasons, the families of Kalman Filter, and its extensions are still the most widely used estimation algorithms. Kalman filtering is proposed for estimating the state vectors of a discrete-time nonlinear system.

Wu *et al.*, the authors of [59], apply an adaptive order tracking fault diagnosis technique based on a recursive Kalman filtering algorithm. In this study, a high-resolution order tracking method with an adaptive Kalman filter is used to diagnose the fault in a gear set and damaged engine turbocharger wheel blades. The adaptive Kalman filtering algorithm can overcome some of the problems encountered in conventional methods. The problem is treated as the tracking of frequency-varying band-pass signals. Ordered amplitudes can be calculated with high resolution after experimental implementation. Experiments are also carried out to evaluate the proposed system in gear-set defect diagnosis and engine turbocharger wheel blades damaged under various conditions. The experimental results indicate that the proposed algorithm is effective in fault diagnosis of both cases.

Yang presents the experimental results of a failure prediction method for preventative maintenance by state estimation using the Kalman filter on a DC motor [60]. The rotating speed of the motor was uninterruptedly measured for a period of two and a half months. The measured data was used to execute Kalman prediction and to verify the accuracy of the prediction. The resultant prediction errors were acceptable. Furthermore, it was found that the shorter the increment time for every step used in the Kalman prediction, the higher the prediction accuracy it achieved. Failure can be prevented in time so as to promote reliability by state estimation for predictive maintenance using the Kalman filter.

Borguet and Léonard present and compare two adaptive algorithms for engine health monitoring [57]. Both combine a Kalman filter, which provides accurate estimation of the health condition for long-time-scale deterioration (such as engine wear), and an adaptive component which monitors the residuals and detects abrupt changes in the health condition. On one hand, a covariance matching scheme performs an on-line tuning of the process noise variances. On the other hand, a generalised likelihood ratio test detects and estimates rapid changes in the engine condition. Interestingly, the approach employed by the authors does not require the set-up of a pre-defined bank of accidental faults. The methodology could also be extended to handle system faults such as stuck bleed valves or mistuned variable stator vanes. The implementation of the adaptive algorithms is straightforward and involves only basic matrix operations.

The work by Karami *et al.* presents a model-based fault detection approach for induction motors [58]. A filtering technique using Unscented Kalman Filter (UKF) and Extended Kalman Filter (EKF) is utilised as a state estimation tool for on-line detection of broken bars in induction motors based on rotor parameter value estimation from stator current and voltage processing. The hypothesis on which the detection is based is that the failure events are detected by jumps in the estimated parameter values of the model. Both UKF and EKF are used to estimate the value of rotor resistance. Upon breaking a bar the estimated rotor resistance is increased instantly, thus providing two values of resistance after and before bar breakage. In order to compare the estimation performance of the EKF and UKF, both observers are designed for the same motor model and run with the same covariance matrices under the same conditions. Computer simulations are carried out for a squirrel cage induction motor. The results show the superiority of UKF over EKF in nonlinear system (such as induction motors) as it provides better estimates for rotor fault detection.

Shao and Mechefske developed a fully automatic and robust vibration monitoring system for gearboxes [61]. The primary objective was to determine how to exclude the effects of variable load conditions. The proposed technique features a number of appealing advantages, including extended Kalman filter-based time-varying autoregressive modelling, automatic autoregressive model order selection with the aid of a non-paired two-sample Satterthwaite's *t*-test, a highly effective and robust condition indicator, and an automatic alert generating mechanism for incipient gear faults with the aid of a Wilcoxon rank-sum test. Two sets of entire lifetime gearbox vibration monitoring data with distinct variable load conditions were used for experimental validation. The proposed condition indicator was compared with other well-known and/or recently proposed condition indicators. The results demonstrated excellent performance of the proposed technique in four aspects: the effectiveness of identifying the optimum model order, a minimum number of false alerts, constant behaviour under variable load conditions, and to some extent an early alert for incipient gear faults. Furthermore, the proposed condition indicator can be directly employed by condition-based maintenance programmes as a condition covariate for operational maintenance decision analysis.

3.3.4 Support Vector Machines

Support vector machines (SVM) are a class of machine learning algorithms that was originally developed for classifying data from two different classes [62]. The basic principle of a binary support vector classifier is as follows: given a dataset comprising data from two different categories, it constructs an optimal linear classifier in the form of a hyper plane which has the maximum margin. The margin of a classifier can be defined as the width up to which the boundary can be extended on both sides before it hits one of the data points. The points onto which this margin hits are called the *Support Vectors*. In the case of datasets which are not linearly separable, the *kernel trick* is used. The original data is mapped into higher dimensional feature space and a linear classifier is constructed in this feature space. This is equivalent to constructing a non-linear classifier in the original input space. This mapping is implicitly given by the so called kernel function.

Work conducted by Widodo et al. presents a study of the application of independent component analysis (ICA) and support vector machines (SVMs) to detect and diagnose induction motor faults [63]. The ICA is used for feature extraction and data reduction from original features. Principal components analysis is also applied in the feature extraction process for comparison with what ICA does. The training of the SVMs is carried out using the sequential minimal optimisation algorithm, and the strategy of multi-class SVMs-based classification is applied to identify faults. In addition, the performance of the classification process due to the choice of kernel function is presented to show the excellent characteristic of kernel function. Various scenarios are examined using data sets of vibration and stator current signals from experiments, and the results are compared to get the best performance of the classification process.

Ge *et al.* propose applying SVMs to the fault diagnosis in sheet metal stamping processes [64]. According to tests on two different examples, one being a simple blanking and the other a progressive operation, SVMs are very effective. In both cases, the success rate is over 96.5%. In comparison, the success rate of the popular artificial neural network (ANN) is just 93.3%. In addition, SVMs require only a few training samples, which is an attractive feature for shop-floor applications. The authors found that SVMs have a number of desirable features: they are accurate and capable of dealing with unseen samples; they require only a small amount of training samples, and they need simple computation in decision-making. All these features make SVMs very attractive for fault diagnosis.

A novel test technique for machine fault detection and classification in electro-mechanical machinery from vibration measurements using one-class support vector machines (SVMs) is presented by Shin et al. [65]. Furthermore, it is shown how comparable the one-class SVMs method is to artificial neural networks in the outlier detection problem of high dimensions. Experiments performed on artificial and real dataset show that the performance of this method is mostly superior to that of the artificial neural network methods. A variety of methods are used in the detection and diagnosis of faults efficiently in real life. The accuracy and fastness is the barometer of reducing the cost of maintenance and operation of process plants. The proposed method, one-class SVMs served to exemplify that kernel-based learning algorithms are indeed highly competitive on a variety of problems with different characteristics and can be employed as an efficient method for machine fault detection and classification.

Qu and Zuo discuss a data-processing algorithm developed to achieve the effectiveness, which includes data cleaning followed by feature selection [66]. The data-cleaning algorithm was developed based on a support vector machine and random sub-sampling validation. Candidate outliers are selected based on fraction values provided by the proposed data cleaning algorithm and final outliers are determined based on their removal impacts on classification performance. The feature selection algorithm adopts the classical sequential backward feature selection. Performance of the data cleaning algorithm was tested using three benchmark datasets. The tests showed good capability of the data cleaning algorithm in identifying outliers for all datasets. The proposed data-processing algorithm was adopted in the classification of the wear degree of pump impellers in a slurry pump system. The results showed good effectiveness of sequentially using data cleaning and feature selection in addressing fault pattern classification problems.

Widodo and Yang present a survey of machine condition monitoring and fault diagnosis using SVMs [67]. They attempt to summarise and review the recent research and developments of SVM in machine condition monitoring and diagnosis. Numerous methods have been developed based on intelligent systems such as artificial neural network, fuzzy expert system, condition-based reasoning, random forecast, etc. However, the use of SVM for machine condition monitoring and fault diagnosis is still rare. SVMs have excellent performance in generalisation, and can produce high accuracy in classification for machine condition monitoring and diagnosis. It was found that until 2006, the use of SVM in machine condition monitoring and fault diagnosis tended to develop towards expertise orientation and the problem-oriented domain.

3.3.5 Wavelets

“Wavelets are functions that dissect data into different frequency components, and then analyse each component with a resolution matched to its scale. They have advantages over traditional Fourier methods in analysing physical situations where the signal contains discontinuities and sharp spikes [68].” The wavelet analysis procedure is to adopt a wavelet prototype function, called an analysing wavelet or mother wavelet. Temporal analysis is performed with a contracted, high-frequency version of the prototype wavelet, while frequency analysis is performed with a dilated, low-frequency version of the same wavelet. Because the original signal or function can be represented in terms of a wavelet expansion (using coefficients in a linear combination of the wavelet functions), data operations can be performed using only the corresponding wavelet coefficients. Furthermore, if the best wavelets are chosen adapted to the data, or truncate the coefficients below a threshold; the data is sparsely represented. This sparse coding makes wavelets an excellent tool in the field of data compression. Wavelets were developed independently in the fields of mathematics, quantum physics, electrical engineering, and seismic geology. Interchanges between these fields over the past ten years have led to many new wavelet applications such as image compression, turbulence, human vision, radar, and earthquake prediction [68].

Miaoa and Makis propose condition monitoring and classification of machinery state based on hidden Markov models (HMMs) processing information obtained from vibration signals [69]. They state that such condition classification is of great practical importance because it provides updated information regarding machine status on-line and in the process avoids production loss and minimises the chances of catastrophic machine failures. An on-line fault classification system with an adaptive model re-estimation algorithm is presented. The machinery condition is identified by selecting the HMM which maximises the probability of a given observation sequence. The proper selection of the observation sequence is a key step in the development of the HMM-based classification system. The classification system is validated using observation sequences based on the wavelet modulus maxima distribution obtained from real vibration signals. This was proven to be effective in fault detection in previous research conducted by the authors.

Reddy and Mohanta present a real-time wavelet-Fuzzy combined approach for digital relaying [70]. The algorithm for fault classification employs wavelet multi-resolution analysis (MRA) to overcome the problems associated with conventional voltage and current-based measurements due to the effect of factors such as fault inception angle, fault impedance and fault distance. The proposed algorithm for fault location employs wavelet transform together with fuzzy logic. This differs from conventional algorithms that are based on deterministic computations of a projected and well-defined model. The wavelet transform captures the dynamic characteristics of the non-stationary transient fault signals using wavelet MRA coefficients. The fuzzy logic is employed to incorporate expert evaluation through fuzzy inference system (FIS) so as to extract important features from wavelet MRA coefficients for obtaining coherent conclusions regarding fault location. Computer simulations using MATLAB were conducted for a 300 Km 400 KV line. The results of the simulation indicate that both the classification and localisation algorithms are immune from effects of faults inception angle, impedance and distance. A significant contribution of this work is that the proposed location algorithm has a maximum error of 6.5% with a computational time of approximately one cycle. Thus both the classification and location algorithms can be used as effective tools for real-time digital relaying purposes.

Motor current signature analysis (MCSA) has been successfully used for fault diagnosis in induction machines. However, this method does not always achieve good results with variable load torque. Cusidó et al. propose a different signal processing method, which combines wavelet and power spectral density techniques giving the power detail density as a fault factor [71]. This method shows good theoretical and experimental results. MCSA is a good method for analysing motor faults over constant load torque. The experiments performed and the results obtained show that wavelet decomposition is a good technique for analysing non-stationary current signals resulting from variable load torque. However, this method requires a good knowledge of signals to choose the correct frequencies and *mother wavelet* in order to improve the detection of faults. In addition, a merit factor computed by adding the squares of the detail corresponding to the frequency band where the fault harmonic is placed proves to be a powerful tool in order to automate the diagnosis. By means of this merit factor, the introduction of the wavelet analysis could be considered in practical industrial diagnosis systems with the ability to achieve good results both for a constant torque and for a variable load torque.

Prabhakar et al. propose using discrete wavelet transform (DWT) to detect bearing race faults [72]. “Vibration signals from ball bearings having single and multiple point defects on inner race, outer race and the combination faults have been considered for analysis. The impulses visible in vibration signals due to bearing faults are prominent in wavelet decompositions. It is found that the impulses appear periodically with a time period corresponding to characteristic defect frequencies [72].” In case of single defects and two defects on outer race at 180° apart, the impulses appear periodically with a time period corresponding to characteristic defect frequencies. But, in the case of two defects, one on the inner race and the other on the outer race, one set of impulses is running at a time period corresponding to the inner race defect frequency while the other set of impulses is running at a time period corresponding to the outer race defect frequency. Thus, DWT can be used as an effective tool for detecting single and multiple faults in the ball bearings.

Metal loss defects in a buried pipeline are detected by the magnetic flux leakage technique. Characterisation of the defects and sentencing according to the severity is crucial for organised maintenance of pipelines. Mukhopadhyay and Srivastava identify the parameters that characterise a defect and the features of magnetic flux leakage signals (MFLs) that are affected by those parameters. The authors discuss the analysis of the MFL using wavelet transform scores over other methods of its kind and expose some of the incompleteness of the other analysis techniques. A number of experiments were performed on a rotating drum test rig with defects of different shapes and sizes. Wavelet transform decomposition and reconstruction techniques were applied for denoising the raw data. The efficacy of the discrete wavelet transform for denoising MFL is put to test and they present a complete scheme for the characterisation of defects from denoised MFLs. The issue of defect classification is discussed and it is suggested that characterisation to specified accuracies amounts to designing a classifier that assigns a defect into known classes whose shapes and sizes are defined *a priori*.

3.3.6 Conclusion

This section examined several algorithms that can be used for device prognosis. In addition to the algorithms mentioned in the section, several others exist, including Hidden Markov models [73], the sequential Monte Carlo method [74], Gaussian process regression [75], etc. The various applications in which these techniques are applied are noteworthy. The reason for the variety is that some methods are more suited to specific situations than others. Selecting only one technique will be adequate in some situations, but will most probably not be sufficient in every possible case. Making use of several methods does incur a certain cost with respect to implementation complexity. It does, however, present the opportunity to select the best-suited algorithm for the diagnostic application at hand.

3.4 Maintenance Scheduling

3.4.1 Some Scheduling Approaches

Determining whether a device requires maintenance is only half the solution. The next step is to schedule these devices for maintenance and optimise this schedule. It is important to note at this juncture that the overall system will function within a Multi-Agent System and that this system will in all probability be implemented in Java. In the pursuit of an integrated solution the scheduler/planner should preferably be implemented as a Java application. This will eliminate the need to implement some integration middleware between the scheduler and the rest of the system. For this very reason three scheduling/planning implementations based on the Java technology will now be briefly evaluated.

DLPlan is an open-source planner implemented in Java that uses Semantic Web standard technologies for knowledge representation and reasoning [76]. DLPlan utilises OWL-DL (Ontology Web Language – Description Logic: an expressive language to describe domains and problems) and integrates a powerful reasoner to manage knowledge efficiently. OWL (the Web Ontology Language) is designed by the W3C Web Ontology Working Group and provides a language that can be used for applications where the need to understand the content of information is just as important as understanding the human-readable presentation of the content [77]. OWL-DL is a sublanguage that contains the same OWL vocabulary, but is subject to some constraints. DLPlan is able to deal with complex structured domains and incomplete knowledge.

Drools Planner forms part of the of the *Drools* suite of components and from version 5.0 it is now more commonly known as the Business Logic integration Platform (BLiP) [78]. Four main components make up the *Drools* BLiP, namely:

- ***Guvnor*** – Web based governance system, traditionally referred to as a Business Rules Management System (BRMS) or Business Process Management System (BPMS) depending on whether the implementation was rule- or process specific.
- ***Expert*** – The traditional rule engine.
- ***Fusion*** – The event processing component. The name is based on data/sensor fusion terminology.
- ***Flow*** – The workflow module.

JPlan [79] is a java implementation of the *GraphPlan* general-purpose planner. The *GraphPlan* approach consists of constructing and analysing a compact structure called a Planning Graph. *Graphplan* returns the shortest possible partial-order plan, or states that no valid plan exists. Blum and Furst provide empirical evidence to prove that *Graphplan* outperforms the total-order planner, *Prodigy*, and the partial-order planner, *UCPOP*, on a variety of planning problems [80].

Upon closely considering each of these planners and/or schedulers it was decided to implement the maintenance scheduling using Drools Planner. The reason for this is that Drools has shown promise in problem solving (see section 3.4.2.3), especially given the success at the 2007 International Timetabling Competition (ITC 2007) [8]. Geoffrey De Smet achieved a 4th place in the Examination Timetabling track of the ITC 2007 with a preliminary version of *Drools Planner* (then called *Solver*) [81]. Drools planner has also been applied to the Curriculum-based Course Timetabling problem with promising results. This example can quite easily be modified and applied to maintenance scheduling.

3.4.2 Drools Planner

3.4.2.1 *Introduction*

The focus for this work is only on the planner (*Planner*) and the rule engine (*Expert*) used by the planner. The rest of this section gives a brief overview of *Drools Expert* before highlighting some of the essentials of *Drools Planner*.

3.4.2.2 *Drools Expert as Rule Engine*

A Production Rule System is Turing-complete (as in a Turing Machine) with a focus on knowledge representation to express propositional and first order logic in a concise, non ambiguous and declarative manner. The brain of a Production Rules System is an Inference Engine that is able to scale to a large number of rules and facts. The Inference Engine matches facts and data against Production Rules, also called Productions or just Rules, to infer conclusions which result in actions. "A Production Rule is a two-part structure using First-Order Logic for knowledge representation [82]."

```
when
  <conditions>
then
  <actions>
```

The process of matching the new or existing facts against Production Rules is called Pattern Matching, which is performed by the Inference Engine. *Drools* implements and extends the *Rete* algorithm. The *Drools Rete* implementation is called *ReteOO*, signifying that *Drools* has an enhanced and optimised implementation of the *Rete* algorithm for Object Oriented systems.

"A Production Rule System's Inference Engine is state-full and able to enforce truthfulness – called Truth Maintenance [82]." A logical relationship can be declared by actions which mean the action's state depends on the inference remaining true; when it is no longer true the logical dependent action is undone. The "Honest Politician" is an example of Truth Maintenance, which always ensures that hope can only exist for a democracy while there are honest politicians.

```
when
  an honest Politician exists
then
  logically assert Hope

when
  Hope exists
then
  print "Hurrah!!! Democracy Lives"

when
  Hope does not exist
then
  print "Democracy is Doomed"
```

The Truth Maintenance employed by *Drools Expert* is an important aspect as it ensures that assertions made based on facts that are no longer true are automatically removed. This saves execution time by not evaluating cases that have become untrue as well as the associated memory of the deleted assertions.

3.4.2.3 Problems Solved with Drools

Some example problems in which the *Drools Planner* has been applied are briefly described. Some are well known in the classic problem solving domain [83]:

- Curriculum-Based Course Timetabling – The Curriculum-based timetabling problem consists of the weekly scheduling of the lectures for several university courses within a given number of rooms and time periods, where conflicts between courses are set according to the curricula published by the University and not on the basis of enrolment data.
- Patient admission scheduling (PAS) – Assign each patient (that will come to a hospital) a bed for each night that the patient will stay in the hospital. Each bed belongs to a room and each room belongs to a department. The arrival and departure dates of the patients are fixed. Only one bed needs to be assigned for each night.

- The ITC 2007 examination – Each examination should be scheduled during a period and in a room. Multiple examinations can share the same room during the same period.
- The lesson schedule example – No teacher with two lessons may be scheduled in the same time slot and no group of students with two lessons in the same time slot.
- The Miss Manners 2009 example – Miss Manners invites 144 guests and prepares 12 round tables with 12 seats each. Every guest should sit next to someone (left and right) of the opposite gender. Neighbouring guests should have at least one hobby in common. There should be two politicians, two doctors, two socialites, two sports stars, two teachers and two programmers at each table. And the two politicians, two doctors, two sports stars and two programmers should differ.
- The *n*-queens puzzle – Use a chessboard of *n* rows and *n* columns and place *n* queens on the chessboard. No two queens must be able to attack each other and a queen can attack any other queen on the same horizontal, vertical or diagonal line.
- The travelling tournament – Schedule matches between *n* teams with several constraints. Each team should play every other team twice – once home and once away. Each team has exactly one match in each time slot. No team must have more than three consecutive home or three consecutive away matches. No two consecutive matches of the same two opposing teams. All this should be achieved while trying to minimise the total distance travelled by the teams.

3.4.2.4 Solving a Problem with Drools Planner

Every built-in solver implements the Solver interface. The built-in solvers should suffice for most situations and it is thus not usually necessary to implement the Solver interface. Solving a planning problem with *Drools Planner* generally consists of four steps [84]:

1. Build a solver, specifying the particulars of the solver in the configuration.
2. Set a starting solution on the solver, with the help of an implemented *StartingSolutionInitialiser*.
3. Issue the solve command and wait for the solver to terminate.
4. Get the best solution found by the solver.

A Solver should only be accessed from a single thread, except for the methods that are specifically thread-safe. A Solver instance can be built with the *XmlSolverConfigurer* and then configured with a solver configuration XML file. *Drools Planner* makes it relatively easy to switch a solver type by making alterations to the configuration. The benchmark utility allows one to play out different configurations against each other and report the most appropriate configuration for the problem at hand.

3.4.2.5 The StartingSolutionInitializer

The idea behind the solution initialiser is to create a simple algorithm to come up with an acceptable (not perfect) starting solution for the solver to work with. For large problems, a simple filler algorithm that merely places objects to be scheduled in arbitrary positions does not suffice. A (local search) solver starting from a bad starting solution wastes a great deal of time to reach a solution which an initialiser algorithm can generate in a fraction of that time. An initialiser algorithm usually works as follows [84]:

- It sorts the unplanned elements in a queue according to some general rules, for example by examination student size.
- Next, it plans them in the order they come from the queue. Each element is placed into the best, still available spot.
- It does not change an already planned element. It exits when the queue is empty and all elements are planned.

Such an algorithm is very deterministic: it is really fast, but it cannot be given more time to generate a better solution. In some cases, the solution it generates will be feasible, but in most cases it will not. The algorithm only lays the foundation for a real solver to get to a feasible or more optimal solution. Nevertheless, such an initialiser gives the real solver a serious head start. This is done by implementing the *StartingSolutionInitialiser* interface. A (uninitialised) solution is set on the solver. Once the solver starts, it will first call the *StartingSolutionInitialiser* to initialise the solution. If the *StartingSolutionInitialiser* adds, edits or removes facts, it needs to notify the *workingMemory* about this. It can use score calculation during its initialisation process.

3.4.2.6 Rules and Score Calculation

In *Drools* the rules are specified in a special file or set of files. These rule files have the *DRL* extension. The rules determine how a solutions score is calculated. This is how the rule engine determines whether a new solution is an improvement on the previous one or not. Rules come in two varieties, hard- and soft constraints. Hard constraints cannot be broken and if they are, the solution is not considered valid or feasible. Soft constraints can be broken and a solution that has a soft constraint or several soft constraints broken is still considered feasible. It is most likely not the best possible solution, but a solution nonetheless. It is possible to have score calculation based on soft or hard constraints separately, or as a combined score. This configuration is determined by the XML configuration file.

It is recommended that *Drools* be used in the forward-chaining (apply inference rules to extract more data from available data) mode, and for score implementations this will enable delta-based (only changes between previous and present solutions are evaluated) score calculation instead of a full score calculation on each evaluation of a solution. This means that only constraints that have changed in one or more of their conditions since the previous evaluation will be recalculated. The performance gain is considerable and increases exponentially relative to the size of the planning problem. This performance gain is automatic and simplifies implementations considerably without the need to write a complex delta-based score calculation algorithm. The rule engine performs most of the heavy lifting.

3.4.2.7 Choosing a Next Step

A move is the change from a solution A to a solution B. A move can have a small or large impact. Each of the move types will be an implementation of the *Move* interface. *Drools Planner* calls the *doMove(WorkingMemory)* method to do a move. The *Move* implementation must notify the working memory of any changes it does on the solution facts. You need to call the *workingMemory.update(FactHandle, Object)* method after modifying the fact. Note that one can alter multiple facts in a single move and effectively create a big move (also known as a coarse-grained move). *Drools Planner* automatically filters out moves that cannot be done by calling the *isDoable(WorkingMemory)* method on a move. A move that cannot be done is a move that changes nothing on the current solution or a move that is impossible to do on the current solution. A move that can currently not be done can become possible in a later solution.

Each move has an undo move: a move (usually of the same type) which does the exact opposite. An undo move can be created from a move, but only before the move has been done on the current solution. The local search solver can do and undo a move more than once, even on different (successive) solutions. A move must implement the *equals()* and *hashcode()* methods. 2 moves which make the same change on a solution must be equal. It is also recommended to implement the *toString()* method as it allows one to read *Drools Planner*'s logging more easily.

At each solution, local search will try all possible moves and pick the best move to change to the next solution. It is up to the programmer to generate those moves. Not all moves can be done. The number of possible solutions is much more than the number of moves that can be done. It is important not to create moves to every possible solution. Instead use moves that can be sequentially combined to reach every possible solution.

It is recommended to verify that all solutions are connected, in some manner, by your move-set. This means that by combining a finite number of moves one can reach any solution, from any solution. If this is not the case, possible solutions are already being excluded from the start. This is usually caused by limiting the generation to only big moves. Big moves may outperform small moves in a short test run, but it may be an entirely different scenario as the test run size is increased. It is recommended to mix different move types and it is generally a good idea to prefer small (fine-grained) moves to big (course-grained) moves, because the score delta calculation performance benefit will be more prominent. Sometimes it is possible to remove a hard constraint by using a certain set of big moves, thus gaining performance and scalability.

The winning move is a step. The local search solver tries every move on the current solution and picks the best accepted move as the step. The move with the highest score is picked as the next step. If multiple moves have the same highest score, one is picked randomly. The step is made and from that new solution, the local search solver tries all the possible moves again, to decide the next step. This occurs continually in a loop.

A simple local search always takes improving moves. This may seem adequate, but this is not the case. There are a number of problems:

- It can get stuck in a local optimum. For example, if it reaches a solution X with a score -1 and there is no improving move, it is forced to take a next step that leads to a solution Y with score -2. Consequently, however, it is very real that it will pick the step back to solution X with score -1. It will then start iterating indefinitely between solution X and Y.
- It can start walking in its own footsteps, picking the same next step at every step.

Drools Planner implements superior local searches, such as *taboo search* and *simulated annealing* which can avoid these problems. It is recommended to never use a simple local search, unless the possibility of local optima is entirely eliminated in the specific planning problem. The local search solver decides the next step with the aid of three configurable components:

- A selector selects (or generates) the possible moves of the current solution.
- An acceptor filters out unacceptable moves. It can also weigh a move it accepts.
- A forager gathers all accepted moves and picks the next step from them.

3.4.2.8 Benchmarking the System

Drools Planner supports several solver types and it can be troublesome to determine the best solver for a particular solution. Although some solver types generally perform better than others, it really depends on the domain of the problem. Several solver types also have settings that can be configured. The settings can influence the results of a solver significantly, although most settings should perform surprisingly well out-of-the-box. *Drools Planner* includes a benchmarker to alleviate the difficulty of choosing a solver and settings. This allows one to play out different scenarios against each other and pick the best configuration for the problem domain at hand.

Every *solverBenchmark* entity contains a solver configuration (for example, a local search solver) and one or more *unsolvedSolutionFile* entities. It will run the solver configuration on each of those unsolved solution files. A name is optional and generated automatically if omitted. The common sections of multiple *solverBenchmark* entities can be extracted via inheritance and then be overridden for each *solverBenchmark* entity. The benchmarker supports outputting the best score over time statistic as a graph or CSV (comma separated values) file. It will output an overview of all graphs and CSV files as an index.html file in the specified statistic directory.

3.5 Conclusion

This chapter focused on maintenance management as it relates to this work. An overview of e-maintenance was given and followed by a discussion of some AI techniques that can be applied in the e-maintenance domain. This discussion was followed by a review of some applications of agents in the e-Maintenance research field. Some algorithms applied to maintenance prognosis and how the benefits could outweigh the cost considerations of a combined approach were considered. In the section on maintenance scheduling a few Java implementations of schedulers were mentioned. *Drools Planner* was selected for this work and in the subsequent sections some of the crucial elements were discussed. The next chapter examines Multi-Agent Systems (MAS), their applications and methodological considerations.

Chapter 4 Multi-Agent Systems

4.1 Introduction

Computational agents and their theory date back at least a quarter of a century when research in distributed artificial intelligence (AI) was instigated [85]. The early 1990s were characterised by the simultaneous appearance of agents in the information and communication technologies. The real breakthrough occurred a decade ago when there was a notable shift in emphasis of the mainstream AI research community. "The focus on logic was extended and attention changed from goal-seeking to rational behaviour; from ideal to resource-bound reasoning; from capturing expertise in narrow domains to re-usable and sharable knowledge repositories; from the single to multiple cognitive entities acting in communities [85]." As Monostori et al. also stated in [85], this shift in focus coincided with the development of several technologies, including network-based computing, the Internet, mobile computing, ubiquitous computing and human-oriented software engineering methodologies. This combination later resulted in what is now known as the agent paradigm.

There is no universally accepted definition for the concept of agent and related software. No consensus has been reached since there are still ongoing debates and controversies surrounding the subject. Another reason is the fact that the field is highly interdisciplinary [6]. Inspiration is drawn from diverse areas such as economics, philosophy, logic, ecology, and the social sciences. Instead of focusing on a specific definition of an agent or multi-agent systems, it is more important to describe the basic characteristics of an individual agent and how a group of agents form a multi-agent system.

An agent is an autonomous entity. Autonomy distinguishes an agent from a normal object in the sense that normal objects have no control over their execution [86]. An agent also has goals or intentions, which it strives to fulfil by taking certain actions. These goals relate to the environment in which the agent is situated. Jussila defines an agent as follows: "*An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives* [87]." This is an adaptation of the definition provided by Jennings [88] and Wooldridge [89]. A multi-agent system can be viewed as several agents working together (or competing) to achieve the designed specifications of the system.

4.2 Applications of Multi-Agent Systems

This section examines some of the applications of multi-agent systems (MAS). It is interesting to note the variety of applications and how vastly they differ from one another. Several agents' applications related directly to automation and manufacturing are mentioned, as well as agent applications in other fields of research.

At the Automation Technology Laboratory of the Helsinki University of Technology there are ongoing research activities utilising agent technologies with a focus on automation. Chakraborty used MAS to facilitate fault detection and recovery as part of abnormal situation handling in a process automation system (PAS) [90]. The idea was to develop an iterative approach to fault detections. Once an agent detects a fault it should autonomously initiate the recovery processes. Focus was placed on the development of a method that could at least partly recover from a fault and lay the foundation for further corrective action. Even though the system suffered performance degradation, the upside was that total production downtime could probably be avoided.

Fajt on the other hand argued the use of information agents in information processing in PAS as an alternative to technologies such as OLE for Process Control (OPC) [91]. Individual aspects of information agents are highlighted in the work, as well as the advantages and disadvantages information agents pose over OPC. An information access module and an agent message evaluating mechanism were also implemented. These were tested and presented in a demonstration scenario where information agents were connected to a physical test environment. Agents were used to monitor and diagnose the status of a running system and an external agent was notified regarding any changes in system status.

Leitão *et al.* discuss the opportunity to use multi-agents technology in automation and distributed manufacturing systems [92]. They describe a manufacturing cell control application which is developed using multi-agent technology. The authors aimed at exploiting the autonomy, cooperation, reactivity and pro-activity of MAS and applying it to the manufacturing domain. The authors presented an agent-based manufacturing cell controller implementation, using an architecture known as ADACOR (Adaptive and Cooperative Control Architecture for Distributed Manufacturing Systems) [93], developed by them for the control of distributed manufacturing systems. The shop floor of the platform is organised as a set of several cells. A cell refers to a group of resources, grouped on the basis of its functionality and location. As a test, four different products were designed: *Base*, *Corpo*, *Tampa* and *Pega*. These individual parts were combined in an assembly to create a final product called *Cinzeiro*. The cell controllers within the flexible platform are represented by Supervisor agents. Inside the manufacturing cell, the physical resources are represented by Operational agents. Each available product in the system is represented by a Product agent. When it is necessary to produce a product, a Task agent is launched to supervise the execution of the product assembly. Each product launches a task agent to supervise its assembly. If the product comprises other sub-products, the task agent will launch the required task agents to supervise the execution of each sub-product assembly. Each task agent announces all operations to the available operational- and supervisor agents, allocate them to the best proposals and wait for the end of the execution of the operation.

Roy *et al.* present a novel approach to shop floor control, called *SYROCO* [94]. It aims to solve dynamic production control problems in real-time, and to automate the control process as much as possible in order to adapt the system to production plan modifications. In addition, it aims to rationalise decision-making by means of a strong hierarchical structure. To achieve this, the system is based on a two-fold hybrid multi-agent platform. First, control is hierarchically distributed and decision-making is centralised. This type of system mediates between centralised and hierarchical architecture types. Centralisation limits the competition between agents competing to solve a problem while hierarchical distribution allows each agent to manage only one product. Secondly, a full schedule is completed at the beginning of the process; this schedule can be partially or totally modified during the process. The hybrid approach offers significant gains in terms of response times and reactivity capabilities.

Cowling *et al.* proposed the use of a multi-agent architecture to solve some of the complexities accompanying the production of steel [95]. They aimed to find coherent and effective schedules for the wide variety of production steps. This is a challenging task in the associated dynamic environment. The MAS is used for the integrated dynamic scheduling of a complex steel production process, the focus of which is the hot strip mill (HSM) and the continuous caster. These processes have scheduling systems with very different objectives and constraints that govern them. The environment also has a substantial quantity of real-time information concerning production failures and customer requests. Each process is assigned to an agent. This agent independently seeks an optimal dynamic schedule at a local level, taking into account local objectives, real-time information and information received from other agents. Each agent reacts to real-time events in order to fix any problems that occur. Special focus is given to the HSM agent which uses a taboo search heuristic ("a meta-heuristic superimposed on another heuristic [96]") to promptly create good quality predictive-reactive schedules. During the scheduling process the other agents simulate the production of coil orders and real-time events. When real-time events on the HSM are triggered, the HSM agent might decide whether to update the current schedule or discard the schedule entirely in favour of a new one. To address this problem, several strategies for schedule repair and complete rescheduling are investigated in order to address the problem. Their performance is assessed using an experimental simulation framework to determine measurements of utility, stability and robustness.

The use of MAS is not merely limited to automation and manufacturing. Rendón-Sallard *et al.* exploited MAS's capability to cope with complexity to simulate multiple scenarios in a river catchment system in order to support the decision-making of the river basin's management [97]. In addition to the main goal of the developed MAS, other objectives included managing critical episodes, minimising the discharge of poorly treated wastewater, maximising the use of the installations treatment capacity, minimising economical costs of new investments and daily managing and maintaining a minimum flow in the river to guarantee an acceptable ecological state. A user could simulate various scenarios and the MAS would take into account the consequences that could affect any element of the water system. It would provide a set of actuation alternatives and then propose the best strategy to deal with the given situation.

A multi-agent framework was applied to coordinate video camera-based surveillance by Patricio *et al.* [98]. The agents' ability to coordinate improved the global image and task distribution efficiency. A software agent was embedded in each camera to control the capture parameters. Coordination among the agents was facilitated by the exchange of high-level messages. Agents used the messages from all other agents and the internal symbolic model to interpret the current situation and improve global coordination. “Each agent deliberatively makes decisions to carry out the system tasks coherently with other agents, considering both the information generated in its local process and the information available in the network [98].”

Mathieson *et al.* demonstrated how an agent based approach could be used to facilitate travel and tourism planning [99]. Web and agent services were combined to provide complex services such as the creation of an itinerary. The project aimed to work particularly on the technical research issues, such as flexible and scalable distributed directory services, service description languages, and service composition and execution. Contributing to the openNet platform [100] was also a priority by developing agents in the area of tourism. This project also had potential industry value by focusing on the realisation of web services in a business-to-consumer B2C model. Tourism information could be provided by exploiting wireless accessibility on devices such as mobile phones and PDAs. This was aided by the high quality of tourism data provided by the Australian Tourism Data Warehouse (ATDW) [101] with whom the authors had a collaboration agreement. The ATDW stores information on a range of Australian destinations. This data was used to develop agents which are capable of providing real, context specific data.

Shang and Shi used MAS to overcome some of the problems surrounding medical image interpretation for a computerised physician [102]. They developed an intelligent agent approach based on the concept of active fusion and agent-oriented programming to create a Web-based multi-agent system for interpreting medical images. Two major types of intelligent agents were created. The radiologist agents decompose the image interpretation task into smaller subtasks, and then made use of multiple agents to solve the subtasks. Finally, the solutions to the subtasks are intelligently combined to solve the image interpretation problem. Through a Web-based interface a patient representative agent is able to take questions from the user (usually a patient). A given set of images are then interpreted by multiple radiologist agents, each forming an opinion. These opinions are then integrated and presented to the user. A patient representative agent could also answer questions based on the information in a medical information database. To maximise efficiency and user satisfaction, the patient representative agents had to be as informative and timely as communicating with a human being. By combining the use of pseudo-natural language processing, a knowledge base in XML, and user communication through a Microsoft Agent, the patient representative agents are able to answer queries effectively.

Vermeulen *et al.* present an approach to the hospital patient scheduling problem, where patients can have multiple appointments that have to be scheduled to different resources [103]. To efficiently solve this problem they developed a multi-agent Pareto-improvement [104] (a change that makes somebody better off and nobody worse off) appointment exchanging algorithm called MPAEX (Multi-agent Pareto Appointment Exchanging). "It respects the decentralization of scheduling authorities and continuously improves patient schedules in response to the dynamic environment [103]." The hospital patient-scheduling problem is modelled in terms of the health care cycle where a doctor repeatedly orders sets of activities to diagnose and/or treat a patient. The authors introduced the Theil index [105] (a measure of economic inequality) to the health care domain to characterise different hospital patient-scheduling problems in terms of the degree of relative workload inequality between required resources. The distributed and dynamic MPAEX performed almost as good as the best centralised and static scheduling heuristic, and its flexibility allows for variations in the model settings.

4.3 AOSE Methodologies Review

Software engineering is "the application of a systematic, disciplined, quantifiable, approach to the development, operation, and maintenance of software [106]." It focuses on developing large applications, covering not only the technical aspects of building software systems, but also the management issues. Agent Oriented Software Engineering (AOSE) is required to sustain the novel abstractions introduced by agent-based computing.

With the increasing number of successful agent-based applications and techniques, the question of applicability in an industrial context becomes increasingly prominent. This implies the definition of repeatable, reusable, measurable and robust software process and techniques for multi-agent systems (MAS) development [7]. This is the reason for an increased devotion to the definition of methods and tools for supporting AOSE. It also involves the definition of [7]:

- Modelling languages for the specification of MAS.
- Techniques for requirements elicitation and analysis.
- Architectures and methods for designing agents and their organisations.

- Platforms for the implementation and deployment of MAS.
- Validation and verification methods.

The remainder of this section is concerned with the evaluation of several AOSE methodologies as well as a comparative study to determine a possible candidate. Several prominent methodologies exist but a full comparison of all of them is beyond the scope of this work. The methodologies (in alphabetical order) to be evaluated and compared are as follows:

- Gaia [107].
- MaSE [108].
- PASSI [109].
- Prometheus [110].
- ROADMAP [111].
- Tropos [112].

These methodologies will be evaluated according to the completeness of support for the software development life cycle, in particular analysis, design and implementation, as well as maturity and comprehensiveness of support tools.

4.3.1 Reviewed Methodologies

4.3.1.1 Gaia

The Gaia methodology for AOSE was first introduced by Woolridge *et al.* [113] in 2000. The Gaia methodology is general and can be applied to a wide variety of multi-agent systems (MAS). It deals with both the macro level (societal) and micro level (agent) aspects of systems. In Gaia, a MAS is a computational organisation which consists of various interacting roles. “Gaia is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly [113].”

In the Gaia methodology the requirements capture phase is viewed as independent of the paradigm used for analysis and design. The analyst moves from abstract to increasingly concrete concepts. Each successive step introduces greater implementation bias, and reduces the number of possible systems that could be implemented to satisfy the original requirements statement. “Analysis and design can be thought of as a process of developing increasingly detailed models of the system to be constructed [113].” Figure 4-1 summarises the main models used in Gaia.

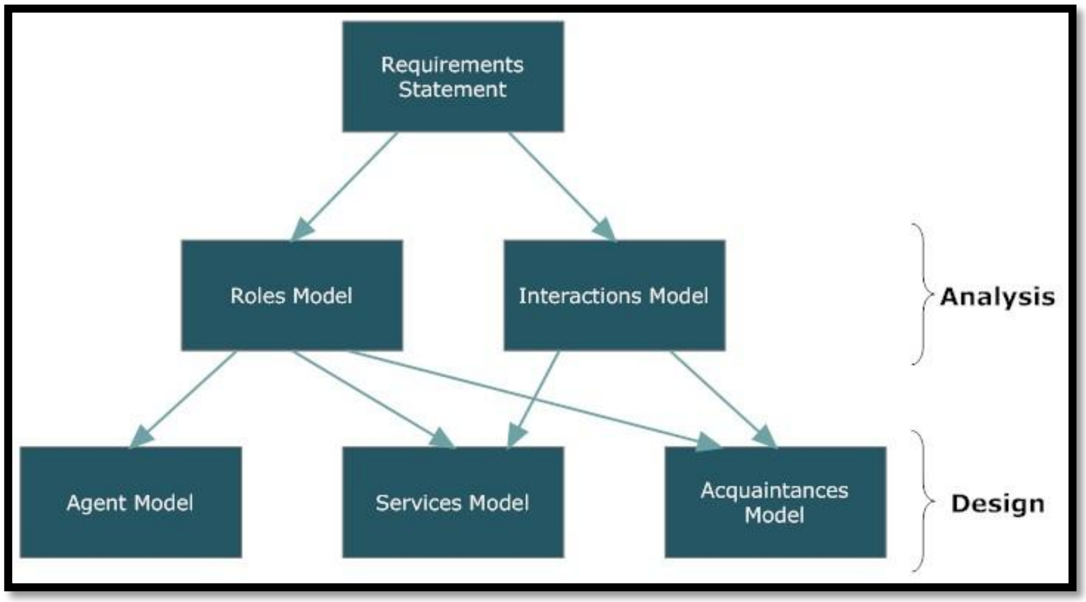


Figure 4-1: Relationships between Gaia's models

Some of the terminology and notation used in Gaia is borrowed from object-oriented analysis and design. This, however, is not simply a native attempt to apply such methods to agent-oriented development. An agent-specific set of concepts is provided whereby a software engineer can understand and model a complex system. A developer using the Gaia methodology is encouraged to think of building agent-based systems as a process of organisational design [113].

The main concepts in Gaia can be divided into two categories, namely *abstract* and *concrete*. These concepts are summarised in Table 4-1: Abstract and concrete concepts in Gaia. Abstract entities are used to analyse and conceptualise the system, but they do not necessarily have any direct realisation within the system. By contrast, concrete entities are used in the design process, and will typically have direct counterparts in the run-time system.

Table 4-1: Abstract and concrete concepts in Gaia

Concept	Detail
Abstract	Roles
	Permissions
	Responsibilities
	Protocols
	Activities
	Liveness Properties
	Safety Properties
Concrete	Agent Types
	Services
	Acquaintances

Until now, no support tools have been developed specifically for Gaia in its original form. Some researchers have proposed implementing MAS on JADE [114] using the Gaia methodology, as discussed in [115], [116] and [117]. Considerable research has been done in extending Gaia in an attempt to overcome some of the perceived drawbacks. One of the disadvantages is the fact that the Gaia methodology only covers the requirements, analysis and design phases of the software development cycle [115].

4.3.1.2 MaSE

"MaSE (Multi-agent Systems Engineering) is an attempt on how to engineer practical multi-agent systems. It provides a framework and a complete lifecycle methodology for analyzing, designing and developing heterogeneous multi-agent systems [118]." MaSE is a further abstraction of the object-oriented paradigm where agents are at the same level of abstraction as objects. Inspired by traditional object oriented software engineering, the MaSE approach is based on a cascading development model.

Figure 4-2 illustrates the detailed methodology map of MaSE [118]. MaSE is a goal-based methodology and the analysis is role-directed. Roles are played by the agent classes that capture the organisation. A very important concept in the MaSE methodology is the Goal Hierarchy diagram. The goals are structured into a form which can be used in the design phase. Goals in the Goal Hierarchy diagram are organised according to importance. Use cases are also used in MaSE and these are drawn from the system requirements. From the use cases, sequence diagrams are constructed to determine the minimum set of messages that must be passed between roles.

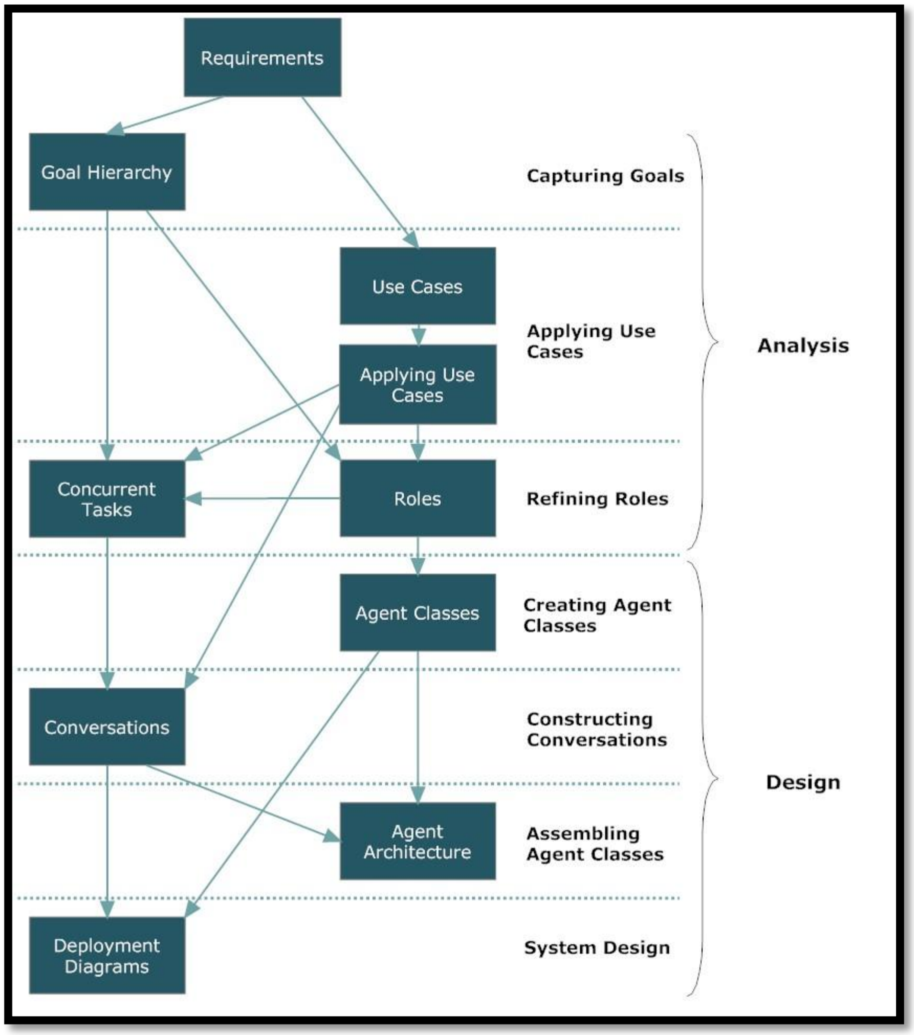


Figure 4-2: MaSE detailed methodology map

MaSE has a few limitations due to the assumptions made in an effort to simplify the research in developing the methodology [108]. The assumptions are as follows:

- The system being created is closed and all external interfaces are encapsulated by an agent that participates in the system communication protocols.
- Dynamic systems, where agents can be created, destroyed, or moved during execution, are not considered.
- Inter-agent conversations are one-on-one as opposed to being multi-cast.
- Systems designed using MaSE are not very large; the target is ten or less software agent classes.

MaSE can be used to design agents that operate in an open environment, even though not initially designed to meet this specification, as long as there are appropriately defined protocols for the agents to use.

MaSE builds upon the work of many agent-based methodologies and approaches; it takes many ideas and attempts to combine them into a complete, end-to-end methodology. The main advantage of MaSE compared to other methodologies is its scope and completeness [108]. The primary focus of MaSE is to guide the designer through the software development life cycle, from initial specification to an implemented agent system. MaSE is independent of any particular multi-agent system architecture, agent architecture, programming language, or message-passing system.

In conjunction with the MaSE methodology, a tool known as *AgentTool* was also developed to support the development of multi-agent systems using MaSE [108]. Developing MaSE and *AgentTool* together allowed the focus to remain on automation. The *AgentTool* system supports the entire life cycle, from Goal Hierarchy diagram to code generation.

4.3.1.3 PASSI

Process for Agent Societies Specification and Implementation (PASSI) is the result of a long period of study and experimentation mainly in the robotics field [109]. PASSI is composed of five models that include several distinct phases, namely System Requirements, Agent Implementation, Case Model and Deployment Model. Figure 4-3 illustrates the models and phases of PASSI. One of the advantages of using PASSI is derived from the complete support provided by *PTK* (PASSI Toolkit) and *AgentFactory*.

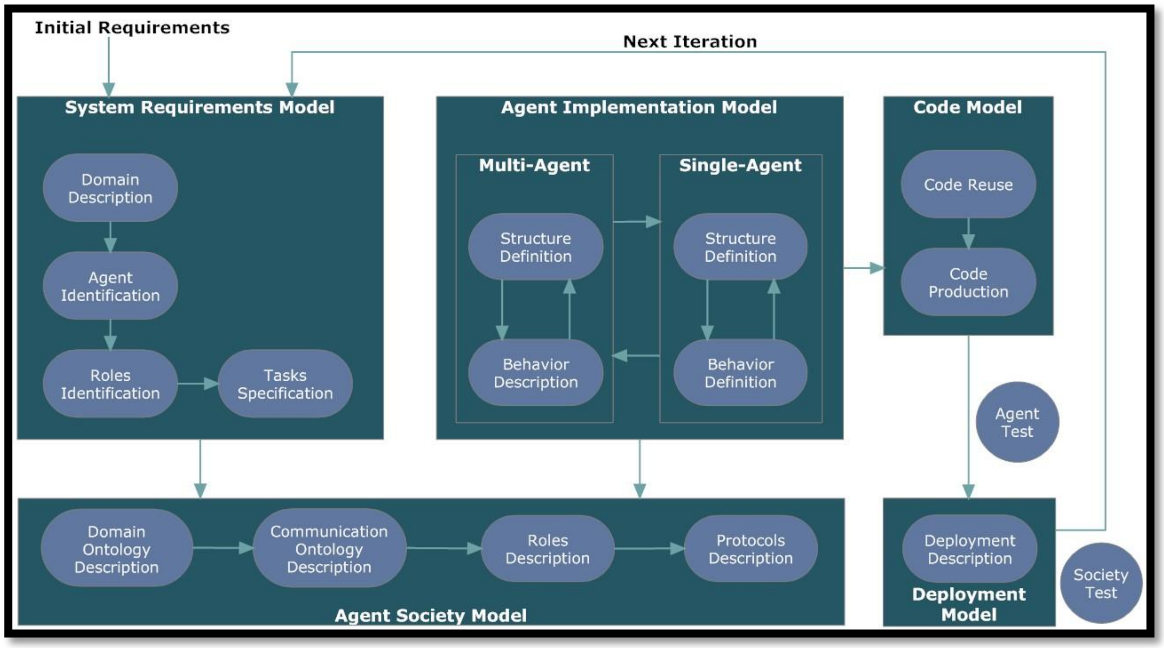


Figure 4-3: Models and phases of the PASSI methodology

PTK is an add-in for Rational Rose, a diffused commercial UML-based CASE tool, which can enhance the design robustness and coherence, thus lowering the designer effort. *PTK* also provides a strong support for the code production phase by means of automatic code generation of a considerable amount of code. This designer is guided throughout all the design work, allowing the automatic compilation of several diagrams and the generation of the code for the agent skeletons.

AgentFactory is a pattern re-use application that, with a few mouse clicks, allows for the rapid prototyping of great parts of complex FIPA-compliant systems. It facilitates the automatic generation of pattern code for both the JADE and FIPA-OS, which are among the most diffused FIPA compliant agent platforms.

Chella *et al.* have completed several projects using the PASSI methodology and the aforementioned support tools in robotics and information systems [109]. The outcomes of their work have been very encouraging.

4.3.1.4 *Prometheus*

"It is widely accepted in the agent research community that a key issue in the transition of agents from research laboratories to industrial practice is the need for a mature software engineering methodology for specifying and designing agent systems [110]." The Prometheus methodology attempts to address this need. Prometheus is intended to be a practical methodology, with the aim of completing and providing everything that is needed to specify and design agent systems. Other features of Prometheus that distinguish it from other methodologies are:

- Prometheus is detailed and provides detailed guidance on how to perform the various steps that form the methodology.
- Prometheus supports, but is not limited to, the design of agents that are based on goals and plans.
- Prometheus covers activities that range from requirements specification through to detailed design.
- Prometheud facilitates the use of support tools in the form of the Prometheus Design Tool (*PDT*) [119], [120], which is freely available.
- Various student groups have been educated in the use of Prometheus and the feedback of these students has been used in enhancing the methodology.

The Prometheus methodology includes a description of concepts for designing agents, a process, and a number of notations for capturing designs, as well as many "tips" and techniques that aid the developer in how to follow the various Prometheus design steps [110]. The Prometheus methodology consists of three phases:

- System Specification:
  - Where the system is specified using goals and scenarios;
  - System interface to environment is described in terms of actions, precepts and external data, and
  - Functionalities are defined.
- Architectural Design:
  - Where agent types are identified;
  - Overall structure is captured in a system overview diagram, and

- Scenarios are developed into interaction protocols.

- Detailed Design:
  - Where details of the internals of each agent are developed and defined in terms of capabilities, data, events and plans, and
  - Process diagrams act as a stepping stone between interaction protocols and plans.

These phases include models that focus on the dynamics of the system, (graphical) models that focus on the structure of the system or its components, and textual descriptor forms that provide details for individual entities [110]. Figure 4-4 depicts the phases of the Prometheus methodology.

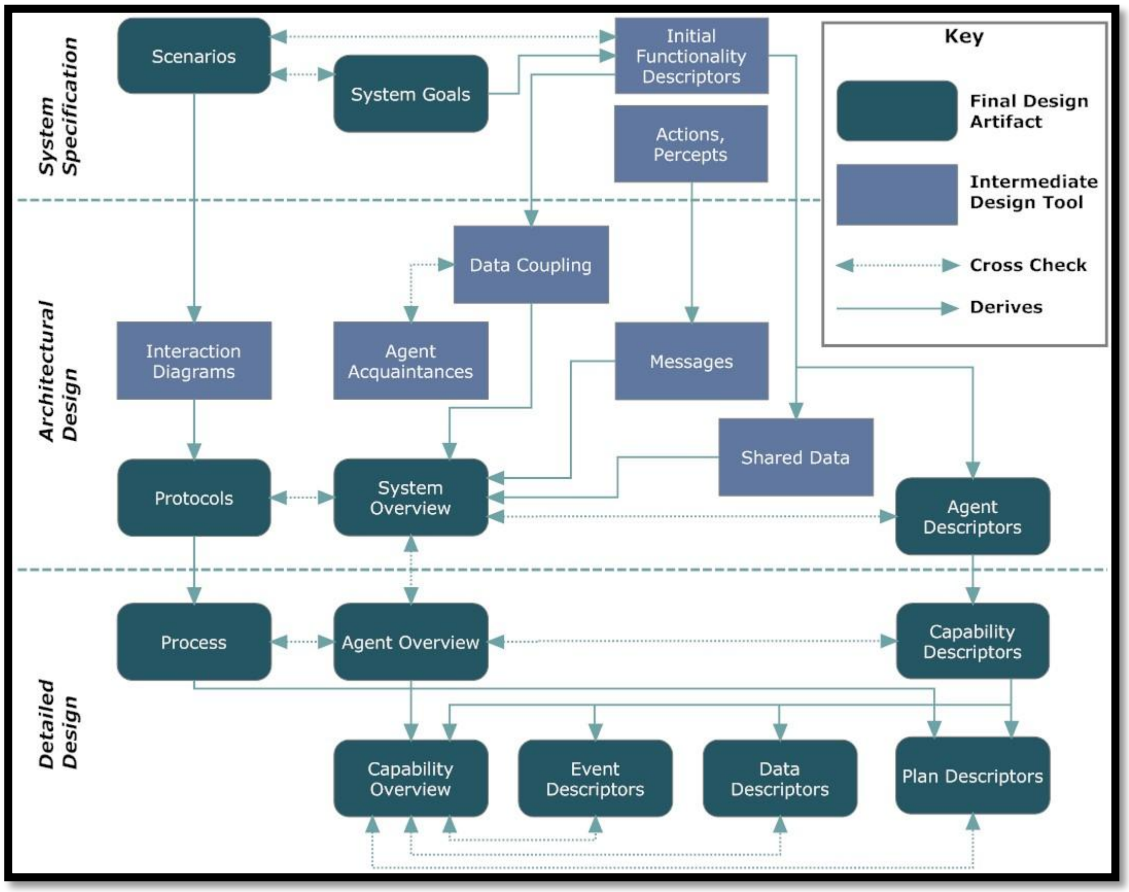


Figure 4-4: Phases of the Prometheus methodology

The developers of Prometheus have attempted to emphasise the iterative nature of design, both across the phases of Prometheus and within each phase [110]. The result of the iterative nature is that the design is often modified. These modifications can introduce inconsistencies that can further complicate the design. Developing designs by hand (using only standard tools such as word processors) is also error prone. This is why support tools are invaluable, and as a result the *PDT* was developed.

This tool allows users to create and modify Prometheus designs. Certain inconsistencies cannot be introduced while using the tool and a cross-checking capability is provided to detect other forms of inconsistencies. Individual design diagrams can be exported and the complete design can be generated in the form of a report. The *PDT* is freely available at [120], and ongoing development ensures that further functionality is added with each release.

The JACK Development Environment (JDE) [121] is another tool that supports the Prometheus methodology. It provides Prometheus-style overview diagrams which can be used to generate skeleton code [110]. JACK is a commercial tool which needs to be licensed before development can be done. A trial licence with full functionality is available after registering online.

4.3.1.5 ROADMAP

Role Oriented Analysis and Design for Multi-Agent Programming (ROADMAP) is an extension of the Gaia [107] methodology that attempts to allow better engineering of open systems [111]. Juan et al. focus on the Gaia methodology for its architectural independence and simplicity. The methodology presents explicit and useful models of the social aspects of agents. Gaia was not designed for open systems and only provides weak support for scalability. However, its simplicity allows improvements to be made with relative ease. ROADMAP attempts to extend Gaia in the following ways:

- Support for requirements gathering.
- Explicit models to describe the domain knowledge and execution environment.
- Levels of abstraction during the analysis phase, to allow iterative decomposition of the system.
- Explicit models and representations of social aspects and individual agent characteristics, from the analysis phase to the final implementation.

- Runtime reflection, modelling mechanisms to reason and change the social aspects and individual agent characteristics at runtime.

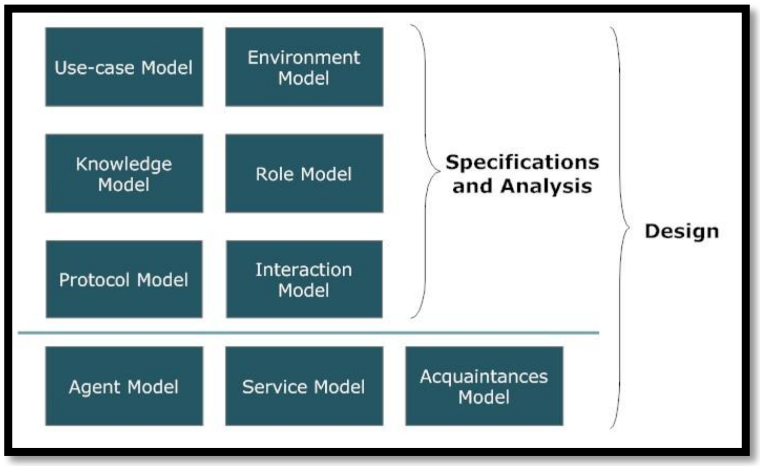


Figure 4-5: ROADMAP models

Figure 4-5 shows the structure of the ROADMAP models. The analysis phase is extended to cover specification as in the original Gaia methodology. Requirements gathering is supported by the introduction of a use-case model. An environment model and a knowledge model are derived from the use-cases and subsequently provide holistic views on the execution environment and the domain knowledge [111].

“The Roadmap Editor Built for Easy development (REBEL) [122] is a tool for building Goal Models and Role Models (or Role Schemas) during the analysis stage [123].” REBEL is available as an Eclipse [124] plug-in for Java. Eclipse was chosen for the following reasons:

- Its extensible plug-in architecture.
- It enables the developer to concentrate on the plug-in without worrying about the capabilities of the development environment.
- Eclipse’s Graphical Editing Framework (GEF) and Eclipse Modelling Framework (EMF) technologies.
- Strong community support plus its API documentation is well maintained.

4.3.1.6 Tropos

Tropos was designed to exploit all the flexibility provided by Agent Oriented Programming (AOP). It is based on Eric Yu's work on the *i** (i-Star) framework [7]. The main motivation underlying the concept of the *i** model was to develop a richer conceptual framework for modelling processes which involve multiple participants (both human beings and software systems). "The rationale of the *i** model is that by doing an earlier analysis, one can capture not only the what or the how; but also the why a piece of software is developed [112]." This promotes a more refined analysis of system dependencies and encourages a uniform treatment of the system's functional and non-functional requirements.

In Tropos, the agent paradigm and related mentalistic notions are used through all phases of the development process. Four phases of development are supported, namely *early requirements, late requirements, architectural design* and *detailed design*. Figure 4-6 (adapted from [125]) illustrates an example actor diagram which is used throughout the design of a system. The diagram shows actors and the relationships between them. All roleplayers, existing or to be designed, are modelled as actors or agents. Agents in the system depend on one another for the completion of goals, supplying resources and executing plans. An actor (*depender*) depends on another actor (*dependee*) for the furbishing, fulfilment or execution of a resource, goal or plan (*dependum*) respectively. Each of the actors has an internal goal diagram that demonstrates how goals and plans that are delegated to the agent in question are accomplished.

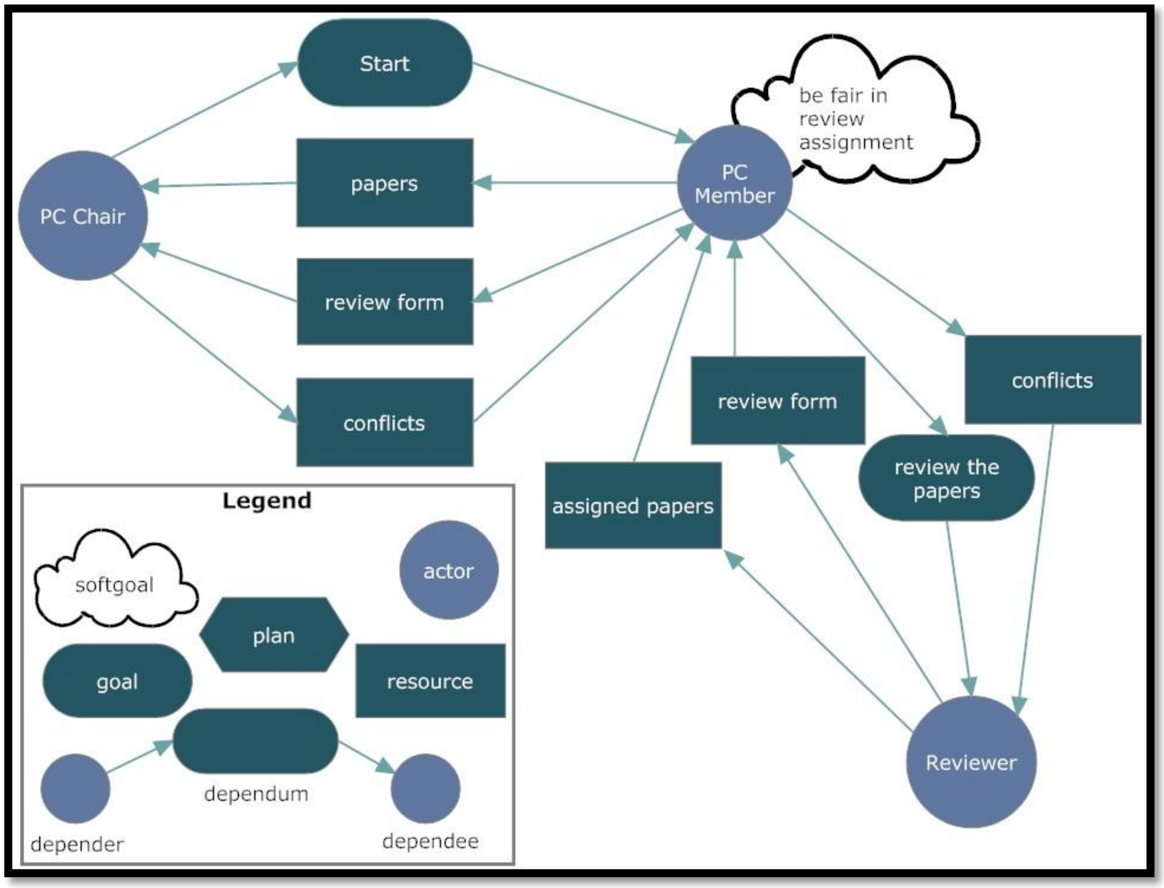


Figure 4-6: The Tropos actor diagram describing a sketch of the conference review process

For the implementation of agents, Tropos chooses a BDI (*Beliefs-Desires-Intentions)* platform, specifically JACK [126] Intelligent Agents for the implementation. JACK provides five main language constructs: agents, capabilities, database relations, events and plans. Developers are required to map each concept in the design phase to the relevant constructs in JACK. "Tropos provides several guidelines and heuristics for mapping Tropos concepts to BDI concepts and BDI concepts to JACK constructs [127]."

An alternative tool for developing agent systems using the Tropos methodology is the Tool for Agent Oriented visual Modelling for the Eclipse [124] platform (TAOM4E) [128]. The TAOM4E environment takes into account emerging guidelines and standards from the Object Management Group's (OMG) [129] Model-Driven Architecture (MDA) [130] initiative which proposes an approach to software development based on modelling and automated mapping of models to code. The tool is based on the Eclipse Platform, which offers a flexible solution to the problem of component integration.

4.3.2 Summary

This section reviewed several AOSE methodologies. Efforts were made to keep the review as objective as possible. Even though more than one methodology qualifies as an implementation candidate, only one could be used and a decision had to be made. Given the documentation, experience with the design tool (if applicable) and bearing in mind the implemented system should be able to interact with existing systems and be able to integrate into future systems, some of the candidates stood out more than others. These candidate methodologies were *PASSI*, *MaSE* and *Tropos*. Design tool ease-of-use, support and documentation played a big role in the final decision. Further deliberations included how the physical design with the specific methodology was experienced and if the concepts felt natural. Unavoidably, some elements of personal preference also played their role in the final choice. For the current implementation it was decided to utilise Tropos. The next section provides a more detailed discussion of Tropos.

4.4 The Tropos Methodology

The intent of the Tropos methodology is to support all analysis and design activities in the software development lifecycle, from application domain analysis to the system implementation. A model of the system-to-be and its environment is built, which is then incrementally refined and extended to provide a common interface to various software development activities, as well as a basis for documentation and evolution of the software [112]. The five main development phases of Tropos are:

- **Early Requirements** – The requirements engineer indentifies the domain stakeholders and models them as social actors, who depend on one another for goals to be achieved, plans to be performed, and resources to be furnished.
- **Late Requirements** – The conceptual model is extended including a new actor, which represents the system and a number of dependencies with other actors of the

environment. These dependencies define all the functional and non-functional requirements of the intended system.

- **Architectural Design** – Defines the system's global architecture in terms of sub-systems, interconnected through data and control flows. Sub-systems are represented, in the model, as actors and data/control interconnections are represented as dependencies.
- **Detailed Design** – Aims at specifying agent capabilities and interactions. At this point (usually) the implementation platform has already been chosen and this can be taken into account in order to perform a detailed design that will map directly to code.
- **Implementation** – Follows step by step, in a natural way, the detailed design specification on the basis of the established mapping between the implementation platform constructs and the detailed design notions.

Models in Tropos are acquired as instances of a conceptual meta-model resting on the following concepts/relationships [112]:

- **Actor** – Models an entity that has strategic goals and intentionally within the system or the organisational setting.
- **Goal** – Represents actors' strategic interests.
- **Plan** – Represents, at an abstract level, a way of doing something.
- **Resource** – Represents a physical or an informational entity.
- **Dependency** – Exists between two actors, which indicates that one actor depends, for some reason, on the other in order to attain some goal, execute some plan, or deliver a resource.
- **Capability** – Represents the ability of an actor to define, choose and execute a plan in order to fulfil a certain goal, given current world conditions and following a specific event.
- **Belief** – Represents the actor's knowledge of the world.

Various activities contribute to the acquisition of a first early requirement model, to its refinement and to its evolution into subsequent models [112]. *Actor modelling* consists of identifying and analysing both the actors of the environment and the system's actors and agents. *Dependency modelling* consists of identifying actors which depend on one another for goals to be achieved, plans to be performed and resources to be furnished. *Goal modelling* relies on the analysis of actor goals, conducted from the point of view of the actor, by using three basic reasoning techniques: means-end analysis, contribution analysis and/or decomposition. *Plan modelling* can be considered as an analysis technique that complements goal modelling. *Capability modelling* follows architectural design when the system sub-actors have been specified in terms of their own goals and the dependencies with other actors.

4.5 Proposed Methodology Restrictions

A "diminished set of Tropos relationships" are used, as suggested in [131]. "The concepts are based on a pragmatic bottom-up approach which attempts to build a small, but not necessarily minimal set of well-defined basic concepts, and enabling the model to still be usable and intuitive [132]." The idea is to remove all relationships that are redundant or have no meaningful semantics from the meta-model. The reduced set of concepts consists of [131]:

- well-defined semantics,
- a simple and clear design,
- adequate expressiveness, and
- an easier implementation of the prototype.

The set is mainly limited to the goal diagram, which is the knowledge level of a single software agent or system actor. The reduction is aimed at an architectural design model, which offers a base for the construction of the software agent. It is important to note that concepts used for earlier requirements, design refactoring and refinement are not forbidden. The diminished model contains the basic Tropos constructs, such as the actor, goal, soft goal, plan and resources. The restricted model also contains a set of relationships with a limitation on the possible uses between the constructs, namely [131]:

- **AND/OR-Decomposition**: A homogeneous, acyclic relationship to decompose goals, soft goals, plans and resources hierarchically into sub-entities.
- **Means-End Relations**: A relationship between a plan (the *means*) and a goal (the *end*). The other possible combinations from the original meta-model are not allowed.

Relationships between soft goals and goals with the similar semantics can be modelled more accurately with contributions. Homogeneous relations are best modelled with OR-decompositions.

- **Use- and Produce Relations**: A special means-end relationship between a resource and another entity. The relation is labelled as use-relationship if the resource is the *means* and as a produce-relationship if it is the *end*.
- **Delegation**: A relationship between two actors in which a goal, soft goal, resource or plan is fully delegated to another actor. For the proposed prototype the dependum is restricted to a goal.
- **Dependency**: A relationship between two actors, with a goal, resource or plan as dependum and a goal or plan as the "why"-argument. Focusing only on the knowledge-level and on the Agent Oriented paradigm, for the prototype both arguments are restricted to a goal.
- **Contribution**: Contribution relationships can only exist between a plan/goal and a soft goal, of which the values can be "++", "+", "-", and "--". A double symbol (ex. ++) means a greater positive or negative contribution, while a single symbol indicates a lesser contribution.

All relationships are non-reflexive (an entity cannot be related to itself) and can be many-to-many (for example a goal can be decomposed into any number of sub goals and, vice versa, sub goals can belong to numerous decompositions, having several parent goals). No combination of different types of relationships (AND/OR/means-end) is allowed when decomposing a goal or plan in the diminished set. The only exception to the rule is a 'why'-dependency, which can be added to any decomposition. If other combinations are required, they can be modelled by inserting 'dummy' goals or plans, as illustrated by Figure 4-7 from [131].

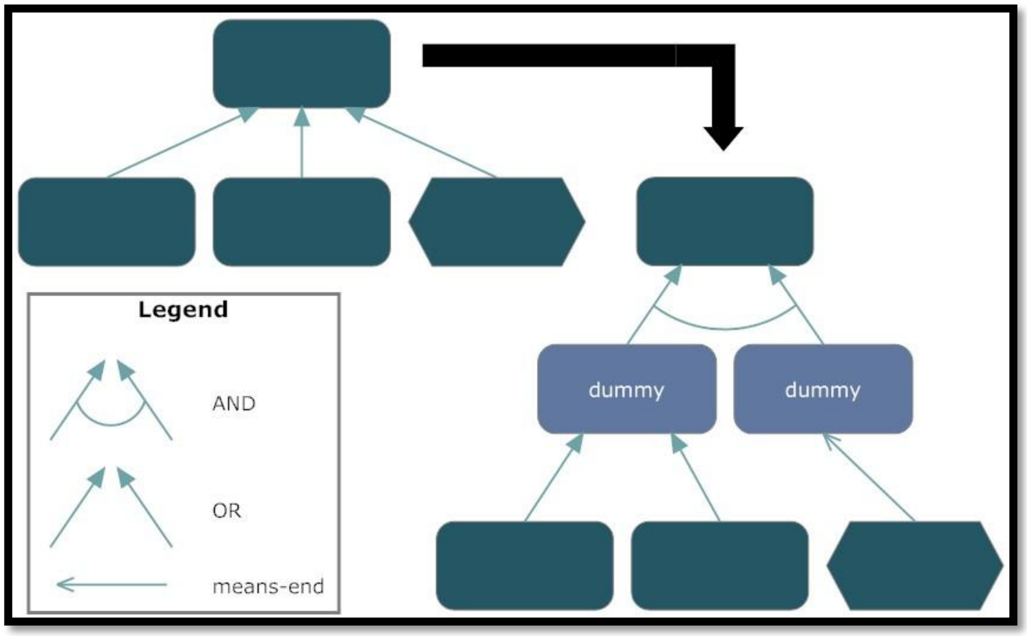


Figure 4-7: Modelling relationship combinations in the diminished Tropos model

4.6 TAOM4E

Tool for Agent-Oriented Modelling (*TAOM*) is a plug-in for Eclipse (*4e*) that supports all phases of the Tropos methodology. Figure 4-8 shows a screenshot of the *TAOM4e* Interactive Development Environment (IDE). A tool palette provides all the constructs to create the various required diagrams.

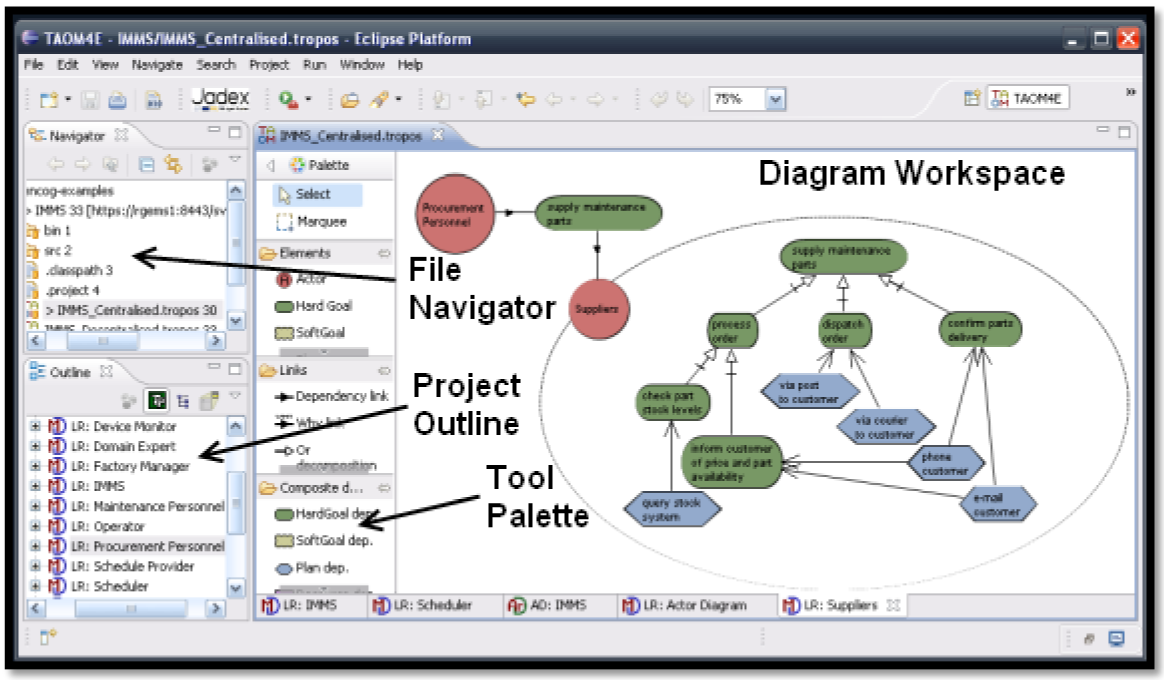


Figure 4-8: Screen capture of TAOM4E Graphical User Interface

TAOM4E is a graphical modelling framework that extends some existing plug-ins [133]:

- **Eclipse Modelling Framework (EMF)** [134] – Offers a modelling framework and code generation functionality for building tools and other applications based on a model described in XML Metadata Interchange (XMI).
- **Eclipse Graphical Editing Framework (GEF)** [135] – Provides the ability to create a graphical editor from an existing application model.
- **Tefkat** [136] – Provides a rule-based language to implement model-to-model transformation.

EMF is used as the basis for implementing the Tropos meta-model and signifies the *TAOM4e* model. The graphical representation and the different views of the model are realised by GEF, which forms the *TAOM4e* platform. *eCAT* (Continuous Agent Testing on Eclipse) [137] implements a method for automated continuous testing of MAS. It supports a goal-oriented testing approach and facilitates test suites from goal analysis diagrams produced with *TAOM4e*. Graphical User Interfaces (GUIs) aid the human tester to specify test inputs and outputs. The *Tefkat* plug-in is used to transform top-level plans and their decompositions into UML (Unified Modelling Language) activity diagrams. Figure 4-9 depicts the architecture of *TAOM4e* and *eCAT*. Any UML editor can be used to edit the resulting diagrams. The diagrams can be further detailed with sequence diagrams, which define communication protocols between agents [133].

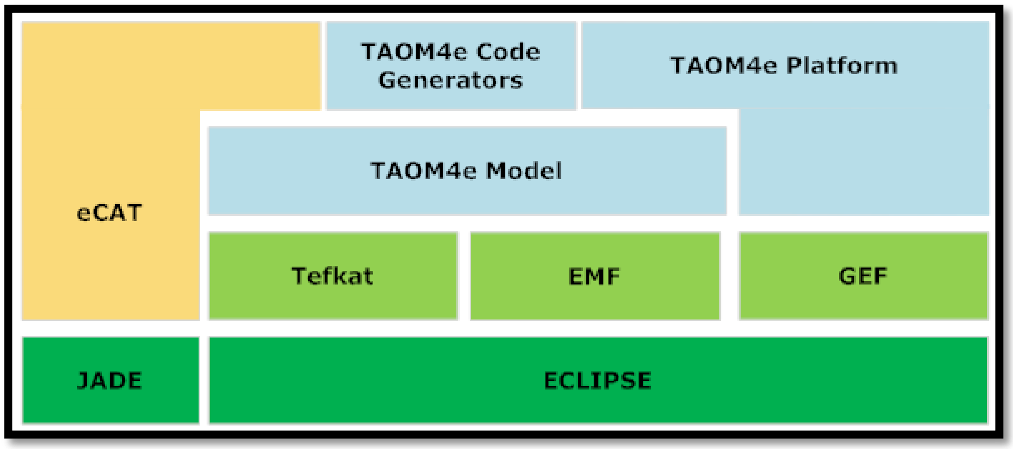
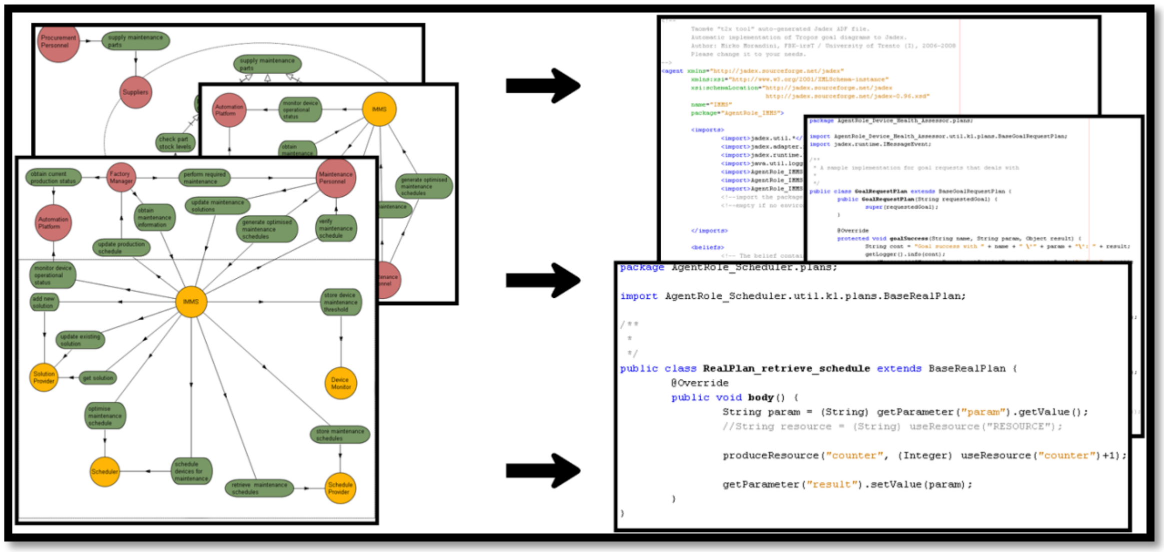


Figure 4-9: Architecture of *TAOM4e* and *eCAT*

Code generation is greatly simplified as the TAOM4e modeller is enriched with *TAOM4e* generators to derive skeletons of code for either the *JADE* [114] or *Jadex* [138] agent platforms. This is done directly from a UML Specification of a detailed design artefact or a Tropos goal model. The *TAOM4e* generators include *UML2JADE*, *t2x* (*Tropos* to *Jadex*) [139], and *Tropos2UML*. *Tropos2UML* is used to generate UML activity diagrams from the Tropos goal model. *UML2JADE* uses UML activity and sequence diagrams that specify Tropos plans (capabilities) to generate code.

"The part of an agent that is responsible for choosing the right plans at runtime in order to reach the desired goals is called *knowledge level* [133]." The *knowledge level* consists of goals and their decompositions, contributions, dependencies to other agents and means-end relationships to plans. These form the input of the *t2x* tool, which generates code skeletons for agents based on the Belief-Desires-Intentions (BDI) architecture. Figure 4-10 illustrates this. The generated code skeletons are executable on the *Jade* BDI (*Beliefs-Desires-Intentions*) agent platform.


Procurement Personnel
supply maintenance parts
Suppliers
supply maintenance parts
check part stock levels
Automation Platform
monitor device operational status
IMMS
obtain maintenance
generate optimised maintenance schedules
obtain current production status
Factory Manager
perform required maintenance
Maintenance Personnel
Automation Platform
obtain maintenance information
update maintenance solutions
generate optimised maintenance schedules
verify maintenance schedule
update production schedule
monitor device operational status
add new solution
IMMS
store device maintenance threshold
update existing solution
Solution Provider
get solution
Device Monitor
optimise maintenance schedule
store maintenance schedules
Scheduler
schedule devices for maintenance
retrieve maintenance schedules
Schedule Provider

```
<!--
        Taom4e "t2x tool" auto-generated Jadex ADF file.
        Automatic implementation of Tropos goal diagrams to Jadex.
        Author: Mirko Morandini, FBK-irsT / University of Trento (I), 2006-2008
        Please change it to your needs.
-->
<agent xmlns="http://jadex.sourceforge.net/jadex"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://jadex.sourceforge.net/jadex
                            http://jadex.sourceforge.net/jadex-0.96.xsd"
        name="IMMS"
        package="AgentRole_IMMS">

        <imports>
                <import>jadex.util.*</
                <import>jadex.adapter.
                <import>jadex.runtime.
                <import>java.util.logg
                <import>AgentRole_IMMS
                <import>AgentRole_IMMS
                <import>AgentRole_IMMS
                <!--import the package
                <!--empty if no enviro


        </imports>

        <beliefs>
                <!-- The belief contai
```

```
package AgentRole_Device_Health_Assessor.plans;

import AgentRole_Device_Health_Assessor.util.kl.plans.BaseGoalRequestPlan;
import jadex.runtime.IMessageEvent;

/**
 * A sample implementation for goal requests that deals with
 *
 */
public class GoalRequestPlan extends BaseGoalRequestPlan {
        public GoalRequestPlan(String requestedGoal) {
                super(requestedGoal);
        }

        @Override
        protected void goalSuccess(String name, String param, Object result) {
                String cont = "Goal success with " + name + " \"" + param + "\": " + result;
                getLogger().info(cont);
```

```
package AgentRole_Scheduler.plans;

import AgentRole_Scheduler.util.kl.plans.BaseRealPlan;

/**
 *
 */
public class RealPlan_retrieve_schedule extends BaseRealPlan {
        @Override
        public void body() {
                String param = (String) getParameter("param").getValue();
                //String resource = (String) useResource("RESOURCE");

                produceResource("counter", (Integer) useResource("counter")+1);

                getParameter("result").setValue(param);
        }
}
```

Figure 4-10: Diagram structure used to generate skeleton code files

The reasoning part of a software agent is implemented by the generated code skeletons. An Agent Definition File (ADF) is an XML file that defines goals, plans, beliefs and messages for every agent in the system. The ADF is an XML representation of the structure of an agent and its relations to other actors. In addition to the ADF, the skeleton java code files are created for the plans and goals, as well as meta-plans and meta-goals. These files share associations to elements in the ADF.

4.7 Jadex Reasoning Engine

Different agent platforms are available for developing multi-agent applications. Most of these platforms are developed with a specific technological focus, such as the cognitive or infrastructural architecture [140]. For this reason, not all aspects of agent technology are covered equally well. The existing platforms can be classified into two almost distinct groups. FIPA-compliant platforms mainly address openness and middleware issues by realising the FIPA communication standards; reasoning-centered platforms focus on the behaviour model of a single agent, e.g. trying to achieve rationality and goal-directedness. This perceived gap between middleware and reasoning-centred systems is one of the main reasons for the realisation of the Jadex BDI (Belief-Desire-Intention) reasoning engine, which aims to merge both research focus areas.

In addition to the core motivation for developing the Jadex BDI platform, the design of the system was driven by two main factors. First, an effort to enhance the BDI architecture in general by addressing various shortcomings of earlier BDI agent systems and, secondly, it respects the current state of the art regarding mainstream object-oriented software engineering methods. The developers designed the system to be used not only by AI experts, but also by normally skilled software developers. "Therefore, agent development with Jadex is based on established techniques such as Java and XML, and is further supported by software engineering aspects, such as reusable modules and development tools [140]."

Figure 4-11 presents an overview of the abstract Jadex architecture. Viewed from the outside, an agent is a black box, which sends and receives messages. Incoming messages or goal events serve as input to the internal reaction and deliberation mechanism, which dispatches the events to plans selected from the plan library. The reaction and deliberation mechanism is the only global component of an agent. All other components are grouped into reusable modules referred to as capabilities. The Jadex architecture consists of [140]:

- **Beliefs** – One objective of the Jadex project is the adoption of a software engineering perspective for describing agents. An object-oriented representation of beliefs is employed, where arbitrary objects can be stored as named facts (called beliefs) or named sets of facts (called belief sets). Operations against the belief base can be issued in a descriptive set-oriented query language.
- **Goals** – Goals are a central concept in Jadex, following the general idea that goals are concrete, momentary desires of an agent. For any goal it has, an agent will more or less directly engage into suitable actions, until it considers the goal as being reached, unreachable, or no longer wanted.
- **Plans** – Plans represent the behavioural elements of an agent and are composed of a head and a body part. The plan head specification is similar to other BDI systems and mainly specifies the circumstances under which a plan may be selected, e.g. by stating events or goals handled by the plan and preconditions for the execution of the plan. The plan body provides a predefined course of action, given in a procedural language.
- **Capabilities** – Capabilities represent a grouping mechanism for the elements of a BDI agent, such as beliefs, goals, plans, and events. In this way, closely related elements can

be put together into a re-usable module, which encapsulates certain functionalities. The enclosing capability of an element represents its scope, and an element only has access to elements of the same scope.

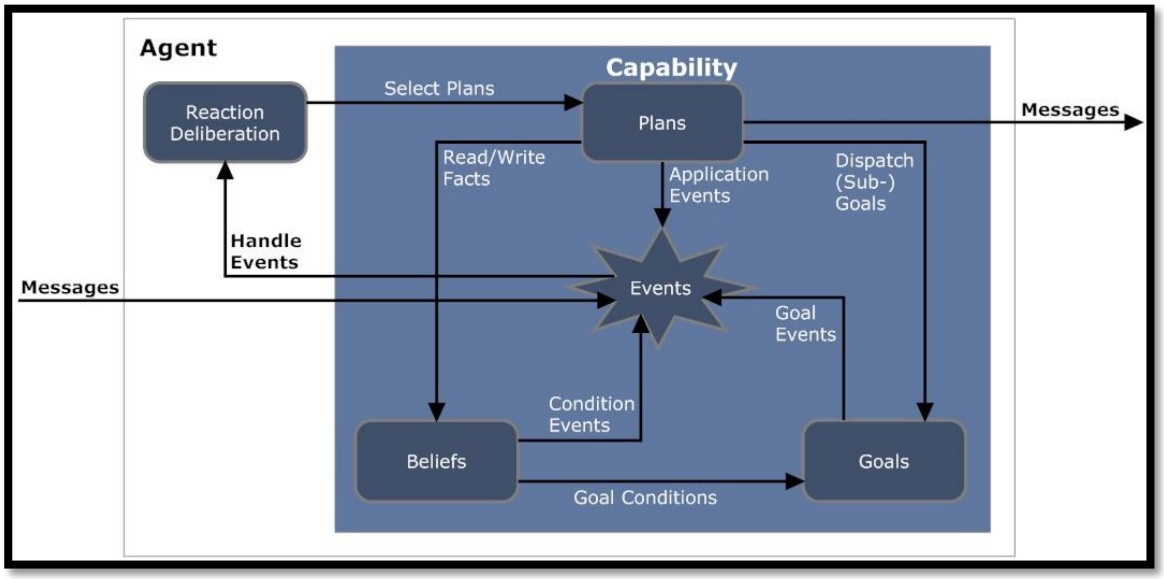
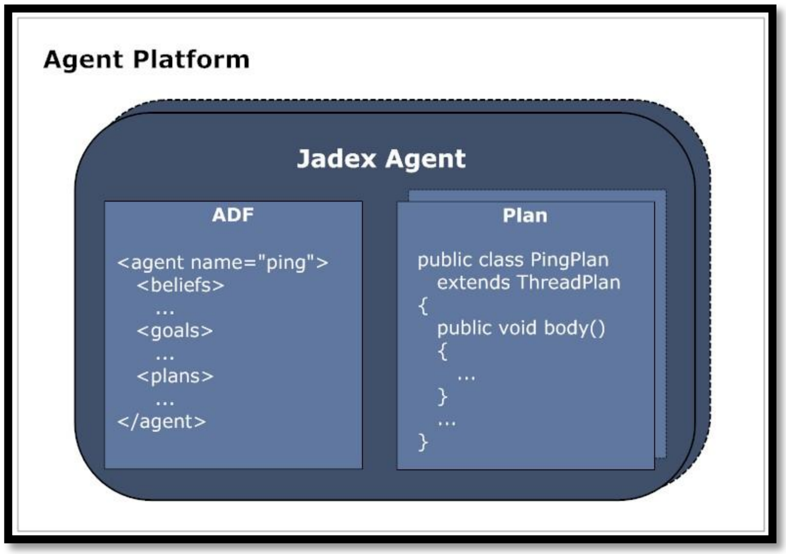


Figure 4-11: Jadex abstract architecture

Jadex is neither based on a new agent programming language nor does it employ or revise an existing one. Instead, a hybrid approach was selected, distinguishing explicitly between the language used for static agent type specification and the language for defining the dynamic agent behaviour [140]. According to this distinction, a Jadex agent consists of two components: An agent definition file (ADF) for the specification of beliefs, goals and plans as well as their initial values in addition to procedural plan code (see Figure 4-12). XML is used for defining ADFs that follow the Jadex BDI meta-model specified in XML Schema. The XML structure specification is augmented by a declarative expression language, e.g. for specifying goal-conditions. The procedural part of plans (the plan bodies) are realised in an ordinary programming language (Java) and has access to the BDI facilities of an agent through an application program interface (API).



```
<agent name="ping">
  <beliefs>
    ...
  <goals>
    ...
  <plans>
    ...
</agent>
```

```
public class PingPlan
  extends ThreadPlan
{
  public void body()
  {
    ...
  }
  ...
}
```

Figure 4-12: Jadex agent structure

The reasoning engine has been realised as a separate component, intentionally limiting the dependencies to the underlying platform. To use the reasoning engine on top of other platforms, an adapter has to be developed, as illustrated by Figure 4-13. This adapter has to implement a handful of methods used by the Jadex engine (e.g. to send messages) and has to call engine functions when it is expected to do the reasoning. Although the current implementation is designed to be used with JADE, the reasoning engine can be easily integrated with other FIPA-compliant agent platforms, given that they provide a similar interface for message handling. It is also possible to use the system in conjunction with other middleware environments such as J2EE or .NET, when FIPA-compliance is not a requirement [140].

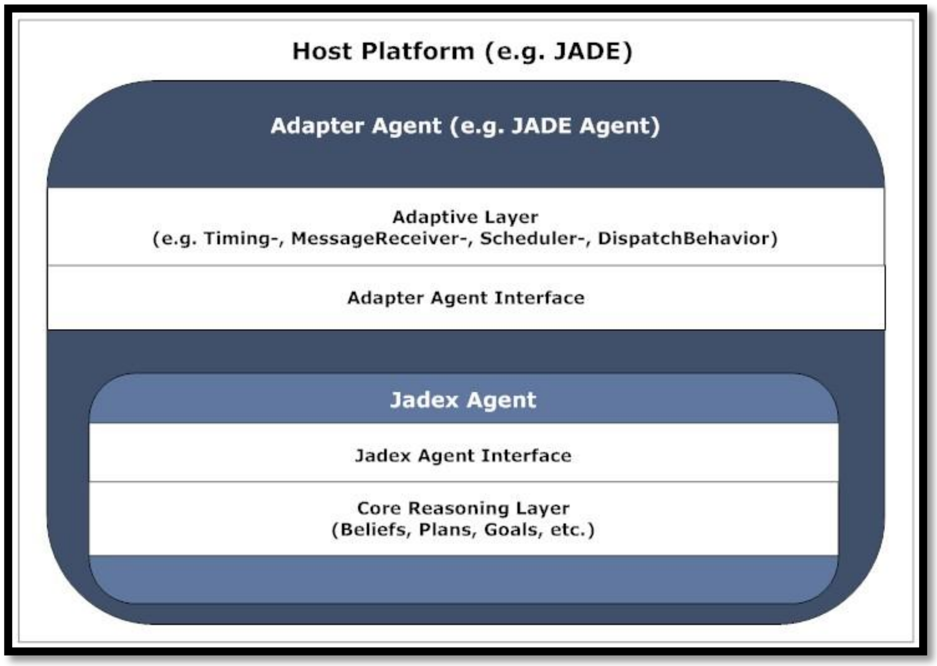

Host Platform (e.g. JADE)
Adapter Agent (e.g. JADE Agent)
Adaptive Layer
(e.g. Timing-, MessageReceiver-, Scheduler-, DispatchBehavior)
Adapter Agent Interface
Jadex Agent
Jadex Agent Interface
Core Reasoning Layer
(Beliefs, Plans, Goals, etc.)

Figure 4-13: Jadex platform integration

4.8 Conclusion

This chapter closely examined several aspects related to Multi-Agent Systems (MAS). Several applications of MAS were discussed and the variety of the different application domains was noteworthy. Some of the more prominent AOSE methodologies were reviewed. It was decided to use Tropos, with some proposed restrictions, as the designated methodology. This decision was then followed by a more detailed account of the methodology. TAOM4e was introduced and some of its functionality was highlighted. The runtime environment or more specifically the Jadex reasoning engine was also discussed as it is the final environment in which the implemented system will exist. The next chapter presents the design of the IMMS using the Tropos methodology.

Chapter 5 IMMS Design Using Tropos

5.1 Introduction

This chapter contains the design of the IMMS using Tropos and TAOME4e as explained in sections 4.4 and 4.6, respectively. It is important to note that the design was conducted taking into consideration the restrictions proposed in 4.5. For this reason, the *detailed design* phase of the Tropos methodology is not performed. Instead, the code is generated from the *architectural design* diagrams. The code generated is then completed in an interactive development environment (IDE) to suit the requirements of the system. The remainder of the chapter covers the *early requirements*, *late requirements* and *architectural design* of the Tropos methodology for the design of the system. Some diagrams of the separate phases are omitted in an effort to reduce redundancy.

5.2 Early Requirements

As the Tropos methodology specifies, the *early requirements* phase identifies all the stakeholders of the system. This models the system in its entirety according to the current configuration. The reasons for modelling the actors are to better understand the relationships and interactions between the actors and the processes that take place in the system. It is important to note that the actors of the IMMS do not represent the entire company but all the actors relating (directly or indirectly) to maintenance. Another important consideration is that the actors can represent human individuals, software- or hardware systems. Once the role-players have been indentified the dependencies between them are modelled. These dependencies signify the relationship between the actors. Figure 5-1 provides the early requirements diagram of the IMMS. The following stakeholders were identified:

- ***Domain Expert*** – This actor represents the specialists on hardware devices that are present in the system – usually the manufacturer or the consultants. This actor is consulted by the maintenance personnel in the event that a problem arises which the maintenance personnel themselves cannot solve. The domain expert also performs an advisory capacity role to answer any other queries raised by the maintenance personnel.

- ***Factory Manager*** – The factory manager is the actor that manages all the processes relating to assembly and production. This actor is also responsible for production planning and is assisted by the operator and automation platform actors.
- ***Automation Platform*** – This actor represents the physical automation platform. It provides an interface for other actors to interact with the automation platform and control processes that govern assembly or production.
- ***Operator*** – The operator represents the plant operator. This actor is responsible for monitoring the production process and ensuring normal operation. Another responsibility of the operator actor is to ensure that the production plan is followed and correctly executed.
- ***Maintenance Personnel*** – The maintenance personnel are responsible for maintaining the physical components in the assembly system. These individuals perform any maintenance-related tasks. Maintenance-related problems that cannot be solved by the maintenance personnel are referred to the domain expert. The number of parts used during maintenance operations is logged so that the stock can be replaced. The maintenance personnel rely on the procurement personnel to acquire the required parts from the suppliers.
- ***Suppliers*** – This actor represents the supplier of maintenance parts. It is present to model interactions between suppliers and the company. The supplier’s sole interaction with the company occurs via the procurement personnel.
- ***Procurement Personnel*** – The procurement personnel are responsible for sustaining the necessary stock levels required by the maintenance personnel. The part usage is reported by the maintenance personnel and is used to place proactive part orders with the supplier. This ensures that whenever maintenance is to take place, the necessary parts are available.

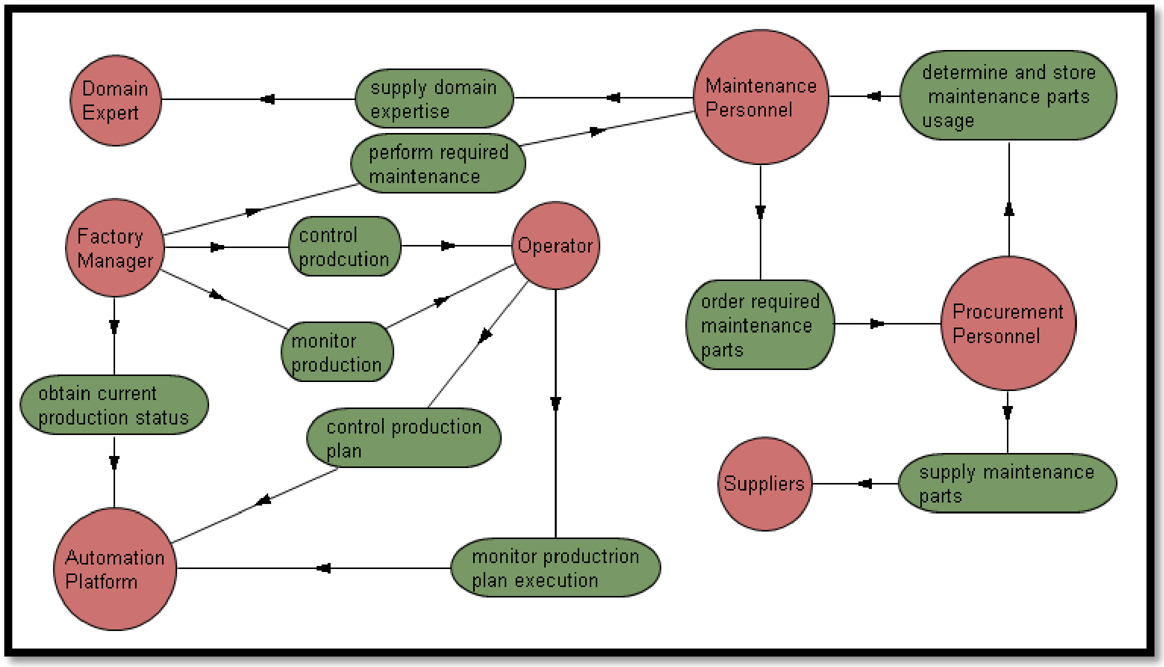


Figure 5-1: IMMS early requirements diagram

The goals of the individual actors are not discussed. These are typical entities you would find in a business specialising in assembly. The actors modelled here do not represent the system being designed. They will interact with the system being designed but not form part of it. Thus modelling their internal goal diagrams is not required for implementation. It may also be found that the stakeholders identified in this instance are not always present in every industrial automation system. This is one of the characteristics and reasons for doing requirements analysis for every system to be developed. Even though requirements analysis is an iterative process, it is important not to underestimate the very early stages. As the design progresses a transition from abstract to more concrete concepts is encountered between the different stages. The general heuristic is that the earlier steps in the design (such as early- and late requirements) are easier to alter than the later steps. It is thus crucial not to underestimate the importance of requirements analysis.

5.3 Late Requirements

The end-result of *early requirements* analysis is a model of the current system configuration. The next step is *late requirements*. The main goal of *late requirements* is to introduce the *IMMS*, the system being designed, as a single actor in the environment. This is done to model the relationships, dependencies and interactions between existing actors and the newly introduced system-to-be actor. Once the system-to-be actor is introduced, the expected requirements start to emerge. Figure 5-2 depicts the *late requirements* diagram of the IMMS system. The system-to-be actor is represented by the orange circle and is easily distinguished from the existing actors.

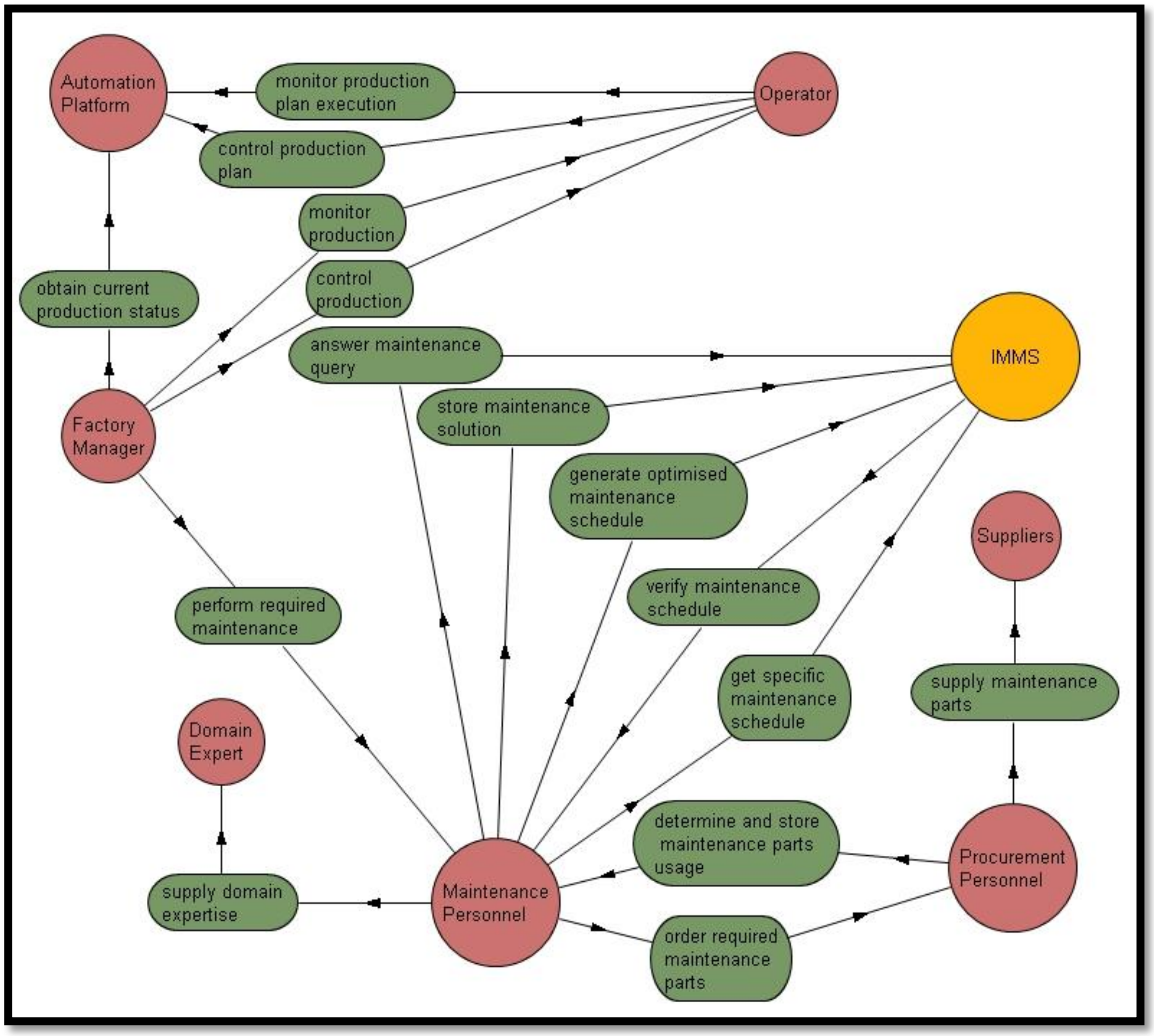

Automation Platform
monitor production plan execution
Operator
control production plan
monitor production
control production
obtain current production status
answer maintenance query
IMMS
store maintenance solution
Factory Manager
generate optimised maintenance schedule
Suppliers
verify maintenance schedule
perform required maintenance
get specific maintenance schedule
supply maintenance parts
Domain Expert
determine and store maintenance parts usage
Procurement Personnel
supply domain expertise
Maintenance Personnel
order required maintenance parts

Figure 5-2: Late requirements actor diagram

The introduction of this new actor establishes a new set of goals that signify the main goals of the system. The identified goals of the system are discussed in Table 5-1. These goals signify the dependencies between the *IMMS* actor and the other actors and indicate the type of relationship existing between it and the other actors of the system. In the later stages of the design these goals will be delegated to a number of possible sub-actors of the *IMMS*. They will also be decomposed into sub-goals and means-end relationships with plans, depending on the situation and requirements thereof.

Table 5-1: Description of the main goals of the system-to-be

System Goal	Description of Goal
answer maintenance query	The system attempts to answer maintenance queries posed to it mainly by the *maintenance personnel*. Maintenance queries in this sense are previous solutions to maintenance scenarios that were logged by the maintenance team after solving them. In a situation where the system is unable to answer the query, the *maintenance personnel* then consult the *domain expert*.
store maintenance solution	Any solutions to maintenance related issues are logged by the *maintenance personnel* and it is the responsibility of the IMMS system to store these solutions in a suitable data storage medium.
generate optimised maintenance schedules	The IMMS is commissioned to generate maintenance schedules for the devices in the automation platform. These schedules should also be optimised to overlap suitable tasks, minimise downtime and increase production. Although these criteria are easy to comprehend, implementing them is another matter entirely.

System Goal	Description of Goal
verify maintenance schedule	Once the maintenance schedules are generated they should be verified by a member or members of the *maintenance personnel*. It is imperative that this step is performed by a human being as the logical schedule for device maintenance is not always correct. For this reason a human expert on maintenance scheduling is consulted as a final step of verification.
get specific maintenance schedule	Once schedules are generated and have been verified, they are stored. These schedules can then be requested and the *IMMS* agent should then retrieve them from the designated storage medium and present them as requested. These schedules will be uniquely identifiable to simplify requests.

Each of the actors has an internal goal diagram. This goal diagram illustrates how the main goals are decomposed into sub-goals and means-end relationships with plans at the lowest level. Figure 5-3 is an example of such a diagram. All the actors defined in the system have an internal goal diagram. Even though most of these actors are already in existence and will therefore not be designed and implemented as part of the system, it is still worthwhile to model them in order to fully understand how the current configuration operates and how the system should fit into this. The plans in the means-end relationships of the lower level goals are the way that the goals are to be achieved. These are also the constructs that will contain most of the code to be executed by the agent.

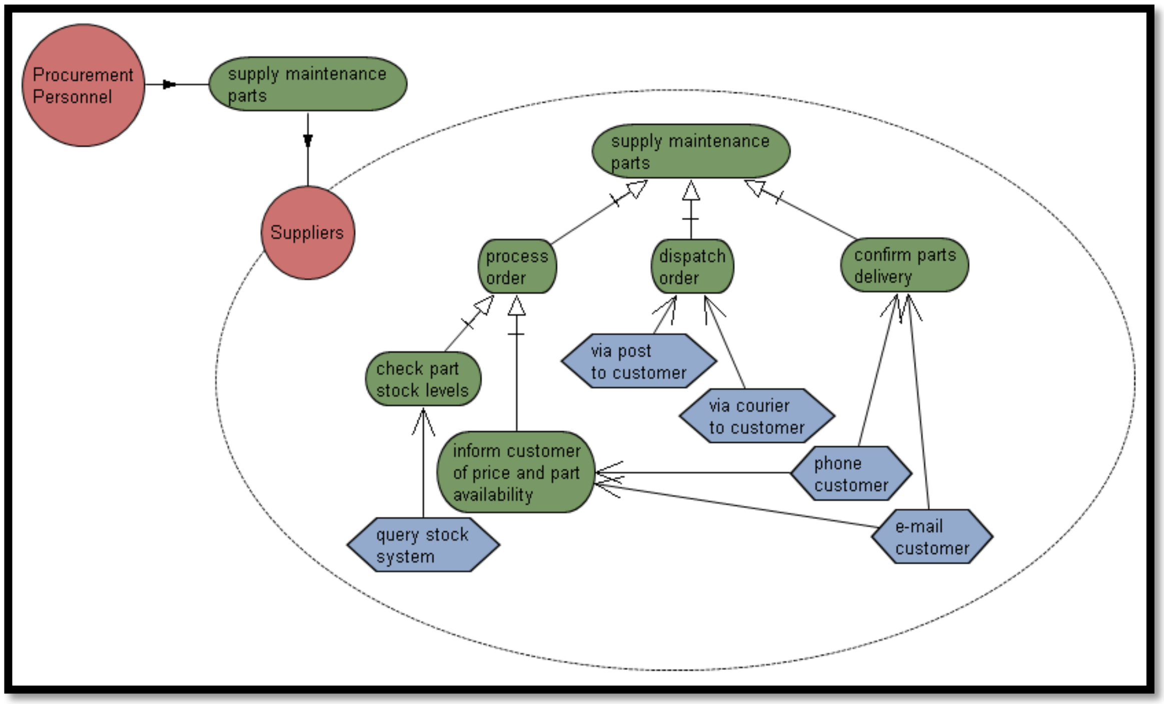


Figure 5-3: Supplier actor internal goal diagram

5.4 Architectural Design

The *Architectural design* phase follows logically after *late requirements*. During *architectural design*, the objective is to decompose the system-to-be (*IMMS*) actor into sub-actors that fulfil *IMMS* actor functionality. The *IMMS* then becomes a mere design artefact (representing the combination of the sub-actors) and all interactions between it and other actors are delegated to the sub-actors. The reason for this decomposition is to create sub-actors that are specialised for a specific function in the operational scheme. Splitting the *IMMS* actor into sub-actors also has the benefit of segregating the messages intended for a single agent between multiple agents and in so doing avoiding a possible message-bottleneck. Troubleshooting is also substantially simplified by decomposing the main actor into sub-actors as each of the sub-actors represents a core functionality of the original *IMMS* agent. Isolating a fault is simply a matter of identifying the function that is found to be problematic. Figure 5-4 depicts the *architectural design* actor diagram.

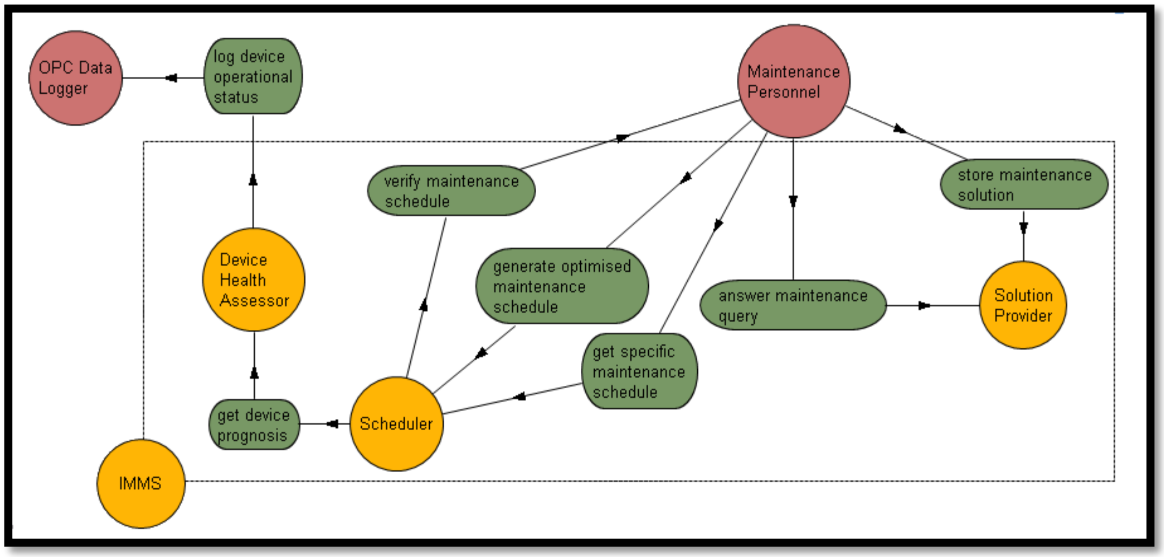

OPC Data Logger
log device operational status
Maintenance Personnel
verify maintenance schedule
store maintenance solution
Device Health Assessor
generate optimised maintenance schedule
answer maintenance query
Solution Provider
get specific maintenance schedule
get device prognosis
Scheduler
IMMS

Figure 5-4: Architectural design actor diagram

Three sub-actors were identified for the *IMMS* actor: *device health assessor*, *scheduler* and *solution provider*. As mentioned earlier and confirmed by Figure 5-4, the goals previously assigned to the *IMMS* agent are now delegated, depending on the type of goal, to a specific sub-actor. The internal goal diagram of these actors is important, as these agents will physically be implemented. The remainder of section 5.4 is devoted to the discussion of these sub-actors, the roles they fulfil in the system and detailing their goal decomposition.

5.4.1 Device Health Assessor

The role of the *device health assessor* actor is to analyse the degradation of devices in the automation platform. Figure 5-5 depicts the internal goal diagram of the *device health assessor* actor. The data used for this analyses are provided by historical logs stored in a database and live information provided by the OLE for Process Control (OPC) connection to the physical devices. As the goal is to determine device prognosis, the main source of information is the historical database. The aim is to establish the degradation trend and to determine when an unacceptable level is reached, or will be reached, and to schedule the device for maintenance before that point. The *device health assessor* does not detect instantaneous or catastrophic failures, but the reduced fatigue due to well-timed and adequate maintenance can reduce the occurrences thereof.

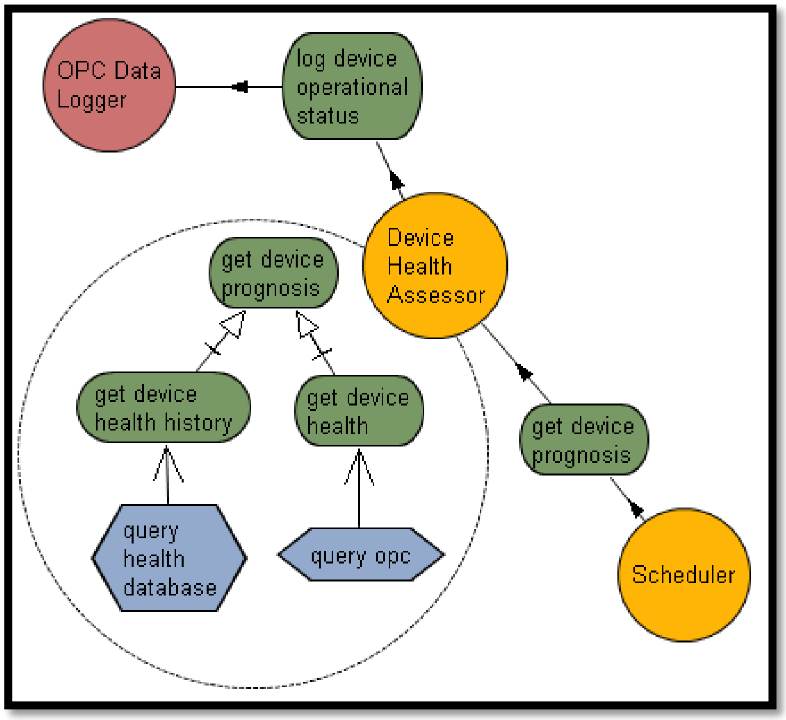


Figure 5-5: Device health assessor internal goal diagram

Figure 5-5 also illustrates the goal that is delegated to the *device health assessor* by the *scheduler*, as well as the goal that is delegated to the *opc data logger* by the *device health assessor*. The purpose of the *get device prognosis* goal is to provide the necessary information required by the *scheduler* actor to enable it to generate the maintenance schedules. This goal is central to the operation of the system. The *log operational status* goal logs the results of the analyses done by the *device health assessor* and stores them in a database.

5.4.2 Scheduler

The main function of the scheduler actor is to provide maintenance schedules to the rest of the system. All schedule management-related operations are also performed by this actor. Figure 5-6 illustrates the internal goal decomposition and goal delegation to other actors. The scheduler agent depends on the *device health assessor* for the completion of the *get device prognosis* goal. The device prognostic information is vital to the schedule generation process.

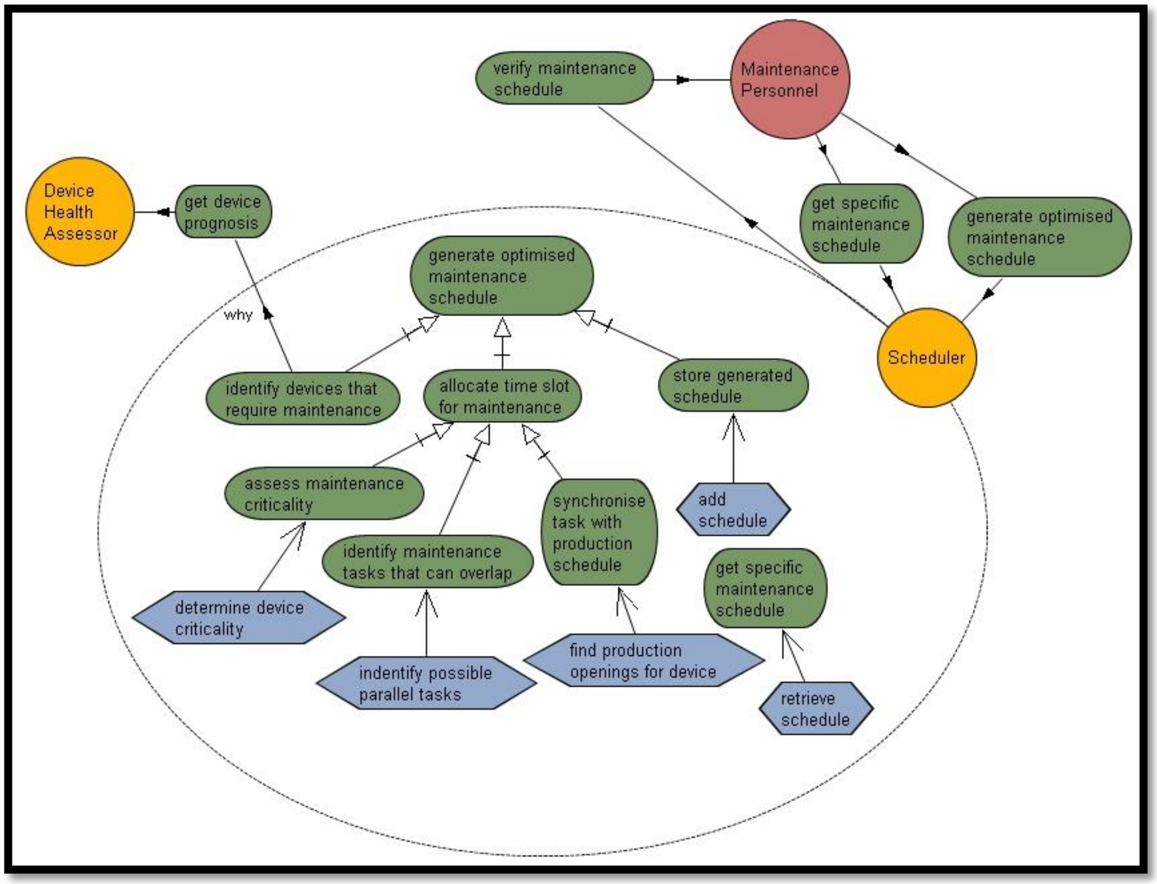

verify maintenance schedule
Maintenance Personnel
Device Health Assessor
get device prognosis
get specific maintenance schedule
generate optimised maintenance schedule
generate optimised maintenance schedule
why
Scheduler
identify devices that require maintenance
allocate time slot for maintenance
store generated schedule
assess maintenance criticality
synchronise task with production schedule
add schedule
identify maintenance tasks that can overlap
get specific maintenance schedule
determine device criticality
find production openings for device
indentify possible parallel tasks
retrieve schedule

Figure 5-6: Scheduler actor internal goal diagram

Maintenance personnel and the *scheduler* rely on each other to fulfil their duties. The *maintenance personnel* rely on the *scheduler* to automate the generation of maintenance schedules. These generated maintenance schedules need to be verified by the *maintenance personnel*. Only verified schedules are valid and are subsequently stored to be used. In addition, to simply generate the schedules the *scheduler* also attempts to optimise the schedules by synchronising the schedule tasks with the production schedule. The *scheduler* identifies maintenance tasks that could possibly be executed concurrently and attempts to schedule them accordingly.

5.4.3 Solution Provider

The *device health assessor* and *scheduler* actors accomplish the primary objectives of the IMMS. The *solution provider* fulfils the secondary role of managing maintenance solutions.

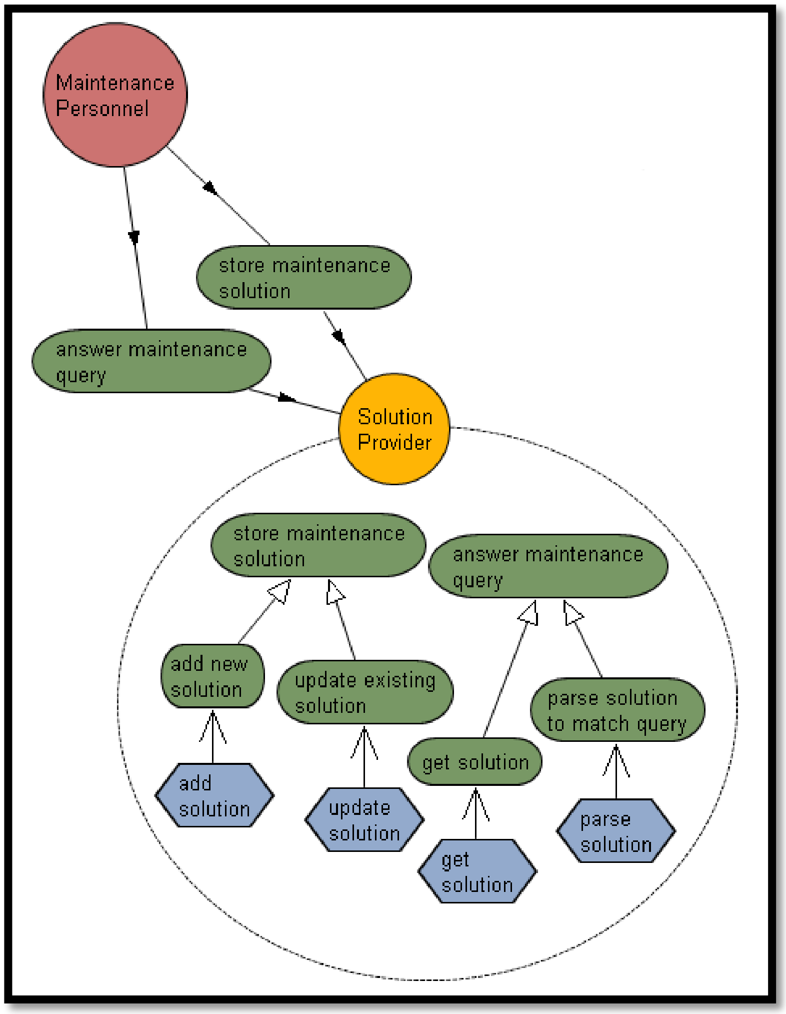


Figure 5-7: Solution provider internal goal diagram

5.5 Conclusion

This chapter examined the design of the IMMS system and the associated agents. The Tropos design steps were followed as discussed in section 4.4. In early requirements the domain stakeholders were introduced and the inter-relationships and dependencies were modelled. The system-to-be actor was added in late requirements and with it some additional goals and dependencies. In the architectural design, the system-to-be actor was decomposed in a number of sub-actors. These sub-actors combined fulfil the same functionality as the system-to-be actor, thus reducing the latter to merely a design artefact. The remainder of the chapter dealt with the sub-actors' functionality and their internal goal and plan decompositions. The next chapter examines a simulation scenario in which several previously discussed topics are incorporated to fulfil the functionality of the IMMS.

Chapter 6 IMMS Simulation

6.1 Introduction

This chapter examines a simulation scenario which incorporates concepts from previous chapters. The primary reason for deciding to create a simulation in favour of a real implementation is due to limitations imposed by the research lab setup. The RGEMS lab, as discussed in Chapter 2, is (at the time of writing) in the process of becoming a fully functional reconfigurable assembly system (RAS). When the assembly is operational, data logging applications will ensure that sufficient operational data is logged for several research endeavours, including an implementation of an Intelligent Maintenance Management System (IMMS).

The current primary dilemma is that devices do not operate over their normal maintenance lifecycle. The system is only run for brief periods of time to test new PLC/robot programs, camera settings, pneumatic configuration changes, etc. After the sufficient amount of run-time has been achieved the system is switched off. None of the devices are operated under realistic factory conditions for extended periods of time which would allow for operational data to be logged and used in research.

This limitation of the RGEMS lab was the reason for several attempts at contacting individuals in industry and research groups conducting similar research or in a position to provide the required maintenance measurement data. Such data would have been a more accurate representation of what is encountered in industry. Unfortunately, none of the attempts were successful and most resulted in failure to even respond. Initially some individuals responded with interest in the research being conducted, but failed to supply any data once it was discussed what was required. These responses, or rather lack thereof, was somewhat discouraging when considering the fact that the research being conducted could ultimately benefit both researchers and industry.

As stated previously, when the upgrades and new installations are complete, the RGEMS system will operate as a fully functional assembly system. For this reason it was decided to settle for a controlled simulation to test certain concepts.

6.2 Simulation Scenario

This section will briefly discuss the scenario which this simulation aims to achieve. Usually a device will have a maintenance guide in the form of hours worked or operations performed. This is not accurate in all situations as the same device, be it a robotic arm, can be used in different configurations. The work this device will perform will vary from configuration to configuration. The idea is to determine how strenuously a device has been operated for a set period of time. This information is then used to augment the maintenance interval provided by the manufacturer as a guide. The devices are prioritised based on certain criteria as well as a maintenance threshold that is calculated based at what levels the device was operated, as well as the period at each load level. These thresholds and priorities are then used to schedule the devices for maintenance. Figure 6-1 illustrates the main components and interactions of the simulation.

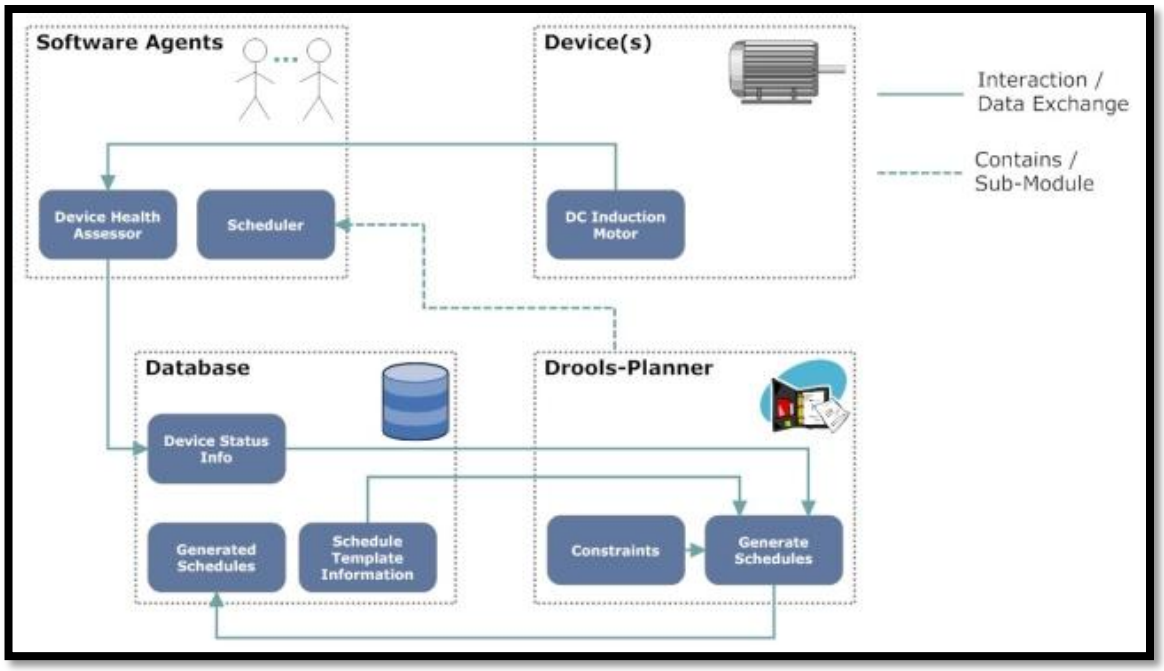


Figure 6-1: Main components and interactions of the simulation

Data indicating the prognostic state is collected for a device, in this case a DC (Direct Current) induction motor. This data is then processed and formatted as required, after which it is stored in the database. The scheduling component then retrieves the device information from the database, along with the maintenance schedule template. This template specifies which days and time slots are available for maintenance to be scheduled. The scheduler component then proceeds to iteratively generate the schedule using the schedule template. This process continues until set criteria are reached, which can be a time limit or some measurement (possibly constraints) indicating that an acceptable schedule has been reached, or a combination of both. After the scheduler concludes, the generated schedules are once again stored in the database to be used as required.

6.3 Physical Setup

6.3.1 Data Collection

The data for this simulation was collected from a DC induction motor in one of the practical labs at the CUT. The motor is a 500V, 10kW, 20A rated- DC motor with armature resistance of 1 ohm. When supplied at 500V, the unloaded motor runs at 1040 rev/min, drawing a current of 0.8A (ideally current is zero at no-load).The motor drives a generator that powers rows of 100W light bulbs. Figure 6-2 displays the motor and the generator used to drive the bulbs. There are four rows of bulbs, as illustrated in Figure 6-3, and the rows can be switched individually. Each row contains five bulbs and can be viewed as a load level of 25%. Data was collected for the following load settings:

- **50% Under Load** – None of the rows of bulbs were switched on in this scenario.
- **25% Under Load** – One of the four rows was switched on.
- **Normal Load** – Two of the four rows were switched on.
- **25% Overload** – Three of the four rows were switched on.
- **50% Overload** – All four rows of bulbs were switched on.

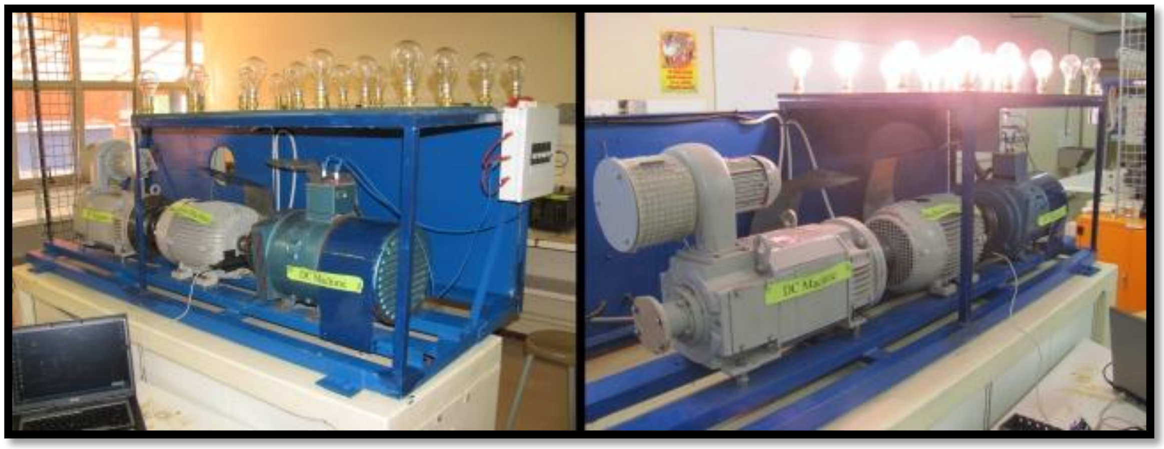


Figure 6-2: DC induction motor driving generator

For this simulation the values considered part of the transition period that elapsed after the load level was adjusted, were not taken into account. It was determined (over a four-hour test run at full load) that it takes approximately 30 minutes for the measurement values to stabilise after the load level was altered. For each load level listed above, the motor was given 30 minutes to stabilise and then another 60 minutes period of measurements were logged.

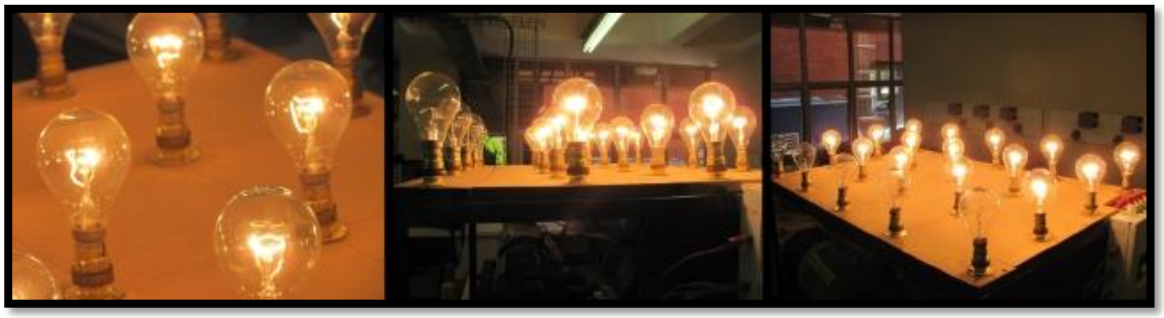


Figure 6-3: Rows of bulbs simulating different load levels

Two temperature sensors were used to measure the ambient temperature and the temperature of the motor right by the bearings. The measurements were taken in one second intervals as this was sufficiently accurate for the simulation. The ambient sensor was important in order to obtain the difference in device and ambient temperature. Rows of 100W bulbs can generate a considerable amount of heat over time. At first LM35 integrated-circuit temperature sensors were used for the measurements but these sensors are not shielded from any interference. This can be problematic as one was to be placed on an induction motor that emits considerable amounts of electromagnetic interference. Thus it was decided to substitute the temperature sensors for more accurate and better shielded PT100 platinum resistance thermometers (PRTs).

A National Instruments (NI) CompactRIO (cRIO) 9014 real-time (RT) controller was used for recording the temperature. The cRIO-9014 has a 400 MHz processor and 2GB solid-state storage which can be used for software installed on the module as well as logging values in some form or another. A 4-channel universal C series analogue module (NI 9219) was used for the measurements. This module has pre-configured settings for temperature measurements with a PT100 (three- and four wire configurations). Figure 6-4 shows the cRIO, analogue module and temperature sensor. It also has four sample settings: *High speed*, *Best 60 Hz rejection*, *Best 50 Hz rejection* and *High resolution*. For the simulation the *high resolution* setting was selected, which gave very accurate and stable temperature measurements by cancelling out the electromagnetic noise generated by the DC motor.

The configuration of NI LabVIEW, the software in which the temperature measurement project was set up, allowed for a distribution of different parts of the projects. It is possible to execute functionality on the CompactRIO, the built-in Field-Programmable Gate Array (FPGA) and a computer or laptop connected to the CompactRIO (via Ethernet, USB or RS232). The reason for this is that time-critical operations are executed on the RT module and time-insensitive operations (such as file I/O or user interfaces) are executed on the computer. In this set-up the RT module measured the temperature by averaging the values of the preceding second (moving/running average) and storing this value as a Comma Separated Value (CSV) in a file that can be opened by spreadsheet compatible software, such as Microsoft Excel. The user interface program that displayed the current temperature as well as a graphically depicting the historic values (up to two minutes), was executed on the laptop connected to the cRIO via Ethernet. The user could specify the length of time that should be logged and the application would automatically create a file with the time and date in the name of the file and automatically finalise the file after the specified amount of time had expired. These files were stored on the cRIO and retrieved later.

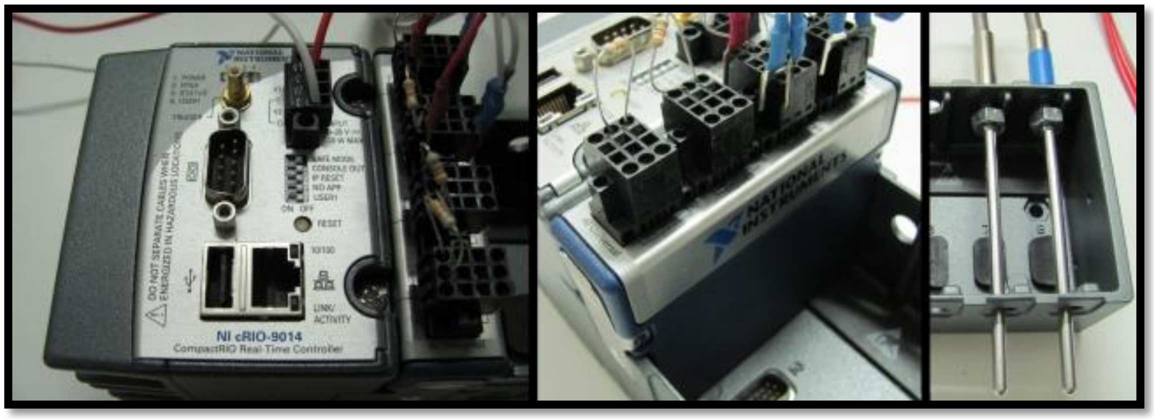


Figure 6-4: CompactRIO with analogue module and PT100 temperature sensors

The input voltage, current and frequency were measured using a Fluke 43B Power Quality Analyser at one second intervals. The Fluke meter, shown in Figure 6-5, has a similar functionality as the cRIO in that values can be logged and stored on the device. These values can then be downloaded later to a computer or laptop using the provided optic-to-USB cable. An alternative to logging the data on the device is to connect to a device with a USB port and store the measurements as they are taken. This has the advantage of larger storage space for measured values. The latter method was used in the current scenario. The Fluke meter was connected via the USB cable to a laptop. The values were logged using the software provided with the meter. Once the measurements were logged they were exported to a CSV file. Both the data from the cRIO and Fluke meter were saved in CSV files. This was done so that it could easily be entered into a database and be used in the rest of the simulation.

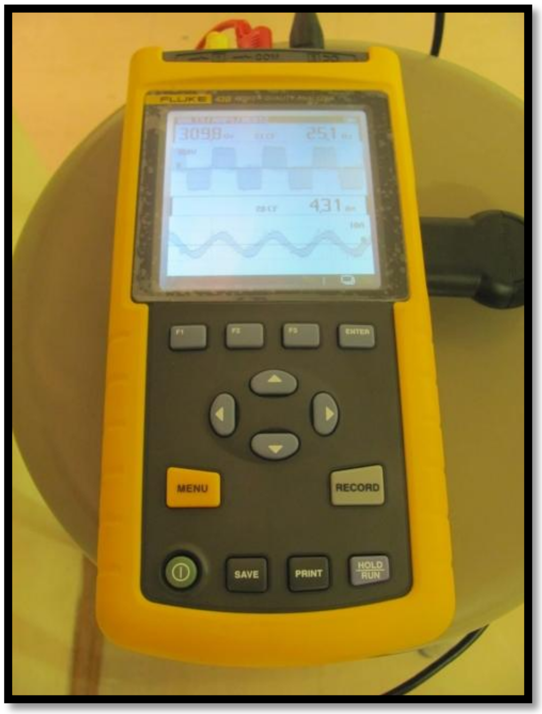


Figure 6-5: Fluke power quality analyser

6.3.2 Simulation Concepts

6.3.2.1 Motor Priority

The data collected from the DC induction motor in 6.3.1 is used to simulate 50 motors. These motors have been functioning at different load levels and different priorities. Priority in this scenario is defined as the product of three parameters: *Need Urgency*, *Customer Rank* and *Equipment Criticality* (NUCREC) [141]. This is an improved method over the standard First-In-First-Out (FIFO) or first-come-first-served approaches more commonly used. NUCREC improves on the Ranking Index for Maintenance Expenditures (RIME) in several ways:

- The customer rank is added.
- The most important item is given the number-one rating.
- The number of ratings in the scale may be varied according to the needs of the particular organisation.

- Part essentiality may be considered.

A rating system of numbers 1 to 4 is recommended. Since most human beings think of number 1 as the first priority to get done, the NUCREC system does number 1 first.

Need urgency ratings include the following:

1. Emergency; safety hazard with potential further damage if not corrected immediately; call back for unsatisfactory prior.
2. Downtime; facility or equipment is not producing revenue.
3. Routine and preventive maintenance.
4. As convenient, cosmetic.

The customer ranks are usually as follows:

1. Top management.
2. Production line with direct revenue implications.
3. Middle management, research and development facilities, frequent customers.
4. All others.

The equipment criticality ratings are as follows:

1. Utilities and safety systems with large area effect.
2. Key equipment or facility with no backup.
3. Most impact on morale and productivity.
4. Low, little use or effect on output.

The product of the ratings gives the total priority. That number will range from 1 (which is 1 x 1 x 1) to 64 (4 x 4 x 4). The lowest number signifies the first priority. A '1' priority is a first-class emergency. "With these predetermined evaluations, it is easy to establish the priority for a work order either manually by taking the numbers from the equipment card and the customer list and multiplying them by the urgency or by having the computer do so automatically [141]." Naturally, there may be a few situations in which the planner's judgment should override and establish a different number, usually a lower number so that the work is performed faster.

6.3.2.2 Maintenance Threshold

In addition to priority, another metric is also used to determine the scheduling order and frequency of motors. This metric is referred to as a *maintenance threshold*. The maintenance threshold indicates how far the device has progressed in nearing its specific maintenance 'due date'. It is important to note at this juncture that the determination of this threshold has been simplified for the implementation of the simulation. It is advised for real-world scenarios and implementations that the designer employs a method (or combination of methods) such as those discussed in 3.3. The device for which the maintenance threshold is established will determine the most appropriate algorithm or technique. It is wise to implement the methods in such a way that they are interchangeable. This will provide the implementation system with greater flexibility to cope with various different types of devices on which maintenance is to be performed.

For this simulation the maintenance threshold was determined by taking the time units a motor was operational and multiplying these by the normal operational value, which in this case was load level three. This gives a base value to be used in a calculation to determine the percentage of time that has elapsed towards the full maintenance term. Figure 6-6 depicts a situation where a motor was operated at different load levels ranging between load level two and load level four for a set period of time units (ten in this case). The y-axis represents the load levels and the x-axis equates the time units. In this example the motor has a maintenance term of 30, which is calculated by multiplying the normal load value (three) by the amount of elapsed time units (ten). The sum of the actual load levels is equal to 29. It is then simply a matter of calculating the percentage as follows: ${}^{29}/_{30} \times 100 \cong 97\%$. Note that 100% would signify the operational time as per the manufacturer guidelines for the specific device. The maintenance threshold can and would in all probability be less than 100%. The specific maintenance threshold would vary from device to device as well as the requirements and maintenance model of the factory in which the device resides. It could be specified that the maintenance threshold is 85%, for example. This would require that all devices be maintained when their individual maintenance thresholds are close to 85%. Even if, for some reason, a device is only maintained when its threshold approaches 95% it is still safely within the manufacturers maintenance recommendations. For this simulation it was decided to use what would be the manufacturers recommended maintenance period of 100% for the maintenance threshold.

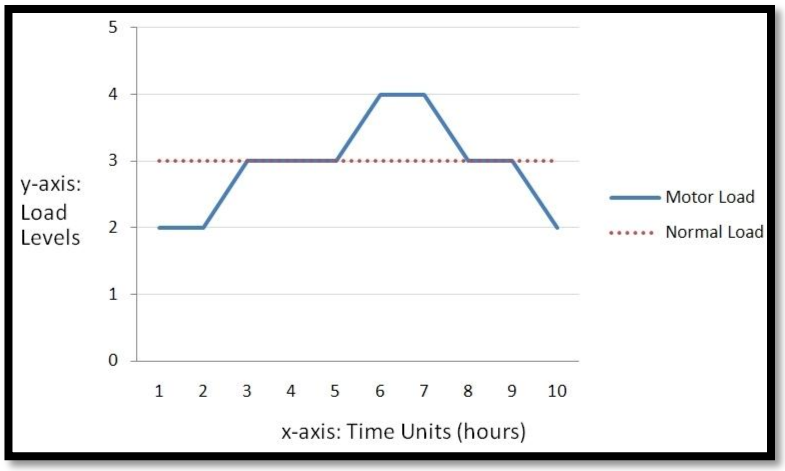


Figure 6-6: Simulation load levels of a motor

6.3.2.3 *Maintenance Personnel*

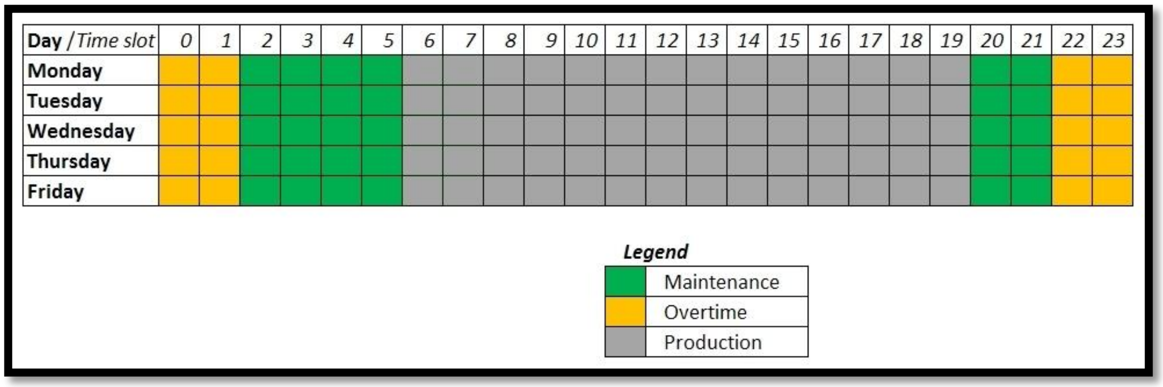
Devices that require maintenance constitutes only one part of the maintenance process. Human technicians are required to perform the physical tasks. The human element requires that certain ethical requirements be taken into consideration. One such requirement is the legal regulations associated with work hours. The department of labour usually determines what is fair when it comes to working hours and overtime. This is to protect the employees and companies involved by clearly stating the legal requirements. For this simulation the regulations as specified by the South African Department of Labour (DoL) [142] were used as a guideline. The employee and employer enter into an agreement that specifies the normal working hours. The employer may require the employee to work a certain amount of additional hours in the form of overtime. Overtime, according to the DoL, is one-and-a-half (1.5) times normal pay. An employee may work a total of three (3) hours overtime per day and 12 hours of overtime in a seven (7)-day period. Note that special circumstances do exist for certain employees (for example, emergency services employees) or if the employee and employer agree on different terms. The standard laws regarding overtime applied for this simulation.

Maintenance personnel are required to perform maintenance for a set amount of time per day and allowed a certain amount of overtime per day and per week according to the abovementioned regulations. Four maintenance technicians are available to be scheduled for maintenance tasks in this simulation. The next section examines the physical maintenance schedule and discusses normal maintenance, production and overtime.

6.3.2.4 Schedule

The maintenance schedule will depend on various elements such as the production schedule, the maintenance policy of the company, available maintenance personnel, the physical state of the devices, to name a few. This section introduces the schedule for this simulation. It is first assumed that no maintenance or production takes place over weekends (Saturday and Sunday). It is likely that most real assembly systems will operate everyday for the maximum amount of time per day.

A simplified schedule is used in this simulation as it serves only to test some concepts. The schedule is the same for all working days (Monday to Friday) of the week and is illustrated in Figure 6-7. Production occurs for a period of 14 hours a day, from 6am until 8pm. During this time no maintenance is scheduled which will disrupt production. Unavoidable, emergency maintenance, such as a catastrophic failure, is not considered in this simulation. Normal maintenance can be scheduled from 2am until 6am and from 8pm until 10pm, respectively. It is assumed that the maintenance technicians work in shifts around the production. Any maintenance scheduled between 10pm and 2am of the following day is considered to be overtime.



Day / *Time slot*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
Monday	🟨	🟨	🟩	🟩	🟩	🟩	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	🟩	🟩	🟨	🟨
Tuesday	🟨	🟨	🟩	🟩	🟩	🟩	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	🟩	🟩	🟨	🟨
Wednesday	🟨	🟨	🟩	🟩	🟩	🟩	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	🟩	🟩	🟨	🟨
Thursday	🟨	🟨	🟩	🟩	🟩	🟩	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	🟩	🟩	🟨	🟨
Friday	🟨	🟨	🟩	🟩	🟩	🟩	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	🟩	🟩	🟨	🟨

Legend

🟩	Maintenance
🟨	Overtime
▨	Production

Figure 6-7: Schedule time slot assignment

6.4 Roles of Agents in the Simulation

Chapter 5 examined the design of the IMMS agents. The purpose these actors would fulfil if this simulation were to be implemented will now be discussed briefly. Note that the *IMMS* actor is not listed in this instance here due to the fact that it was reduced to a design artefact in architectural design and the simulation (and eventual implementation) is aimed at the actors at this particular phase of the design. Figure 6-8 presents some flowcharts of the possible actions the actors would perform in an implementation.

The *device health assessor* would be responsible for gathering, processing and formatting (where applicable) relevant data from the hardware devices. This data relates specifically to the maintenance prognosis of the devices. This actor is also responsible for storing the collected and processed information in the database, so that other actors in the system can access it.

The *scheduler* actor retrieves the device prognostic information gathered and stored by the device health assessor. The maintenance schedule requirements for schedule generation, i.e. allowed days, time slots within days and any other relevant information, are stored in the database. The device status information is then used to generate a maintenance schedule based on the maintenance schedule template. This process in the simulation is physically performed by *Drools-Planner*. The schedule generation process runs for a specified period of time or until the finalisation criteria are met. The schedule generation is also affected by the constraint-rules governing the process. The end result is the best (not perfect) the *scheduler* (*Drools-Planner*) could achieve given the allocated time period and/or other completion criteria.

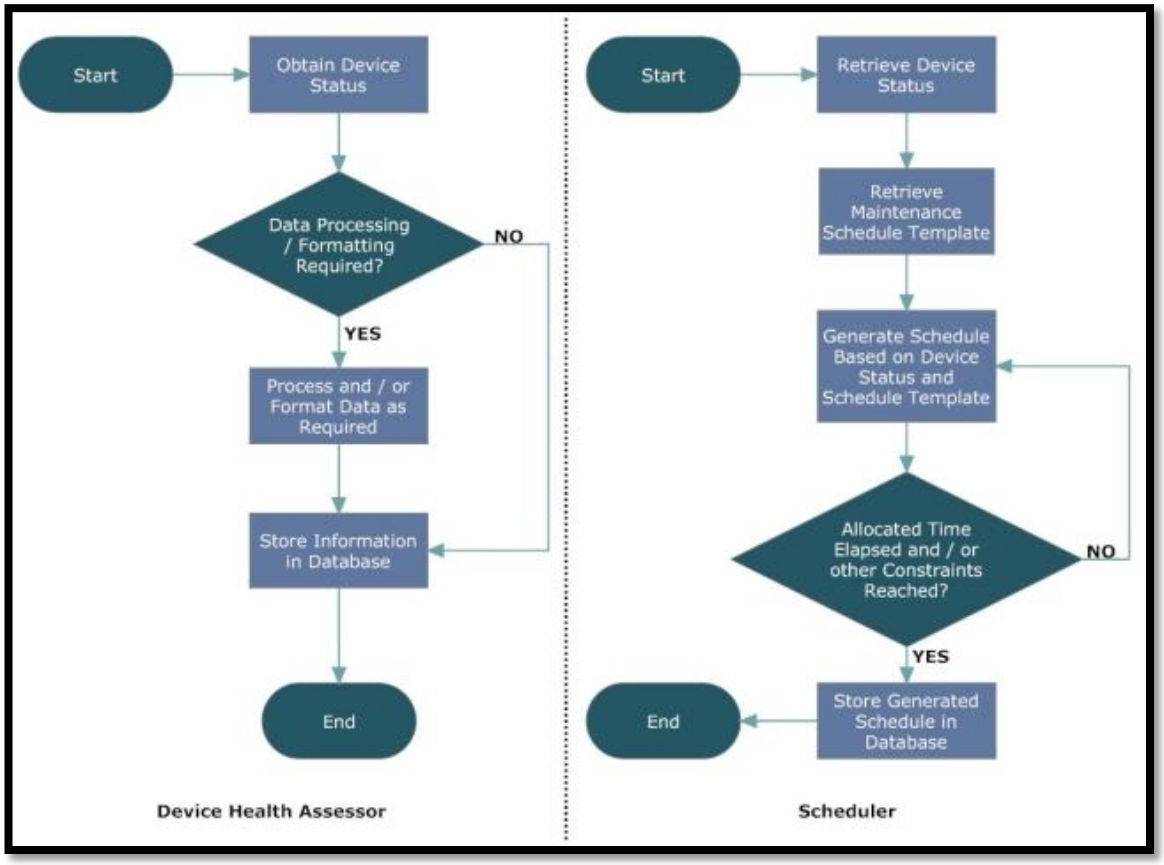

Start
Obtain Device Status
Data Processing / Formatting Required?
NO
YES
Process and / or Format Data as Required
Store Information in Database
End
Device Health Assessor
Start
Retrieve Device Status
Retrieve Maintenance Schedule Template
Generate Schedule Based on Device Status and Schedule Template
Allocated Time Elapsed and / or other Constraints Reached?
NO
YES
Store Generated Schedule in Database
End
Scheduler

Figure 6-8: Flowcharts of simulated agent behaviour

As stated in section 5.4.3, the *solution provider* is merely an interface for maintenance staff to manage and organise solutions to previously resolved maintenance conditions. This, when implemented, will be a non-critical extra to the system that adds no real value to the scenario the simulation aims to achieve. The *solution provider* is thus omitted from the simulation.

6.5 Simulation Database

This section examines the database used in the simulation to store everything required by the maintenance scheduling process. Note that in the *Drools* examples the data used for the solver is stored in XML files. For this simulation it was decided to make use of a database instead, due to the availability of a dedicated database server. It was trivial to substitute the data source from XML to a

MySQL database by simply creating custom implementations by implementing a set of interfaces and abstract classes provided by *Drools*. An entity relationship diagram (ERD) depicting the schema of the simulation database was constructed using a visual editor (*MySQL* Workbench [143]). The ERD is shown in Figure 6-9. Once the tables and relationships were defined in the editor, a connection was made to the database server and the scheme was implemented.

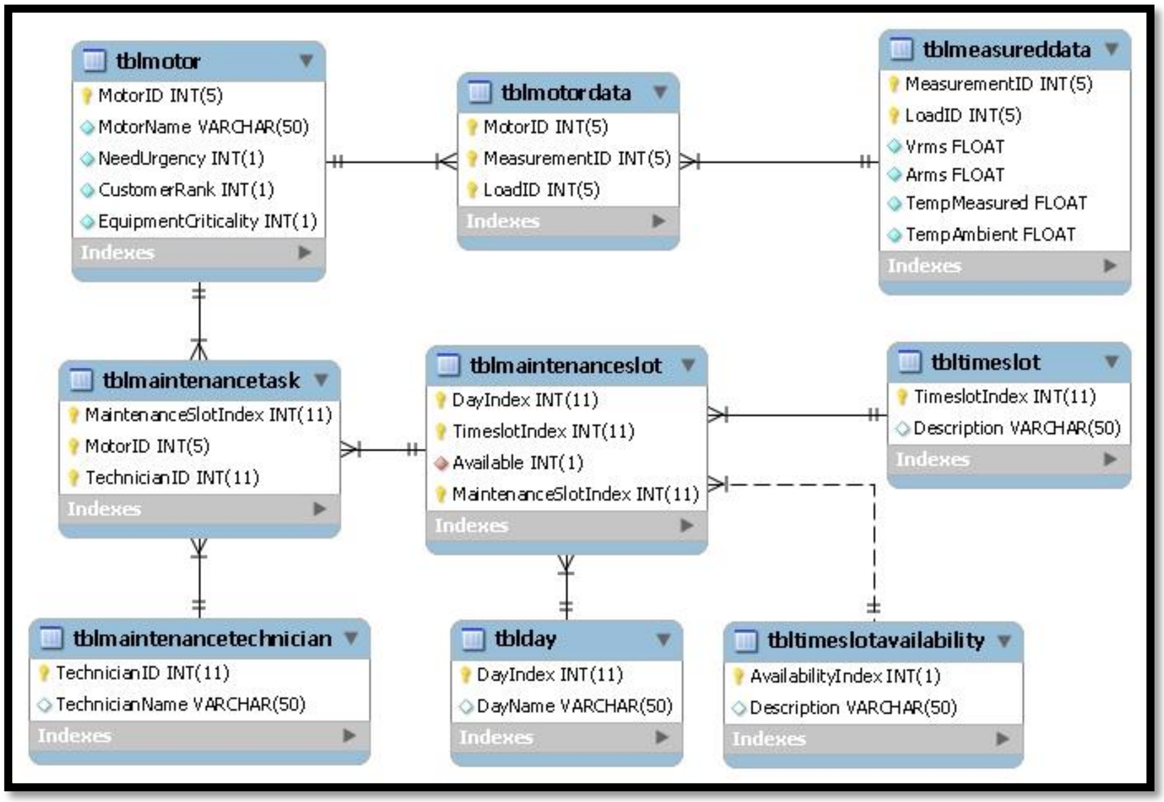


Figure 6-9: Simulation database entity relationship diagram (ERD)

The storage engine for all the tables of the simulation database are *InnoDB*, which is optimised for transactions. Here is a list of the database tables in alphabetical order:

- *tblDay* – List the days for which the schedule should be generated. Uniquely identified by an index.
- *tblMaintenanceSlot* – A list of maintenance slots uniquely identifiable by the day- and time slot indexes. A maintenance task is assigned to this combination key of day and

time slot and not to the day and time slot entities directly. This entity also determines when production is scheduled, maintenance may occur and which hours qualify as overtime by its relationship to the *tblTimeslotAvailability* table.

- *tblMaintenanceTask* – A list of maintenance tasks uniquely identified by the combined indices of the Motor, Technician and *MaintenanceSlot*. This entity defines the *Who*, *What* and *When* of the schedule.
- *tblMaintenanceTechnician* – Maintenance technicians have a unique *TechnicianID*, which is an integer primary key.
- *tblMeasureData* – This table contains all the physical data that was collected (see 6.3.1).
- *tblMotor* – Defines a list of motors on which maintenance should be performed.
- *tblMotorData* – The bridging entity in which the motors are assigned a random subset of data from the total set of measured data.
- *tblTimeslot* – Defines the hours of the day.
- *tblTimeslotAvailability* – All possible hour categories that exist in the system, i.e. normal maintenance, production or overtime.

6.6 Drools Planner Configuration

6.6.1 The Curriculum Course Scheduling Problem

The Curriculum-based timetabling problem consists in the weekly scheduling of the lectures for several university courses within a given number of rooms and time periods, where conflicts between courses are set according to the curricula published by the University and not based on enrolment data. This formulation applies to the University of Udine (Italy) and several other Italian universities, although it was slightly simplified with respect to the real problem to maintain a certain level of generality.

The problem consists of the following entities:

- ***Days***, ***Time Slots***, and ***Periods*** – A number of teaching days in the week are given. Each day is split in a fixed number of time slots, which is equal for all days. A period is a pair composed by a day and a time slot. The total number of scheduling periods is the product of the days times the day time slots.

- ***Courses*** and ***Teachers*** – Each course consists of a fixed number of lectures to be scheduled in distinct periods; it is attended by a given number of students, and is taught by a teacher. For each course there is a minimum number of days in which the lectures of the course should be spread. There are some periods in which the course cannot be scheduled.
- ***Rooms*** – Each room has a capacity, expressed in terms of number of available seats. All rooms are equally suitable for all courses (if large enough).
- ***Curricula*** – A curriculum is a group of courses such that any pair of courses in the group have students in common. Based on curricula, there are conflicts between courses and other soft constraints.

Hard constraints

- ***Lectures*** – All lectures of a course must be scheduled, and they must be assigned to distinct periods.
- ***RoomOccupancy*** – Two lectures cannot take place in the same room in the same period.
- ***Conflicts*** – Lectures for courses in the same curriculum or taught by the same teacher must be scheduled in different periods.
- ***Availabilities*** – If the teacher of the course is not available to teach that course at a given period, then no lectures of the course can be scheduled in that period.

Soft constraints

- ***RoomCapacity*** – For each lecture, the number of students that attend the course must be less or equal than the number of seats of all the rooms that host its lectures.
- ***MinimumWorkingDays*** – The lectures of each course must be spread over a minimum number of days.
- ***CurriculumCompactness*** – Lectures belonging to a curriculum should be adjacent to each other (i.e., in consecutive periods). For a given curriculum we account for a violation every time one lecture is not adjacent to any other lecture within the same day.
- ***RoomStability*** – All lectures of a course should be given in the same room.

The solution to the above problem is the assignment of a period (day and time slot) and a room to all lectures of each course.

6.6.2 A Solution Modelled on the Curriculum Course Scheduling Example

It was decided to model the simulation closely on the curriculum course example as the concepts could easily be translated from the latter to the former. This example was also sufficiently complex to be analysed in order to understand the inner workings of Drools Planner. Table 6-1 shows some of the concepts of the curriculum course example and how these were translated in order to create the simulation. The concepts used in the IMMS simulation are:

- ***Day*** – A day in which maintenance can be scheduled.
- *Maintenance Slot* – Defines a period when maintenance can be scheduled (combination of day and time slot in that day).
- ***Maintenance Task*** – Combination of other concepts that specify when maintenance is scheduled, who will perform the maintenance and on which device.
- ***Maintenance Technician*** – A technician that can perform maintenance-related duties.
- ***Motor*** – Device on which maintenance is to be performed.
- ***Time Slot*** – Period in which maintenance is scheduled (in our case 60 minutes/1 hour).

Table 6-1: Concepts translated from the curriculum course example

Curriculum Course Scheduling Example	IMMS Simulation
Course	Maintenance Task
Day	Day
Lecture	Motor
Period	Maintenance Slot
Teacher	Maintenance Technician
Time slot	Time slot

6.6.3 Solver Configuration

This section discusses the solver configuration used for the simulation. The solver XML configuration file is given in Appendix B .

6.6.3.1 scoreDefinition

When setting the *ScoreDefinition* in the xml configuration file one has a choice between two built-in implementations of the *ScoreDefinition* interface. The first (SIMPLE) defines the *Score* as a *SimpleScore*, which has a single integer value indicating the Score, such as -123. The second (HARD_AND_SOFT) defines the *Score* and a *HardAndSoftScore* and is characterised by separate hard- and soft integer values, such as -123hard/-456soft. It is possible to implement one's own version of the *ScoreDefinition*, but the built-in score definitions should suffice for most needs.

6.6.3.2 selector

In the current version of Drools Planner (5.0) the *selector* generates a list of moves by making use of *MoveFactories*. It was decided to use *relativeSelection* with a taboo search *accepter* (section 6.6.3.3). The *relativeSelection* takes a subset of all the possible moves which in our case was 2% (0.02). Different *relativeSelection* values were experimented with such that the number of moves that were returned allowed for two moves to be performed per second. The number of possible moves determines the duration it takes the planner to select a next move. There exists a trade-off relationship between the two values and some time should be taken to reach an acceptable balance.

6.6.3.3 accepter

An *accepter* filters out unacceptable moves. It can also weigh a move it accepts. An accepter is used, in conjunction with a forager, in order to facilitate active taboo search, simulated annealing, great deluge, etc. For each move it generates an accept chance. A move can be rejected based on a score that determines acceptability. One can implement one's own accepter, although the built-in accepters should suffice for most needs. One can also combine multiple accepters. The current version of Drools Planner provides two built-in accepter types [84]:

- **Taboo search accepter** - When the taboo acceptor takes a step, this step is declared to be taboo. By declaring recently visited steps to be taboo (not allowed), the solver can avoid getting stuck in local-optima. Drools-solver implements various taboo types. Solution taboo makes recently visited solutions taboo. It does not accept a move that leads to one of those solutions. If one can spare the memory, do not be cheap on the taboo size. This type of taboo is recommended because it tends to give the best results and requires little or no tweaking. Move taboo makes recent steps taboo. It does not accept a move equal to one of those steps. Undo move taboo makes the undo move of recent steps taboo. Property taboo makes a property of recent steps taboo. To use property taboo, one's moves must implement the *TabuPropertyEnabled* interface. One can even combine taboo types. If the taboo size is too small, the solver can still get stuck in a local optimum. On the other hand, with the exception of solution taboo, if one picks too large a taboo size, your solver can get stuck by bouncing off the walls. Use the benchmarker to fine-tweak your configuration. A taboo search accepter should be combined with a maximum score of all or first best score improving forager.

- **Simulated annealing accepter** - Initially, simulated annealing (unlike taboo search) does not pick the move with the highest score, neither does it evaluate all moves. It gives un-improving moves a chance, depending on the score and temperature. The temperature is relative to how long it has been solving. In the end, it gradually turns into a simple local search, only accepting improving moves. A simulated annealing accepter should be combined with a first randomly accepted forager.

For this simulation a *completePropertyTabuSize* of five (5) and a *completeSolutionTabuSize* of 1500 were selected after trying a few different configurations.

6.6.3.4 *foragerType*

The forager gathers all accepted moves and picks the move which is the next step. A forager can reduce the subset of all selected moves to be evaluated, by quitting early if a suitable move has been accepted. One can implement one's own custom Forager, but for most situations the built-in foragers should suffice. Drools Planner provides several predefined forages [84]:

- **Maximum score of all forager** - Allows all selected moves to be evaluated and picks the accepted move with the highest score. If several accepted moves have the highest score, one is picked randomly, weighted on the *accept chance* metric.

- **First best score improving forager** - Picks the first accepted move that improves the best score. If none improve the best score, it behaves exactly like the maximum score of all forager.
- **First last step score improving forager** - Picks the first accepted move that improves the last step score. If none improve the last step score, it behaves exactly like the maximum score of all forager.
- **First randomly accepted forager** - Generates a random number for each accepted move and if the accept-chance of a specific move is higher, that move is selected as the next move.

A maximum score of all (MAX_SCORE_OF_ALL) forager was selected in conjunction with taboo accepter as suggested by the documentation [84].

6.6.4 Rules and Score Calculation

This section examines the rules that were applied in this simulation and how the score was calculated when some of the rules matched the solution. The rules discussed in this section are given in Appendix A .

6.6.4.1 Hard Constraints

Constraints in this section were not allowed to be broken and any solution that contains broken hard constraints is not considered a valid solution. The hard constraints of the simulation are as follows:

- **maintenanceTasksInSameTimeslot** – This rule ensured that no maintenance task is scheduled in the same maintenance slot (day and time slot). For each maintenance task scheduled on the same slot as another, a *hard score* reduction of -1 is added.
- **scheduleDuringProduction** – This rule was entrusted to ensure that no maintenance is scheduled during periods of production. For each maintenance task scheduled in a slot marked as production a *hard score* reduction of -1 is added.

- **overtimePerDay** – No employee may be scheduled for more than three (3) hours of overtime per day (24 hour period), according to the South African labour guidelines for general employment [142]. The number subtracted from the total *hard score* is equal to the number of overtime tasks exceeding the 3/day limit.
- **overtimePerWeek** – No employee may be scheduled for more than 12 hours of overtime per week (7-day period) according to the South African labour guidelines for general employment [142]. The number subtracted from the total *hard score* is equal to the number of overtime tasks exceeding the 12/week limit.

6.6.4.2 Soft Constraints

The following constraints do not invalidate a solution. They aid in ranking feasible solutions in an effort to find the 'best' possible solution. This is important since finding the perfect solution is in all probability not possible. These are the soft constraints used in the simulation:

- **motorPriority** – Each motor has a priority that is calculated based on three values: *NeedUrgency*, *CustomerRank* and *EquipmentCriticality* (as discussed in 6.3.2.1). Values closer to one (1) signify highest motor priority. Motors with high priorities should be scheduled first. The value subtracted from the *soft score* is determined by the difference between the priority of the current motor and the motor with a higher priority, scheduled after the former.
- **maintenanceThresholdConstraint** – As discussed in 6.3.2.2, each motor has a maintenance threshold that depends on the load level at which the motor was operated. Motors that are closer to qualifying for maintenance should be scheduled before motors that are not. The value of the *soft score* is determined by the difference between the threshold of the current motor and the motor with a more advanced threshold, scheduled after the former.

It is important to note that these two constraints are not necessarily aligned and often contradict one another. This is the reason why this solution will never have a 0 *soft score*, which would indicate a perfect solution. It is rather a question of finding the best possible solution for the supplied parameters.

6.6.4.3 Accumulate Constraints

Two accumulate rules exist that have the sole responsibility of calculating the sum of the other constraints. The *hardConstraintsBroken* rule ensures that the *hard score* is calculated correctly and reflects the true score value after each iteration of the rule engine. The *softConstraintsBroken* fulfils a similar function for the calculation of the *soft score*. These rules are matched (executed) after all the other rules have been evaluated. This is achieved by setting the salience to -1 (less than the value for other rules, which is 0) [84]. The salience determines the rule evaluation priority. If no salience value is specified, the default value of zero (0) is supplied. It is important that the accumulate rules execute after all the other rules have had the opportunity to alter the soft- and *hard score*. It is also important to note that an accumulate rule does not alter the score; it merely reflects the total of the corresponding soft- or *hard score*.

6.6.5 SolutionInitialiser

As stated in 3.4.2.5, it is the solution initialiser's responsibility to create starting solutions from which the solver can commence solving. Without an acceptable starting solution the solver would waste considerable amounts of time just to arrive at the said starting solution. In this simulation two starting solution initialisers were evaluated:

- **The default Initialiser** – Simulates a poor starting solution by generating a starting solution that does not take into account any of the rules. The motors are assigned to the maintenance slots in the order in which they are retrieved from the database.
- **Structured Initialiser** – The list of motors retrieved from the database is arranged according to their individual priority. If the priorities of two (or more) motors are the same, the maintenance threshold becomes the deciding factor in the scheduling order.

For this simulation only the motor scheduling differs between the two initialiser implementations. Once the motor list has been retrieved (either sorted or unsorted), the motors are assigned to the maintenance slots from the highest priority to the lowest. The rest of the parameters are populated in the same manner. The four maintenance technicians are assigned to maintenance slots in a round robin fashion until all maintenance slots have a technician assigned to them.

6.6.6 Moves

This section examines the possible moves the solver can take in the pursuit of a better solution. Each of these moves has its own *MoveFactory* associated with it which fulfils the function of generating all the possible Moves from the current solution. These moves are then evaluated and weighted and a "winning move" is then selected out of the subset determined by relative selection (as discussed in 6.6.3.2). The following moves are applicable to the simulation scenario:

- **Maintenance Slot Change** – This move swaps out the maintenance slot (time slot and day) of the maintenance task. This is the equivalent of swapping one task with another.
- **Maintenance Technician Change** – Swaps the maintenance technician associated with maintenance task with the technician of another task.
- **Motor Change** – Swaps the motor assigned to one task with the motor assigned to another task.

It is important to note that each move only really alters one parameter. The effect that the move in question has on the score of the solution determines whether the move is included in the set of acceptable moves. Which move the solver eventually selects is determined by several factors including the weight of the move compared to others.

6.6.7 Benchmark Settings

In order to illustrate the benefits of creating a starting solution, as opposed to some arbitrary starting solution, the solver for this simulation is executed with the two separate starting solution initialisers introduced in section 6.6.5. It is important to note that in depth benchmarking was not conducted as part of the simulation and only a different initialiser was used. For both scenarios the solver configuration was kept the same, with the *maximumStepCount* parameter that determines the duration of the solver execution being the only exception.

The *maximumStepCount*- was used in favour of the *maximumSecondsSpend* parameter to determine the duration of the test run. This was done for consistency between execution runs and different configurations, as the processer of the virtual machine used in the development process might be busy with other tasks. This could in turn affect the amount of actual processing performed for a set amount of time and invalidate any tests. To determine the steps-per-second ratio, it was decided to execute the solver for 600 seconds (10 minutes) and record the step-count after each run. This was performed while all other applications and tasks were suspended on the server running the virtual machine, which allowed the server to dedicate all processing resources to the benchmark task. This represents an ideal case and is achievable during a 10 minute execution cycle but increasingly difficult for longer durations of solving. This process was repeated 12 times and the highest and lowest values were eliminated. The remaining ten values were used in order to calculate the average step/second ratio. The average step count over the 10 minute period was 2427. This gave step-per-second value of 4.045. This value was then used to calculate the numbers of steps required to give each scenario the same amount of "time". Table 6-2 Illustrates how the number of steps was calculated using the time in seconds (and minutes). Because the *maximumStepCount* is an integer, the value used in the actual benchmarking was rounded up.

Table 6-2: Run time versus step count

Run #	Seconds	Minutes	# of Steps	# of Steps (Rounded Up)
1	30	0.5	121.35	122
2	60	1	242.7	243
3	120	2	485.4	486
4	300	5	1213.5	1214
5	900	15	3640.5	3641
6	1800	30	7281	7281
7	3600	60	14562	14562
8	7200	120	29124	29124
9	14400	240	58248	58248
10	28800	480	116496	116496

6.7 Conclusion

This chapter examined the IMMS simulation scenario and what it aims to achieve. The physical set-up for the data acquisition was discussed as well as all the components used. The simulation concepts and how they were incorporated in the simulation database were considered. The configuration settings used by the Drools Planner implementation were discussed. The curriculum course example and how the concepts were translated for use in our simulation were also considered. The settings for the schedule generation benchmarks were listed. The next chapter examines the generated schedules and the results of the benchmarks discussed.

Chapter 7 Results and Discussion

7.1 Overview

This chapter presents the results of the simulation scenario discussed in Chapter 5. The motors that are scheduled for maintenance and the criteria determining the process are presented in section 7.2. The unsolved schedules are presented for reference purposes in section 7.3. This chapter examines how the schedule is generated and how the solution is affected by the use of different solution initialisers. The generated schedules using both default and structured solution initialisers are then examined.

7.2 Motor Parameters

This section considers the motors that are to be scheduled for maintenance. Table 7-1 lists the criterion that determines in which order the motors are scheduled. As discussed in section 6.3.2.1, the *Priority* is calculated by the product of the *Need Urgency*, *Customer Rank* and *Equipment Criticality* parameters. Values for the *NUCREC* parameters were generated (randomly generated for the simulation). These values will typically be decided by the maintenance manager in a plant or factory. The values will reflect which devices and systems are more important than others. The highest possible priority is a *Priority* rating of 1 (1 x 1 x 1), and the lowest equates to a *Priority* rating of 64 (4 x 4 x 4). *Maintenance Threshold* (Threshold column in Table 7-1) is an indication, in time units, of the period before a motor is due for maintenance. The lower the *Threshold*, the nearer the device is to requiring maintenance. The *Priority* and *Threshold* determine how the motors are scheduled for maintenance.

Table 7-1: Motor priority and maintenance threshold

ID	Name	Need Urgency	Customer Rank	Equipment Criticality	Priority	Threshold
0	M1	3	4	3	36	22.95

ID	Name	Need Urgency	Customer Rank	Equipment Criticality	Priority	Threshold
1	M2	4	3	3	36	19.05
2	M3	3	4	4	48	19.52
3	M4	2	3	2	12	25.00
4	M5	1	1	3	3	20.97
5	M6	1	1	1	1	20.97
6	M7	2	3	1	6	20.48
7	M8	2	1	1	2	19.52
8	M9	3	4	3	36	24.48
9	M10	1	2	1	2	21.95
10	M11	1	4	2	8	18.58
11	M12	2	2	2	8	25.52
12	M13	1	2	3	6	20.97
13	M14	3	4	4	48	14.50
14	M15	4	3	1	12	29.31
15	M16	4	4	3	48	23.46
16	M17	2	4	2	16	19.05
17	M18	4	1	4	16	16.73
18	M19	1	3	3	9	25.52
19	M20	1	2	3	6	20.00
20	M21	3	4	3	36	14.50
21	M22	1	1	3	3	22.95
22	M23	2	1	3	6	23.46
23	M24	1	4	3	12	20.48
24	M25	4	3	4	48	16.73
25	M26	2	2	2	8	20.00
26	M27	2	3	2	12	19.52
27	M28	4	1	1	4	25.52

ID	Name	Need Urgency	Customer Rank	Equipment Criticality	Priority	Threshold
28	M29	1	2	2	4	14.94
29	M30	2	3	4	24	22.95
30	M31	2	4	1	8	16.73
31	M32	1	2	4	8	14.94
32	M33	3	1	2	6	18.11
33	M34	3	2	3	18	14.94
34	M35	2	3	2	12	17.65
35	M36	1	3	1	3	16.28
36	M37	3	3	1	9	17.65
37	M38	3	3	2	18	24.48
38	M39	2	3	2	12	16.73
39	M40	2	1	2	4	22.45
40	M41	4	2	3	24	17.19
41	M42	1	4	3	12	28.76
42	M43	4	3	2	24	23.46
43	M44	2	3	1	6	32.16
44	M45	1	3	2	6	13.21
45	M46	1	2	4	8	21.95
46	M47	1	2	4	8	17.19
47	M48	1	2	1	2	18.58
48	M49	3	2	3	18	12.36
49	M50	4	2	4	32	20.48

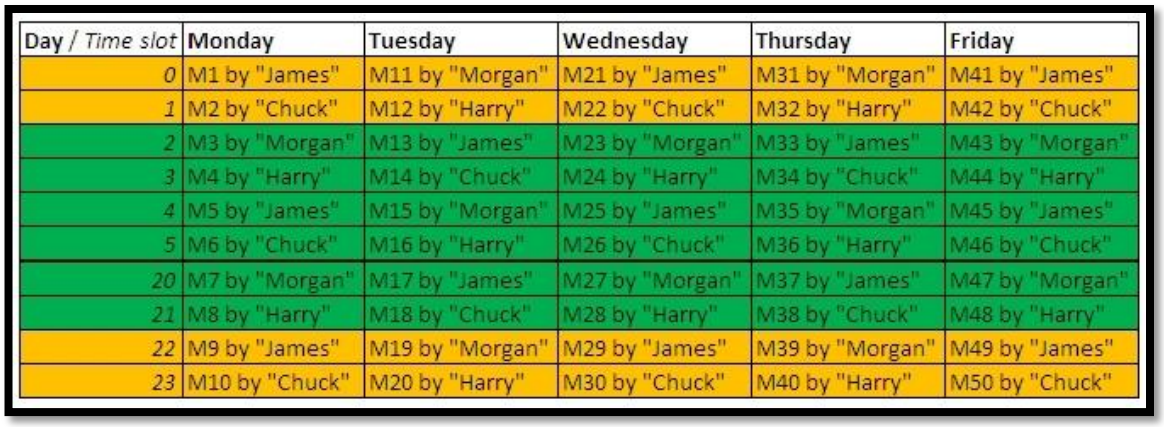
7.3 Initial Solutions

This section examines some initial solutions before the solver is invoked. This allows for the establishment of a baseline to compare the solved schedules once they are introduced later. The time slots indicating production are the same for each of the schedules and have thus been omitted. The grid line between time slot 6 (zero-based index) and 20 is more prominent than the rest. Including these time slots would be to no avail because neither the solution initialisers nor the solver, make any alterations to them. The remainder of the section examines the solutions created by the default and structured initialisers.

The function of the starting solution initialiser is to create a “better”, feasible solution. It forms the starting-point for the solver to commence its processing. If one specified that the solver has a set period to process the solution, the time taken by the solution initialiser also counts towards the total period. In other words, the longer the initialiser takes, the less time will be spent by the solver, whose sole function is to produce a better solution. Creating an acceptable starting solution is a balance between the time spent and the quality of a solution.

7.3.1 Unsolved Schedule Using a Default Initialiser

This initial solution is generated by the default initialiser. This initialiser makes no effort to generate a solution that takes into account the criteria that distinguishes a ‘good’ from a ‘bad’ solution. The motors are retrieved from the database and assigned in the order they are stored to the first available time slot. A similar process is used for the maintenance staff. A round-robin assignment method is used to ensure that the maintenance technicians are varied. There is no adherence to the rules that govern how maintenance technicians are assigned to slots. Figure 7-1 presents the solution created by this initialiser.

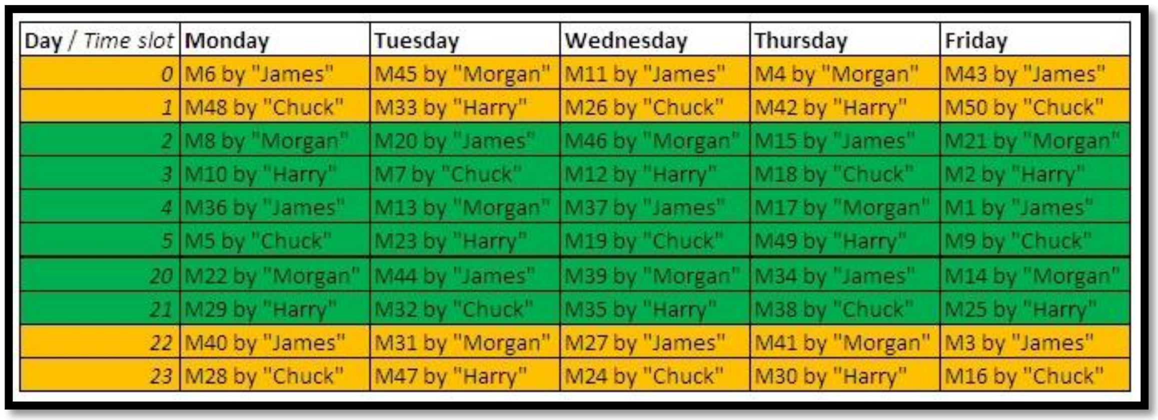


Day / *Time slot*	Monday	Tuesday	Wednesday	Thursday	Friday
0	M1 by "James"	M11 by "Morgan"	M21 by "James"	M31 by "Morgan"	M41 by "James"
1	M2 by "Chuck"	M12 by "Harry"	M22 by "Chuck"	M32 by "Harry"	M42 by "Chuck"
2	M3 by "Morgan"	M13 by "James"	M23 by "Morgan"	M33 by "James"	M43 by "Morgan"
3	M4 by "Harry"	M14 by "Chuck"	M24 by "Harry"	M34 by "Chuck"	M44 by "Harry"
4	M5 by "James"	M15 by "Morgan"	M25 by "James"	M35 by "Morgan"	M45 by "James"
5	M6 by "Chuck"	M16 by "Harry"	M26 by "Chuck"	M36 by "Harry"	M46 by "Chuck"
20	M7 by "Morgan"	M17 by "James"	M27 by "Morgan"	M37 by "James"	M47 by "Morgan"
21	M8 by "Harry"	M18 by "Chuck"	M28 by "Harry"	M38 by "Chuck"	M48 by "Harry"
22	M9 by "James"	M19 by "Morgan"	M29 by "James"	M39 by "Morgan"	M49 by "James"
23	M10 by "Chuck"	M20 by "Harry"	M30 by "Chuck"	M40 by "Harry"	M50 by "Chuck"

Figure 7-1: Unsolved schedule with default solution initialiser

7.3.2 Unsolved Schedule Using a Structured Initialiser

The process in this initialiser involves a partial attempt to adhere to the rules governing the schedule generation process, as discussed in section 6.6.4. It is partial because only the *priority* of the motor is considered. The *maintenance threshold* is ignored as this rule competes with the rule taking into account the *priority*. The rules affecting how the maintenance technicians are assigned are also ignored. How exhaustive this process is performed is up to the designer. Figure 7-2 shows the solution produced by this solution initialiser. The *priority* is used to arrange the motors in ascending order. If the calculated *priority* is found to be identical for two (or more) motors, the *maintenance threshold* is used to decide the order. The motors are then assigned to the first available maintenance slot, in the same way as in the previous section.



Day / *Time slot*	Monday	Tuesday	Wednesday	Thursday	Friday
0	M6 by "James"	M45 by "Morgan"	M11 by "James"	M4 by "Morgan"	M43 by "James"
1	M48 by "Chuck"	M33 by "Harry"	M26 by "Chuck"	M42 by "Harry"	M50 by "Chuck"
2	M8 by "Morgan"	M20 by "James"	M46 by "Morgan"	M15 by "James"	M21 by "Morgan"
3	M10 by "Harry"	M7 by "Chuck"	M12 by "Harry"	M18 by "Chuck"	M2 by "Harry"
4	M36 by "James"	M13 by "Morgan"	M37 by "James"	M17 by "Morgan"	M1 by "James"
5	M5 by "Chuck"	M23 by "Harry"	M19 by "Chuck"	M49 by "Harry"	M9 by "Chuck"
20	M22 by "Morgan"	M44 by "James"	M39 by "Morgan"	M34 by "James"	M14 by "Morgan"
21	M29 by "Harry"	M32 by "Chuck"	M35 by "Harry"	M38 by "Chuck"	M25 by "Harry"
22	M40 by "James"	M31 by "Morgan"	M27 by "James"	M41 by "Morgan"	M3 by "James"
23	M28 by "Chuck"	M47 by "Harry"	M24 by "Chuck"	M30 by "Harry"	M16 by "Chuck"

Figure 7-2: Unsolved schedule with structured solution initialiser

7.4 Solution Initialiser Comparison

This section examines how the *score* of the solution and solver performance is affected by using a different solution initialiser, as discussed in section 6.6.5. The criteria for the benchmarks used were mentioned in section 6.6.7; the results are presented in Table 7-2, and Figure 7-3 shows the accompanying graph. Note that the x axis of the graph represents the time in minutes and the y axis represents the *soft score*. The starting scores were *-0hard/-1280soft* and *-0hard/-921soft* for the default- and structured initialisers, respectively.

Table 7-2: Benchmark scores of a default versus a structured initialiser

Run#	Seconds	Minutes	# of Steps	Default Init.	Struct. Init.
1	30	0.5	122	-1198soft	-914soft
2	60	1	243	-1169soft	-914soft
3	120	2	486	-1165soft	-882soft
4	300	5	1214	-1043soft	-882soft
5	900	15	3641	-963soft	-826soft
6	1800	30	7281	-916soft	-792soft
7	3600	60	14562	-857soft	-792soft
8	7200	120	29124	-772soft	-757soft

Run#	Seconds	Minutes	# of Steps	Default Init.	Struct. Init.
9	14400	240	58248	-733soft	-727soft
10	28800	480	116496	-733soft	-727soft

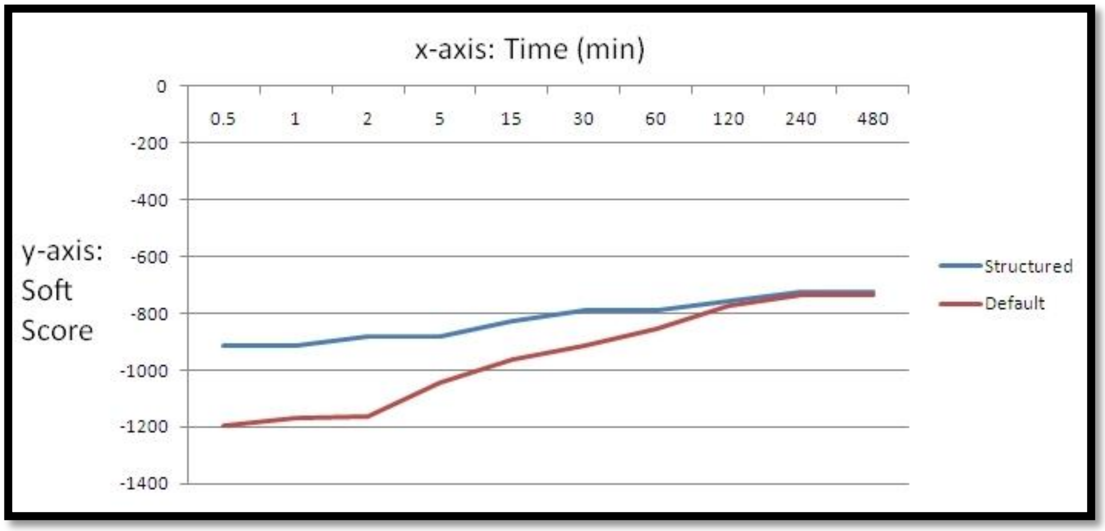


Figure 7-3: Benchmark scores of default versus structured initialiser

Note that the initial difference between the initialisers is quite sizable. This directly translates into less processing required by the solver and highlights the importance of a good, well-balanced starting solution initialiser. As the length of processing increases, the difference between the solution initialisers becomes less prominent. This is the reason why the solution initialiser, although important initially, is not responsible for solving the scheduling problem and thus the actual solution would benefit little from spending a great duration in processing the starting solution initialiser.

The *hard score* of the solutions remains unchanged between iterations of the solver for the entire process. The reason for this is that both solution initialisers create a starting solution which breaks no hard constraints. Even if the initial solution has a *hard score* of 0, the constraints ensure that none of the moves that are accepted diminish the quality of the solution. No matter how great the improvement of the *soft score* is, the move in question will be ignored if the *hard score* is worsened. An example is presented in the excerpt from the *solver* execution log below. Note how the solver ignores moves (line in boldface) that provide a beneficial *soft score* in favour of preserving the *hard score*.

```
INFO: Step index (236), time spend (61234) taking step (4-1: 116-M35 by 2-"Chuck" =>
1-"James") out of 45 accepted moves.
14 Jun 2010 9:49:12 AM BestSolutionRecaller stepTaken
INFO: New score (0hard/-933soft) is not better then last best score (0hard/-914soft).
INFO: Step index (237), time spend (61452) taking step (0-20: 96-M15 by 3-"Morgan" =>
4-6) out of 42 accepted moves.
14 Jun 2010 9:49:12 AM BestSolutionRecaller stepTaken
INFO: New score (-1hard/-878soft) is not better than last best score (0hard/-
914soft).
```

However, if the solver presents a move that improves the *soft score* without worsening the *hard score*, the move is accepted:

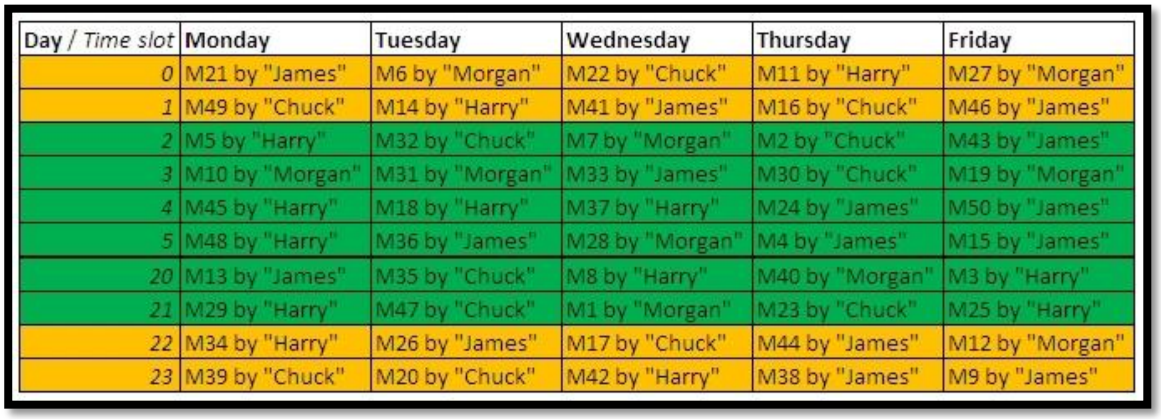
```
INFO: Step index (0), time spend (609) taking step (2-7: 113-M32 by 2-"Chuck" => 1-
"James") out of 53 accepted moves.
14 Jun 2010 9:48:11 AM BestSolutionRecaller stepTaken
INFO: New score (0hard/-922soft) is not better then last best score (0hard/-921soft).
14 Jun 2010 9:48:11 AM DefaultLocalSearchSolver solveImplementation
INFO: Step index (1), time spend (922) taking step (2-2: 101-M20 by 1-"James" => 2-
"Chuck") out of 52 accepted moves.
14 Jun 2010 9:48:11 AM BestSolutionRecaller stepTaken
INFO: New score (0hard/-918soft) is better than last best score (0hard/-921soft).
Updating best solution and best score.
```

The best possible score given in this simulation scenario that the solver could achieve was *-733soft* and *-727soft* for the default- and structured initialisers, respectively. The reason why it was impossible to achieve a *-0hard/-0soft* score is due to two rules specifically that are in competition: the *motorPriority* and *maintenanceThresholdConstraint*. Performing a move to adhere to the one constraint would in most cases break the other. It is possible to weight the scores of the constraint rules so as to favour the one or the other, or to increase/decrease the influence of a specific rule. This is up to the designer, and the problem being solved dictates how the rules should be created and prioritised.

Even if the score is not "perfect", the final solution is still an improvement over that which was initially provided to the solver. In this scenario, the solver would benefit little by increasing the allowed processing duration. This was verified for both initialisers by giving them each 24 hours to process. The resulting score was exactly the same. The criterion that governs the solver is important in order not to waste unnecessary resources, and performance benchmarks give a good indication of what is sufficient.

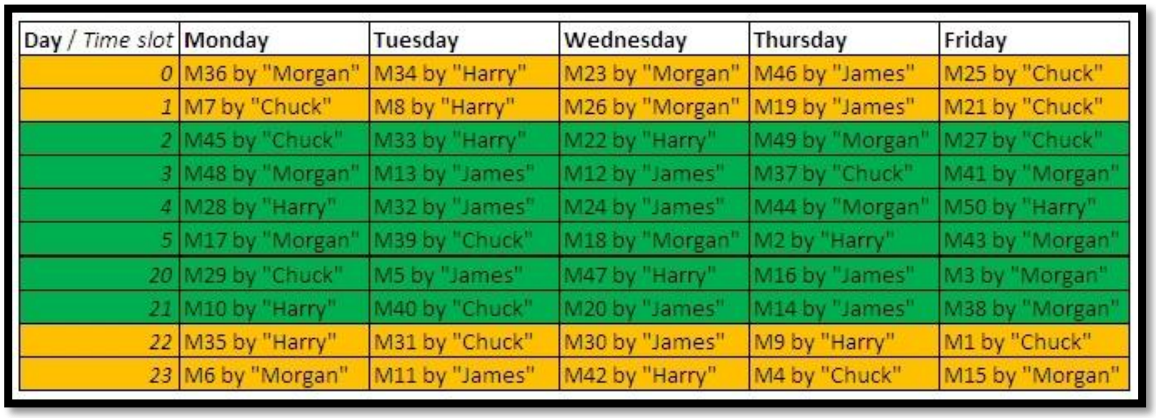
7.5 Solver Solutions

This section considers the solutions once the solver has processed them. The solutions as a result of the default and structured initialiser are presented. It is interesting to note how different these schedules are, even though their final scores only differ by six. It is also interesting to note the differences in the starting and final solution. This only serves to prove that logical solutions are not always the best possible ones. The schedules generated from the initial solutions and initialised by the default and structured initialisers are presented in Figure 7-4 and Figure 7-5, respectively.



Day / *Time slot*	Monday	Tuesday	Wednesday	Thursday	Friday
0	M21 by "James"	M6 by "Morgan"	M22 by "Chuck"	M11 by "Harry"	M27 by "Morgan"
1	M49 by "Chuck"	M14 by "Harry"	M41 by "James"	M16 by "Chuck"	M46 by "James"
2	M5 by "Harry"	M32 by "Chuck"	M7 by "Morgan"	M2 by "Chuck"	M43 by "James"
3	M10 by "Morgan"	M31 by "Morgan"	M33 by "James"	M30 by "Chuck"	M19 by "Morgan"
4	M45 by "Harry"	M18 by "Harry"	M37 by "Harry"	M24 by "James"	M50 by "James"
5	M48 by "Harry"	M36 by "James"	M28 by "Morgan"	M4 by "James"	M15 by "James"
20	M13 by "James"	M35 by "Chuck"	M8 by "Harry"	M40 by "Morgan"	M3 by "Harry"
21	M29 by "Harry"	M47 by "Chuck"	M1 by "Morgan"	M23 by "Chuck"	M25 by "Harry"
22	M34 by "Harry"	M26 by "James"	M17 by "Chuck"	M44 by "James"	M12 by "Morgan"
23	M39 by "Chuck"	M20 by "Chuck"	M42 by "Harry"	M38 by "James"	M9 by "James"

Figure 7-4: Solved schedule with default solution initialiser



Day / *Time slot*	Monday	Tuesday	Wednesday	Thursday	Friday
0	M36 by "Morgan"	M34 by "Harry"	M23 by "Morgan"	M46 by "James"	M25 by "Chuck"
1	M7 by "Chuck"	M8 by "Harry"	M26 by "Morgan"	M19 by "James"	M21 by "Chuck"
2	M45 by "Chuck"	M33 by "Harry"	M22 by "Harry"	M49 by "Morgan"	M27 by "Chuck"
3	M48 by "Morgan"	M13 by "James"	M12 by "James"	M37 by "Chuck"	M41 by "Morgan"
4	M28 by "Harry"	M32 by "James"	M24 by "James"	M44 by "Morgan"	M50 by "Harry"
5	M17 by "Morgan"	M39 by "Chuck"	M18 by "Morgan"	M2 by "Harry"	M43 by "Morgan"
20	M29 by "Chuck"	M5 by "James"	M47 by "Harry"	M16 by "James"	M3 by "Morgan"
21	M10 by "Harry"	M40 by "Chuck"	M20 by "James"	M14 by "James"	M38 by "Morgan"
22	M35 by "Harry"	M31 by "Chuck"	M30 by "James"	M9 by "Harry"	M1 by "Chuck"
23	M6 by "Morgan"	M11 by "James"	M42 by "Harry"	M4 by "Chuck"	M15 by "Morgan"

Figure 7-5: Solved schedule with structured solution initialiser

Both solutions that were created are feasible in the sense that there was no *hard score* penalty. The resulting schedules are almost of equal quality, as these solutions represent the best possible scores, given the scenario and solver parameters. The default initialiser solution showed a significant improvement of 547 (from *-1280soft* to *-733soft*) or 42.7% and the structured initialiser solution had a final improvement of 194 (from *-921soft* to *-727soft*) or 21.1%. Do not be misled by the seemingly more prominent improvement of the initial solution of the default initialiser, as this was a solution far from ideal compared to the one initialised by the structured initialiser. In other words, there was greater room for improvement between the former and the latter.

It is difficult to compare the solutions directly as they had different initial solutions. Given the settings of the solver it was virtually impossible to reach identical final solutions from the two initial solutions. Tweaking the solver parameters might show an improvement in the final score of the solutions, but it is far more likely that improvements of the rules will make a bigger difference. The ultimate solver is a balance between the rules that govern the score calculation and the parameters that control the inner workings of the solver. That aside, if one plainly ignores other aspects and evaluates the solutions on their final scores, the solution resulting from the structured initialiser is a better solution. However marginal the score difference, a better score is still superior.

As this scenario represents a situation where the maintenance schedule is to be generated, it is likely that the solver would be set to halt execution only if no better solution is found after a specified number of steps and/or period of time. If this is a lengthy process it could be scheduled to run overnight or whenever a new schedule is required.

7.6 Conclusion

This chapter discussed the results of the simulation scenario presented in Chapter 6. First, the starting solutions of the different solution initialisers were presented as a reference. The benchmarks of the two solutions were examined and the benefit of a solution initialiser was discussed. The solver solutions and how the motor data was used to schedule the devices for maintenance based on a set of criteria (business rules) were considered. It was shown that device prognosis, irrespective of where the data originated, can be used as a measure to determine how a device is scheduled for maintenance. When implementing the system as a rule engine, the maintenance scheduling can be structured to adhere to the requirements of the company/factory as long as the constraints are formulated as rules.

It was shown that both the solution resulting from the default and structured initialisers were of similar quality, given the comparable final *soft score*. As the complexity of the problem increases so will that of the solution initialiser. Even a relatively simple solving problem, like the one presented in this simulation scenario still exhibit a noticeable performance advantage of using a well-structured solution initialiser as opposed to not initialising the solution before invoking the solver.

Chapter 8 Conclusion

8.1 Introduction

The inefficiencies associated with traditional corrective and blind predictive maintenance can be reduced, if not eliminated, by taking into consideration the manner in which devices are operated, specifically the levels of load and the duration. These parameters can be used as a measure of device prognosis. This prognosis can then, in turn, be used and combined with other relevant criteria, such as policies that govern the allowed working hours of maintenance technicians for example, to schedule devices for maintenance. The maintenance schedules can be generated whenever the requirement for an updated schedule arises. A qualified maintenance technician should first verify schedules before they are used.

This chapter summarises the e-maintenance strategy. It also reviews the achievements of this research work. A reasonable critique, the limitations and some suggestions regarding the work and how it can be extended in future are included.

8.2 Summary

South African assembly systems are exposed to far less volume compared to wealthier, industrialised countries. This is the reason reconfigurable assembly systems are an attractive solution. One product and all its (similar) varieties of it can be produced in the same factory using the same equipment, with limited reconfigurability. This is a more cost-effective solution as it is expensive to reconfigure or rebuild static assembly systems, preconfigured for a specific product, to assemble even a product with slight differences.

Traditional corrective (fail-and-fix) and blind preventive maintenance (follow a schedule and ignore device condition) methods are both inefficient and costly. This inefficiency becomes even more prominent when the target system is a reconfigurable assembly system. This work attempts to address this inefficiency by using the device prognosis as a measure to generate a maintenance schedule. Several artificial intelligence techniques were evaluated to form the basis of the intended system. Multi-agent systems posed several advantages that were the deciding factor. Within the field of agent oriented software engineering, several methodologies were evaluated and Tropos was selected, given the maturity of the methodology and the accompanying design tools.

Several techniques and algorithms used in maintenance management were evaluated and found promising in determining the device prognosis. Although none of these algorithms were physically implemented in the system, the literature provided several examples of implementations and scenarios that were very promising. These algorithms can be used individually or in combination to calculate the prognosis of a device. Certain algorithms are possibly better equipped to determine the prognosis of certain devices or a combination of algorithms can be used to filter, pre- or post-process the information in order to more clearly reflect the desired data.

A simulation scenario was used to test the feasibility of using a measure of prognosis (maintenance threshold) of a device for maintenance schedule generation. In the simulation dc induction motors operating at different loads in order to get different maintenance thresholds were simulated. These thresholds, along with a calculated priority parameter, were used by Drools Planner to generate a maintenance schedule. Maintenance technicians were assigned to perform maintenance on a motor during a certain time slot. The allowed overtime per day and per week were enforced by the inclusion of specific rules that adhere to the South African labour laws. Two solution initialisers were used to create starting solutions for the solver. The benchmark highlighted the performance benefit of using a well-structured solution initialiser. The schedules that resulted from Drools Planner were feasible in the sense that they did not violate the *hard score* of the solver.

It is regrettable that real-world data from a running production or assembly system could not be obtained for this research, despite several attempts to contact individuals in industry or the academia. When the RGEMS assembly system becomes fully operational or an implementation in industry is done, valid data will be available for research purposes. After the failed attempts to gather data from industrial implementations, the decision to conduct a simulation was taken. A simulation unfortunately is just that, a simulation and it is virtually impossible to incorporate every factor that has an effect on a system in the simulation. This affects the accuracy of results and predictions. Simulations do, however, allow one to accurately and reliably recreate situations, which is frequently a requirement when conducting research.

Another limitation of only conducting a simulation is the intrinsic difficulty of predicting the success and efficiency of the proposed system over time. The system (or parts thereof) is only tested in certain controlled situations for short periods of time, compared to systems that would operate in industry. The unique nature of production and assembly systems and their particular needs will most certainly have adverse effects on the operation and performance of a system. This is why it is superior to conduct tests on real-world, operational systems compared to simply performing a simulation.

8.3 Future Work

This research has a number of limitations. An ontology, in the context of science and information sciences, is a formal representation of knowledge by a set of concepts within a specific domain and the relationships between those concepts. The main purpose of an ontology is to reason about domain properties and to describe the domain itself. In this research an ontology was not incorporated or developed. It will be beneficial to consider integrating an existing ontology or developing a custom ontology to allow for formal reasoning and knowledge representation in the system. This would enable external applications or modules to communicate and exchange information with the system in a uniform manner. This is especially true if a standardised ontology language is used, such as the Ontology Web Language (OWL) [144].

The implementation of the multi-agent system was only illustrated in the simulation and an actual system implementation was not done. The implementation of the system and its agents will allow easy integration and communication with other agent-based systems. There also exist several agent implementations that are integrated with an ontology, which will allow the actors to reason about the domain for which they are developed and in which they are placed. TAOM4e is also being redeveloped to address some of the limitations of previous versions. It would be beneficial to use a more refined version of the design tool for any implementation, including the agent system presented in this research.

Since reconfigurable assembly systems (RAS) is an emerging concept, it will be some time before the differences between maintaining RAS and normal assembly systems is established. Once this is achieved researchers can explore the requirements of maintenance scheduling for RAS. Such a scheduling system can be developed using MAS, but it will be interesting to note if there are distinct advantages to using MAS in favour of other approaches to maintenance scheduling.

The data used in this work was not collected via OPC, but when this system is to be implemented it would be advantageous to standardise the communication of information. The RGEMS assembly system presented in Chapter 2 employs OPC as a standard communication protocol. OPC is a viable solution for this as it is currently enjoying much attention in various fields of research and continued development. OPC foundation [145] strives to support the newest trends of communication technologies and standards as they are enhanced and further developed. With the advances of the protocol, backwards compatibility of previous versions and compatibility with other popular industrial standards ensure that legacy systems are also easily supported.

The results obtained from the simulation showed the suitability of the proposed e-maintenance strategy for industrial maintenance applications. Future work should address some of the limitations given above. A future MAS implementation based on the Tropos design system should be carried out and it would be beneficial to use standardised methodologies and protocols in order to ensure maximum compliance with other systems. Although adhering to industrial standards as far as possible can complicate the implementation of the system, integration and compliance with other systems would be simpler. The data used in such an implementation should be collected via OPC.

Given the valuable experiences gained in trying to obtain usable data from individuals in industry and research groups alike, it would definitely be advantageous to be able to obtain measurement data from a local production/assembly system that is easily accessible. This system should also be operated in realistic conditions so that devices follow a natural trend of degradation and subsequent maintenance cycles. Once the assembly system of the RGEMS lab is completed and operational it will provide a system for realistic measurements. These measurements can then be used for research projects similar to the one presented in this dissertation.

Appendices

Appendix A Drools-Planner Rules DRL

```
package za.co.rgems.projects.imms.drl;
    dialect "java"

#List of import classes

import gnu.lists.Convert;

import java.util.ArrayList;

import org.drools.solver.core.score.calculator.HardAndSoftConstraintScoreCalculator;
import org.drools.solver.core.score.constraint.IntConstraintOccurrence;
import org.drools.solver.core.score.constraint.ConstraintType;

import za.co.rgems.projects.imms.domain.Day;
import za.co.rgems.projects.imms.domain.Timeslot;
import za.co.rgems.projects.imms.domain.MaintenanceSlot;
import za.co.rgems.projects.imms.domain.MaintenanceTask;
import za.co.rgems.projects.imms.domain.MaintenanceTechnician;
import za.co.rgems.projects.imms.domain.Motor;
import za.co.rgems.projects.imms.domain.MotorStatus;
import za.co.rgems.projects.imms.domain.MaintenanceSlotHelper;

#Global variable declarations

global HardAndSoftConstraintScoreCalculator scoreCalculator;

// ####################
// # HARD CONSTRAINTS #
// ####################

// No maintenance tasks should be scheduled in the same
//  timeslot (maintenance slot)
rule "maintenanceTasksInSameTimeslot"

  when
        $leftMaintenanceTask :
               MaintenanceTask(
                      $leftMaintenanceTaskId : id,
                      $leftMaintenanceSlot : maintenanceSlot
               );

        $rightMaintenanceTask :
               MaintenanceTask(
                      id > $leftMaintenanceTaskId,
                      maintenanceSlot == $leftMaintenanceSlot
               );
  then
        insertLogical(new IntConstraintOccurrence("maintenanceTasksInSameTimeslot",
               ConstraintType.NEGATIVE_HARD, 1, $leftMaintenanceTask,
               $rightMaintenanceTask));
end
```

```
// No technician should work more than 3 hours overtime per day
rule "overtimePerDay"

  when
         MaintenanceTask(
                $dayIndex : maintenanceSlot.day.dayIndex,
                $maintenanceTechnician : maintenanceTechnician
         );

         $overtimeTaskCount : Number( intValue > 3 ) from
                                accumulate(
                                    $task : MaintenanceTask(
                                      eval(maintenanceSlot.getAvailability() == 1),
                                      motor != null,
                                      maintenanceTechnician != null,
                                      maintenanceSlot.day.dayIndex == $dayIndex,
                                      maintenanceTechnician == $maintenanceTechnician
                                    ),
                                    count(
                                        $task
                                    )
                                );

    then
        insertLogical(new IntConstraintOccurrence("overtimePerDay",
            ConstraintType.NEGATIVE_HARD,
            Convert.toInt($overtimeTaskCount), $maintenanceTechnician));
end

// No technician may work more than 12 hours of overtime per wee
rule "overtimePerWeek"

    when
        MaintenanceTask(
                        eval(maintenanceSlot.getAvailability() == 1),
                        $leftMaintenanceTaskId : id,
                        $leftSlotID : maintenanceSlot.id,
                        $dayIndexOne : maintenanceSlot.day.dayIndex,
                        $maintenanceTechnician : maintenanceTechnician
                    );

        $overtimeTaskCount : Number( intValue > 12 ) from
                                accumulate(
                                  $task : MaintenanceTask(
                                    eval(maintenanceSlot.getAvailability() == 1),
                                    id != $leftMaintenanceTaskId,
                                    maintenanceSlot.id > $leftSlotID,
                                    eval(maintenanceSlot.getDay().getDayIndex() -
                                       Convert.toInt($dayIndexOne) <= 7),
                                    maintenanceTechnician == $maintenanceTechnician
                                ),
                                  count(
                                    $task
                                  )
                                );

    then
        insertLogical(new IntConstraintOccurrence("overtimePerWeek",
      ConstraintType.NEGATIVE_HARD, Convert.toInt($overtimeTaskCount),
            $maintenanceTechnician));
end
```

```
// ####################
// # SOFT CONSTRAINTS #
// ####################

// Constraint on maintenance tasks scheduled during timeslots marked as overtime
rule "minimiseOvertime"

    when
        $maintenanceTask :  MaintenanceTask(
                                eval(maintenanceSlot.getAvailability() == 1),
                                motor != null,
                                maintenanceTechnician != null
                            );
    then
        insertLogical(new IntConstraintOccurrence("minimiseOvertime",
            ConstraintType.NEGATIVE_SOFT, 1, $maintenanceTask));
end

// Arrange motors according to their priority
rule "motorPriority"

    when
        $violatingTask :    MaintenanceTask(
                                $leftMaintenanceTaskID : id,
                                $leftMaintenanceSlotID : maintenanceSlot.id,
                                motor != null,
                                $leftMotorPriority : motor.getMotorPriority
                            );

       $numberOfViolations : Number() from
                                accumulate(
                                    $task : MaintenanceTask(
                                      id != $leftMaintenanceTaskID &&
                                        maintenanceSlot.id >
                                         $leftMaintenanceSlotID &&
                                      motor != null,
                                      $rightMotorPriority : motor.getMotorPriority,
                                        eval( Convert.toInt($rightMotorPriority) <
                                                  Convert.toInt($leftMotorPriority) )
                                    ),
                                    count(
                                        $task
                                    )
                                );
    then
        insertLogical(new IntConstraintOccurrence("motorPriority",
            ConstraintType.NEGATIVE_SOFT, Convert.toInt($numberOfViolations),
            $violatingTask));
end
```

```
// Organise tasks according to motors' time-to-maintenance-required
rule "maintenanceThresholdConstraint"

    when
        $violatingTask :    MaintenanceTask(
                                $leftMaintenanceTaskID : id,
                                $leftMaintenanceSlotID : maintenanceSlot.id,
                                motor != null,
                                $leftTimeToMaintenance :

motor.motorStatus.calculateMotorTimeToMaintenanceRequired
                            );

        $numberOfViolations : Number() from
                                accumulate(
                                  $task : MaintenanceTask(
                                    id != $leftMaintenanceTaskID &&
                                    maintenanceSlot.id > $leftMaintenanceSlotID &&
                                    motor != null,
                                    $rightTimeToMaintenance :
motor.motorStatus.calculateMotorTimeToMaintenanceRequired,

eval(Convert.toDouble($rightTimeToMaintenance) <
                                          Convert.toDouble($leftTimeToMaintenance) )
                                    ),
                                    count(
                                        $task
                                    )
                                );
    then
        insertLogical(new IntConstraintOccurrence("maintenanceThresholdConstraint",
            ConstraintType.NEGATIVE_SOFT, Convert.toInt($numberOfViolations),
$violatingTask));
end

// ##########################
// # ACCUMULATE CONSTRAINTS #
// ##########################

// Accumulate hard constraints
rule "hardConstraintsBroken"
        salience -1 // Finish the other rules first (optional, for performance)
    when
        $hardTotal : Number() from accumulate(
            IntConstraintOccurrence(constraintType == ConstraintType.NEGATIVE_HARD,
      $weight : weight), sum($weight)
        );
    then
        scoreCalculator.setHardConstraintsBroken($hardTotal.intValue());
end

// Accumulate soft constraints
rule "softConstraintsBroken"
        salience -1 // Finish the other rules first (optional, for performance)
    when
        $softTotal : Number() from accumulate(
            IntConstraintOccurrence(constraintType == ConstraintType.NEGATIVE_SOFT,
            $weight : weight), sum($weight)
        );
    then
        scoreCalculator.setSoftConstraintsBroken($softTotal.intValue());
end
```

Appendix B Drools-Planner Solver XML Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : immsSolverConfig.xml
    Created on : 12 April 2010, 8:37 AM
    Author     : De Ville Weppenaar
    Description:
        XML configuration file for Drools-Planner.
-->

<localSearchSolver>
    <scoreDrl>/za/co/rgems/projects/imms/drl/immsRules.drl</scoreDrl>
    <scoreDefinition>
        <scoreDefinitionType>HARD_AND_SOFT</scoreDefinitionType>
    </scoreDefinition>
    <startingSolutionInitializerClass>

za.co.rgems.projects.imms.solution.initialiser.ImmsStartingSolutionInitialiser
    </startingSolutionInitializerClass>
    <finish>
        <!--<maximumSecondsSpend>600</maximumSecondsSpend>-->
        <!--<feasableScore>-999999hard/-999999soft</feasableScore>-->
        <!--<feasableScore>-0hard/-1000soft</feasableScore>-->
        <maximumStepCount>250</maximumStepCount>
        <!--<maximumUnimprovedSecondsSpend>15000</maximumUnimprovedSecondsSpend>-->
    </finish>
    <selector>
        <selector>
         <moveFactoryClass>
                za.co.rgems.projects.imms.move.factory.MotorChangeMoveFactory
         </moveFactoryClass>
         <relativeSelection>
                0.02
         </relativeSelection>
        </selector>
        <selector>
         <moveFactoryClass>
                za.co.rgems.projects.imms.move.factory.MaintenanceSlotMoveFactory
         </moveFactoryClass>
         <relativeSelection>
                0.02
         </relativeSelection>
        </selector>
        <selector>
         <moveFactoryClass>
 za.co.rgems.projects.imms.move.factory.MaintenanceTechnicianChangeMoveFactory
         </moveFactoryClass>
         <relativeSelection>
                0.02
         </relativeSelection>
        </selector>
    </selector>
    <accepter>
         <completePropertyTabuSize>
                5
         </completePropertyTabuSize>
         <completeSolutionTabuSize>
                1500
         </completeSolutionTabuSize>
    </accepter>
    <forager>
         <foragerType>
                MAX_SCORE_OF_ALL
         </foragerType>
    </forager>
</localSearchSolver
```

References

[1] Vermaak, H. and Kinyua, J., “Multi-Agent Systems based Intelligent Maintenance Management for a Component-Handling Platform,” in *IEEE International Conference on Automation Science and Engineering*, Scottsdale, Arizona, September 2007, pp. 1057-1062.

[2] Lee, J., Ni, J., Djurdjanovic, D., Qiu, H. and Liao, H., “Intelligent Prognostics Tools and e-Maintenance,” *Computers in Industry*, vol. 57, no. 6, pp. 476-489, August 2006.

[3] Djurdjanovic, D., Yang, J., Qiu, H., Lee, J. and Ni, J., “Web-enabled Remote Spindle Monitoring and Prognostics,” in *Proceedings of the 2nd International CIRP Conference on Reconfigurable Systems*, Ann Harbor, MI, August 2003.

[4] Jennings, N. R. and Wooldridge, M., “Applications of Intelligent Agents,” in *Agent Technology: Foundations, Applications and Markets*, Jennings, N. R. and Wooldridge, M., Ed. New York, United States of America: Springer-Verlag, 1998, ch. 1, pp. 3-28.

[5] Jennings, N. R. and Wooldridge, M., “Agent Theories, Architectures and Languages: A Survey,” in *Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents*, Amsterdam, The Netherlands, 1995, pp. 1-39.

[6] Wooldridge, M., *An Introduction to MultiAgent Systems*, Chichester: John Wiley and Sons Ltd., 2002.

[7] Bernon, C., Cossentino, M. and Pavón, J., “An Overview of Current Trends in European AOSE Research,” *Informatica*, vol. 29, no. 4, pp. 379-390, 2005.

[8] *International Timetabling Competition*, http://www.cs.qub.ac.uk/itc2007/index.htm, last accessed in April 2010.

[9] *Central University of Technology*, http://www.cut.ac.za/, last accessed in May 2010.

[10] *Research Group in Evolvable Manufacturing Systems*, http://www.rgems.co.za/, last accessed in November 2009.

[11] Janse van Rensburg, J. C., Weppenaar, D. and Bothma, B. C. (October 22, 2009). “RGEMS Infrastructure Overview,” Available: http://www.rgems.co.za/News/Documents/RGEMS%20Infrastructure%20Overview/RGEMS%20IT.pdf, last accessed in November 2009.

[12] *IEEE Standard for gigabit Ethernet over copper wiring*, IEEE 802.3ab, June 28, 1999.

[13] *Amendment to the IEEE 802.11-2007 wireless networking standard to improve network throughput over previous standards (802.11b and 802.11g)*, IEEE 802.11n-2009, October 29, 2009.

[14] *MySQL - The world's most popular open source database*, http://www.mysql.com/, last accessed in November 2009.

[15] *Sun Microsystems*, http://www.sun.com/, last accessed in November 2009.

[16] *GNU General Public License*, http://www.gnu.org/licenses/old-licenses/gpl-2.0.html, last accessed in November 2009.

[17] *Multenet - PocketPAD 4 - RS232 Serial Encapsulation*, http://www.multenet.com/products/pocketpad4.html, last accessed in November 2009.

[18] *About OPC - What is OPC?*, http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC, last accessed in November 2009.

[19] *About OPC - What is the OPC Foundation?*, http://www.opcfoundation.org/Default.aspx/01_about/01_history.asp?MID=AboutOPC, last accessed in November 2009.

[20] OPC Task Force. (October 27, 1998). "OPC Overview," Available: http://www.opcfoundation.org/Archive/33b51ec5-80d9-4b29-80fb-0e7683aa3333/General/OPC%20Overview%201.00.pdf, last accessed in December 2009.

[21] *Cognex*, http://www.cognex.com/, last accessed in December 2009.

[22] *Kuka*, http://www.kuka.com/, last accessed in December 2009.

[23] *Kepware*, http://www.kepware.com/, last accessed in December 2009.

[24] *Allen-Bradley*, http://www.ab.com/, last accessed in December 2009.

[25] *Rockwell Automation*, http://www.rockwellautomation.com/index.html, last accessed in December 2009.

[26] *Cyberlogic*, http://cyberlogic.com/, last accessed in December 2009.

[27] *Bosch Rexroth AG*, http://www.boschrexroth.com/, last accessed in December 2009.

[28] *Multenet - Serial to Ethernet Device Servers*, http://www.multenet.com/, last accessed in March 2010.

[29] Moore, W. J. and Starr, A. G., "An intelligent maintenance system for continuous cost-based prioritization of maintenance activities," *Computers in Industry*, vol. 57, no. 6, pp. 595-606, February 2006.

[30] Muller, A., Crespomarquez, A. and Iung, B., "On the concept of e-maintenance: Review and current research," *Reliability Engineering & System Safety*, vol. 93, no. 8, pp. 1165-1187, August 2008.

[31] Koç, M., Ni, J., Lee, J. and Bandyopadhyay, P., "Introduction of e-manufacturing.," in *Proceedings of the 31st North American manufacturing research conference (NAMRC).*, Hamilton, Canada, May 2003.

[32] Han, T. and Yang, B., "Development of an e-maintenance system integrating advanced techniques," *Computers in Industry: Special Issue on E-maintenance*, vol. 57, no. 6, pp. 569-580, August 2006.

[33] Stergio, C. and Siganos, D. "Neural Networks: An Introduction," Available: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html, last accessed in February 2008.

[34] Petriu, E. "Neural Networks: Basics," Available: http://www.site.uottawa.ca/~petriu/NN_basics-tutorial-2004.pdf, last accessed in February 2008.

[35] Manikandan, V. and Devarajan, N., "Mathematical Approach towards Fault Detection and Isolation of Linear Dynamical Systems," *International Journal of Computational and Mathematical Sciences*, vol. 1, no. 1, pp. 82-91, 2007.

[36] Awad, E. M., *Building Expert Systems: Principles, Procedures, and Applications*, St. Paul, Minnesota, United States of America: West Publishing Co., 1996.

[37] Pirttioja, T., Pakonen, A., Seilonen, I., Halme, A. and Koskinen, K., "Multi-agent based information access services for condition monitoring in process automation," in *3rd IEEE International Conference on Industrial Informatics*, Perth, Australia, August 2005, pp. 240-245.

[38] Zadeh, L. A., Fuzzy Logic Systems: Origin, Concepts, And Trends, 2004, Distinguished Lecture at Hong Kong Baptist University.

[39] Mendel, J. M., “Fuzzy Logic Systems for Engineering: A Tutorial,” *Proceedings of the IEEE*, vol. 83, no. 3, pp. 345-377, March 1995.

[40] Siddique, A., Yadava, G. S. and Singh, B., “Applications of Artificial Intelligence Techniques for Induction Machine Stator Fault Diagnostics,” in *4th IEEE International Symposium on Diagnostics for Electric Machines, Power Electronics and Drives*, Atlanta, United States of America, August 2003, pp. 29-34.

[41] Rein, G., Lautenberger, C., Fernandez-Pello, A. C., Torero, J. L. and Urban, D. L., “Application of Genetic Algorithms and Thermogravimetry to Determine the Kinetics of Polyurethane Foam in Smoldering Combustion,” *Combustion and Flame*, vol. 146, no. 1-2, pp. 95-108, 2006.

[42] Magnus, C., “Evolving Electroacoustic Music: The Application of Genetic Algorithms to Time-Domain Waveforms,” in *International Computer Music Conference*, Miami, United States of America, November 2004, pp. 173-176.

[43] Marczyk, A. (April 23, 2004). “Genetic Algorithms and Evolutionary Computation,” Available: http://www.talkorigins.org/faqs/genalg/genalg.html, last accessed in May 2008.

[44] *Foundation for Intelligent Physical Agents (FIPA)*, http://www.fipa.org/, last accessed in June 2008.

[45] Sedlmayer, M., Knublauch, H. and Rose, T., “Towards a Multi-Agent System for Pro-active Information Management in Anesthesia,” in *4th International Conference on Autonomous Agents*, Barcelona, Spain, 2000.

[46] Pirttioja, T., Seilonen, I., Appelqvist, P., Halme, A. and Koskinen, K., “Agent-based architecture for information handling in process automation,” in *Sixth IFIP International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services*, Vienna, Austria, September 2004, pp. 73-80.

[47] Zhang, W., “Knowledge Management for E-Maintenance of Industrial Automation Systems,” in *Contributions to Ubiquitous Computing*, vol. 42 Heidelberg, Germany: Springer Berlin, 2007, pp. 139-155.

[48] Yu, R., Iung, B. and Panetto, H., “A multi-agents based E-maintenance system with case-based reasoning decision support,” *Engineering Applications of Artificial Intelligence*, vol. 16, no. 4, pp. 321-333, June 2003.

[49] Passadore, A. and Pezzuto, G., “A Multi-Agent Platform Supporting Maintenance Companies on the Field,” in *Workshop: From Objects to Agents*, Genova, Italy, September 2007, pp. 87-95.

[50] Naedele, M., Sager, P. and Frei, C., “Using Multi-Agent Systems for Intelligent Plant Maintenance Functionality,” in *5th World Congress on Intelligent Control and Automation (WCICA 2005)*, Hangzhou, China, June 2004.

[51] Sahin, F., Yavuz, M. Ç., Arnavut, Z. and Uluyol, Ö., “Fault diagnosis for airplane engines using Bayesian networks and distributed particle swarm optimization,” *Parallel Computing*, vol. 33, no. 2, pp. 124-143, March 2007.

[52] van Noortwijk, J. M., Cooke, R. M. and Kok, M., “A Bayesian failure model based on isotropic deterioration,” *European Journal of Operational Research*, vol. 82, no. 2, pp. 270-282, April 1995.

[53] Juang, M. and Anderson, G., “A Bayesian method on adaptive preventive maintenance problem,” *European Journal of Operational Research*, vol. 155, no. 2, pp. 455-473, June 2004.

[54] *The Weibull Distribution*, http://www.weibull.com/LifeDataWeb/the_weibull_distribution.htm, last accessed in April 2010.

[55] Morales, J., Castellanos, M. E., Mayoral, A. M., Fried, R. and Armero, C., “Bayesian design in queues: An application to aeronautic maintenance,” *Journal of Statistical Planning and Inference*, vol. 137, no. 10, pp. 3058-3067, October 2007.

[56] Oukhellou, L., Côme, E., Bouillaut, L. and Aknin, P., “Combined use of sensor data and structural knowledge processed by Bayesian network: Application to a railway diagnosis aid scheme,” *Transportation Research Part C: Emerging Technologies*, vol. 16, no. 6, pp. 755-767, December 2008.

[57] Borguet, S. and Léonard, O., “Comparison of adaptive filters for gas turbine performance monitoring,” *Journal of Computational and Applied Mathematics, Article in Press, Corrected Proof*, August 2009.

[58] Karami, F., Poshtan, J. and Poshtan, M., “Detection of broken rotor bars in induction motors using nonlinear Kalman filters,” *ISA Transactions*, vol. 49, no. 2, pp. 189-195, April 2010.

[59] Wu, J., Huang, C. and Huang, R., “An application of a recursive Kalman filtering algorithm in rotating machinery fault diagnosis,” *NDT & E International*, vol. 37, no. 5, pp. 411-419, July 2004.

[60] Yang, S. K., “An experiment of state estimation for predictive maintenance using Kalman filter on a DC motor,” *Reliability Engineering & System Safety*, vol. 75, no. 1, pp. 103-111, January 2002.

[61] Shao, Y. and Mechefske, C. K., “Gearbox vibration monitoring using extended Kalman filters and hypothesis tests,” *Journal of Sound and Vibration*, vol. 325, no. 3, pp. 629-648, August 2009.

[62] Mahadevan, S. and Shah, S. L., “Fault detection and diagnosis in process data using one-class support vector machines,” *Journal of Process Control*, vol. 19, no. 10, pp. 1627-1639, December 2009.

[63] Widodo, A., Yang, B. and Han, T., "Combination of independent component analysis and support vector machines for intelligent faults diagnosis of induction motors," *Expert Systems with Applications*, vol. 32, no. 2, pp. 299-312, February 2007.

[64] Ge, M., Du, R., Zhang, G. and Xu, Y., "Fault diagnosis using support vector machine with an application in sheet metal stamping operations," *Mechanical Systems and Signal Processing*, vol. 18, no. 1, pp. 143-159, January 2004.

[65] Shin, H. J., Eom, D. and Kim, S., "One-class support vector machines: An application in machine fault detection and classification," *Computers & Industrial Engineering*, vol. 48, no. 2, pp. 395-408, March 2005.

[66] Qu, J. and Zuo, M. J., "Support vector machine based data processing algorithm for wear degree classification of slurry pump systems," *Measurement*, vol. 43, no. 6, pp. 781-791, March 2010.

[67] Widodo, A. and Yang, B., "Support vector machine in machine condition monitoring and fault diagnosis," *Mechanical Systems and Signal Processing*, vol. 21, no. 6, pp. 2560-2574, August 2007.

[68] *An Introduction to Wavelets*, http://www.amara.com/IEEEwave/IEEEwavelet.html, last accessed in April 2010.

[69] Miao, Q. and Makis, V., "Condition monitoring and classification of rotating machinery using wavelets and hidden Markov models," *Mechanical Systems and Signal Processing*, vol. 21, no. 2, pp. 840-855, February 2007.

[70] Reddy, M. J. and Mohanta, D. K., "A wavelet-fuzzy combined approach for classification and location of transmission line faults," *International Journal of Electrical Power & Energy Systems*, vol. 29, no. 9, pp. 669-678, November 2007.

[71] Cusidó, J., Romeral, L., Ortega, J. A., Garcia, A. and Riba, J. R., “Wavelet and PDD as fault detection techniques,” *Electric Power Systems Research*, vol. 80, no. 8, pp. 915-924, December 2009.

[72] Prabhakar, S., Mohanty, A. R. and Sekhar, A. S, “Application of discrete wavelet transform for detection of ball bearing race faults,” *Tribology International*, vol. 35, no. 12, pp. 793-800, December 2002.

[73] Eddy, S. R., “Hidden Markov models,” *Current Opinion in Structural Biology*, vol. 6, no. 3, pp. 361-365, June 1996.

[74] Caesarendra, W., Niu, G. and Yang, B., “Machine condition prognosis based on sequential Monte Carlo method,” *Expert Systems with Applications*, vol. 37, no. 3, pp. 2412-2420, March 2010.

[75] Gregorčič, G. and Lightbody, G., “Gaussian process approach for modelling of nonlinear systems,” *Engineering Applications of Artificial Intelligence*, vol. 22, no. 4-5, pp. 522-533, June 2009.

[76] *DLPlan User's Guide*, Group for Artificial Intelligence Applications - University of Madrid, Madrid, Spain, 2008, pp. 5-6.

[77] McGuinness, D. L. and van Harmelen, F. (January 2, 2002). “Web Ontology Language (OWL Lite, OWL DL, and OWL Full) Feature Synopsis,” Available: http://www.ksl.stanford.edu/people/dlm/webont/OWLFeatureSynopsisJan22003.htm, last accessed in 2010 April.

[78] *Drools Introduction and General User Guide*, Red Hat Documentation Group, Raleigh, NC, USA, 2008.

[79] *JPlan: Java GraphPlan Implementation*, http://jplan.sourceforge.net/, last accessed in Arpil 2010.

[80] Blum, A. and Furst, M., “Fast planning through planning graph analysis,” *Artificial Intelligence*, vol. 90, no. 1-2, pp. 281-300, February 1997.

[81] *International Timetabling Competition - The Finalists*, http://www.cs.qub.ac.uk/itc2007/winner/finalorder.htm, last accessed in April 2010.

[82] *JBoss Rules Reference Manual*, Red Hat Documentation Group, Raleigh, NC, United States of America, 2008.

[83] de Smet, G. (May 19, 2009). “Drools Solver - Community Documentation,” Available: http://downloads.jboss.com/drools/docs/5.0.1.26597.FINAL/drools-solver/html_single/index.html, last accessed in 2010 April.

[84] *Drools Solver (Planner) - Community Documentation*, Red Hat Documentation Group, Raleigh, North Carolina, United States of America, 2009.

[85] Monostori, L., Váncza, J. and Kumara, S. R. T., “Agent-Based Systems for Manufacturing,” *CIRP Annals - Manufacturing Technology*, vol. 55, no. 2, pp. 697-720, January 2007.

[86] Pirttjioja, T., “Agent-Augmented Process Automation System,” Master's Thesis, Automation Technology Laboratory of the Department of Electrical and Communications Engineering, Helsinki University of Technology, Otaniemi, Espoo, Finland, 2002.

[87] Jussila, J., “Agent-Based Approach to Supervisory Information Services in Process Automation,” Masters Thesis, Automation Technology Laboratory of the Department of Automation and Systems Technology, Helsinki University of Technology, Otaniemi, Espoo, Finland, 2006.

[88] Jennings, N. R., “On Agent-Based Software Engineering,” *Artificial Intelligence*, vol. 117, no. 2, pp. 277-296, September 1999.

[89] Wooldridge, M., “Agent-Based Software Engineering,” *IEEE Proceedings - Software*, vol. 144, no. 1, pp. 26-37, February 1997.

[90] Chakraborty, S., "Agent Based Approach to Fault Recover in a Process Automation System," Masters Thesis, Automation Technology Laboratory of the Department of Automation and Systems Technology, Helsinki University of Technology, Otaniemi, Espoo, Finland, 2003.

[91] Fajt, M., "Information Aagents in Process Automation," Master's Thesis, Helsinki University of Technology, Automation Technology Laboratory of the Department of Automation and Systems Technology, Otaniemi, Espoo, Finland, 2003.

[92] Leitão, P., Restivo, F. and Putnik, G., "A Multi-Agent Based Cell Controller," in *Proceedings of 8th IEEE International Conference on Emerging Technologies and Factory Automation*, Antibes, France, October 2001, pp. 463-470.

[93] Leitão, P. and Restivo, F., "Holonic Adaptive Production Control Systems," in *Proceedings of the 28th Annual Conference of the IEEE Industrial Electronics Society*, Sevilla, Spain, November 2002, pp. 2968-2973.

[94] Roy, D., Anciaux, D. and Vernadat, F., "SYROCO: A Novel Multi-Agent Shop-Floor Control System," *Journal of Intelligent Manufacturing*, vol. 12, no. 3, pp. 295-307, June 2001.

[95] Cowling, P. I., Quelhadj, D. and Petrovic, S., "A Multi-Agent Architecture for Dynamic Scheduling of Steel Hot Rolling," *Journal of Intelligent Manufacturing*, vol. 14, no. 5, pp. 456-470, October 2003.

[96] Glover, F. G., "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Operations Research*, vol. 13, no. 5, pp. 533-549, 1986.

[97] Rendón-Sallard, T., Sànchez-Marrè, M., Devesa, F. and Poch, M., "Simulating scenarios for decision-making in river basin systems through a Multi-Agent system," in *Proceedings of the International Environmental Modelling and Software Society's Third Biennial Meeting: Summit on Evironmental Modelling and Software*, Burlington, United States of America, July 2006, pp. 133-139.

[98] Patricio, M. A., Carbó, J., Pérez, O., García, J. and Molina, J. M., “Multi-Agent Framework in Visual Sensor Networks,” *EURASIP Journal on Advances in Signal Processing: Special Issue on Visual Sensor Networks*, vol. 2007, no. 1, pp. 226-247, January 2007.

[99] Mathieson, I., Padgham, L. and Vo, B. Q., “Agent Based Travel and Tourism Planning,” in *International Conference on Autonomous Agents: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, The Netherlands, July 2005, pp. 129 - 130.

[100] Willmott, S., “Deploying Intelligent Systems on a Global Scale,” *Intelligent Systems*, vol. 19, no. 5, pp. 71-73, September 2004.

[101] *Australian Tourism Data Warehouse (ATDW)*, http://www.atdw.com.au/, last accessed in January 2010.

[102] Shang, Y. and Shi, H., “A Web-based Multi-Agent System for Interpreting Medical Images,” *World Wide Web*, vol. 2, no. 4, pp. 209-218, April 1999.

[103] Vermeulen, I. B., Bohte, S., Somefun, D. J. A. and La Poutré, J. A., “Multi-agent Pareto Appointment Exchanging in Hospital Patient Scheduling,” *Service Oriented Computing and Applications*, vol. 1, no. 3, pp. 185-196, November 2007.

[104] Friedman, D. D. (May 27, 1987). “Does Altruism Produce Efficient Outcomes? Marshall vs. Kaldor,” Available: http://www.daviddfriedman.com/Academic/Marshal_Pareto/Marshal_Pareto.html, last accessed in January 2010.

[105] Conceição, P. and Ferreira, P. (February 29, 2000). “The Young Person's Guide to the Theil Index: Suggesting Intuitive Interpretations and Exploring Analytical Applications,” Available: http://utip.gov.utexas.edu/papers/utip_14.pdf, last accessed in January 2010.

[106] Silaghi, G. C., “Software Engineering Approaches for Design of Multi-agent Systems,” *Economy Informatics*, vol. 6, 2006.

[107] Zambonelli, F., Jennings, N. R. and Wooldridge, M., “Developing multiagent systems: The Gaia methodology,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 12, no. 3, pp. 317-370, July 2003.

[108] Wood, M. F. and DeLoach, S. A., “An Overview of the Multiagent Systems Engineering Methodology,” in *First International Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, 2001, pp. 207-221.

[109] Chella, A., Cossentino, M. and Sabatucci, L., “Tools and patterns in designing multi-agent systems with PASSI,” *WSEAS Transactions on Communications*, vol. 3, no. 1, pp. 352-358, 2004.

[110] Padgham, L. and Winikoff, M., “The Prometheus Methodology,” in *Methodologies and Software Engineering for Agent Systems*, vol. 11 Boston, United States of America: Kluwer Academic Publishers, 2004, ch. 11, pp. 217-234.

[111] Juan, T., Pearce, A. and Sterling, L., “ROADMAP: Extending the gaia methodology for complex open systems,” in *First international joint conference on Autonomous agents and multiagent systems*, Bologna, Italy, 2002, pp. 3-10.

[112] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. and Mylopoulos, J., “Tropos: An Agent-Oriented Software Development Methodology,” *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203-236, May 2004.

[113] Woolridge, M., Jennings, N. R. and Kinny, D., “The Gaia Methodology for Agent-Oriented Analysis and Design,” *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 3, no. 3, pp. 285-312, 2000.

[114] *Java Agent DEvelopment Framework (JADE)*, http://jade.tilab.com, last accessed in November 2009.

[115] Moriatis, P. and Spanoudakis, N., “Combining Gaia and JADE for Multi-Agent Systems Development,” in *Proceedings of the 17th European Meeting on Cybernetics and Systems Research*, Vienna, Austria, April 2004, pp. 13-16.

[116] Moraitis, P., Petraki, E. and Spanoudakis, N., “Engineering JADE Agents with the Gaia Methodology,” in *International Workshop of 3rd united GI conference: Object-Oriented Programming for the Net World*, Erfurt, Germany, October 2002.

[117] Spanoudakis, N. and Moriatis, P., “The Gaia2JADE process for Multi-Agent Systems Development,” *Applied Artificial Intelligence*, vol. 20, no. 2-4, pp. 251-273, February 2006.

[118] Silaghi, G. C., “Software Engineering Approaches for Design of Multi-agent,” *Economy Informatics*, vol. 5, 2005.

[119] Thangarajah, J., Padgham, L. and Winikoff, M., “Prometheus Design Tool,” in *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, The Netherlands, July 2005, pp. 127-128.

[120] *Prometheus Design Tool (PDT)*, http://www.cs.rmit.edu.au/agents/pdt/, last accessed in November 2008.

[121] *JACK*, http://www.agent-software.com.au/jack.html, last accessed in November 2008.

[122] *REBEL*, http://www.agentlab.unimelb.edu.au/rebel.html, last accessed in November 2008.

[123] Kuan, P. P., Rarunasekera, S. and Sterling, L., “Improving Goal and Role Oriented Analysis for Agent Based Systems,” in *2005 Australian conference on Software Engineering*, Brisbane, Australia, March 2005, pp. 40-47.

[124] *Eclipse*, http://www.eclipse.org/, last accessed in November 2008.

[125] Susi, A., Perini, A., Giorgini, P. and Mylopoulos, J., “The Tropos metamodel and its use,” *Informatica*, vol. 29, no. 4, pp. 401-408, 2005.

[126] Winikoff, M., “Jack Intelligent Agents: An Industrial Strength Platform,” in *Multiagent Systems, Artificial Societies, And Simulated Organizations*, vol. 15, 2005, pp. 175-193.

[127] Dam, K. H. and Winikoff, M., “Comparing Agent-Oriented Methodologies,” in *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems*, Melbourne, Australia, July 2003, pp. 78-93.

[128] *Tool for Agent Oriented Modelling (TAOM)*, http://sra.itc.it/tools/taom4e/, last accessed in November 2009.

[129] *Object Management Group (OMG)*, http://www.omg.org/, last accessed in November 2009.

[130] *OMG Model Driven Architecture*, http://www.omg.org/mda/, last accessed in November 2009.

[131] Morandini, M., “Knowledge Level Engineering of BDI Agents,” Master's Thesis, Department of Engineering and Information Sciences, University of Trento, Trento, Italy, 2006.

[132] Weppenaar, D., Vermaak, H. J. and Kinyua, J., “Towards the Development of a Multi-Agent Systems based Intelligent Maintenance Management System using the Tropos Methodology,” in *3rd Robotics and Mechatronics Symposium*, Pretoria, South Africa, November 2009.

[133] Morandini, M., Nguyen, C. D., Perine, A., Siena, A. and Susi, A., “Tool-supported Development with Tropos: the Conference Management System Case Study,” presented at the *8th International Workshop on Agent Oriented Software Engineering, at the International Conference on Autonomous Agents and Multiagent Systems*, Honolulu, Hawaii, May 2007.

[134] *Eclipse Modelling Framework Project (EMF)*, http://www.eclipse.org/modeling/emf/, last accessed in January 2010.

[135] *Eclipse Graphical Editing Framework*, http://www.eclipse.org/gef, last accessed in January 2010.

[136] *Tefkat: The EMF Transformation Engine*, http://tefkat.sourceforge.net/, last accessed in January 2010.

[137] *eCAT - Continuous Agent Testing on Eclipse*, http://sra.itc.it/people/cunduy/ecat/, last accessed in January 2010.

[138] *Jadex: BDI Agent System*, http://jadex.informatik.uni-hamburg.de/bin/view/About/Overview, last accessed in November 2009.

[139] *t2x tool: from Tropos to Jadex*, http://disi.unitn.it/~morandini/home.html, last accessed in November 2009.

[140] Pokahr, A., Braubach, L. and Lamersdorf, W., “JADEX: A BDI Reasoning Engine,” in *Multi-Agent Programming: Languages, Platforms and Applications*, vol. 15, Bordini, R.H., Dastani, M., Dix, J. and Seghrouchni, A. El Fallah, Ed., 2005, pp. 149-174.

[141] Mobley, R. K., *Maintenance Fundamentals*, 2nd ed., Woburn: Butterworth-Heinemann, 1999.

[142] *South African Department of Labour*, http://www.labour.gov.za/, last accessed in April 2010.

[143] *MySQL Workbench*, http://www.mysql.com/products/workbench/, last accessed in April 2010.

[144] *Web Ontology Language*, http://www.w3.org/2004/OWL/, last accessed in July 2008.

[145] *OPC Foundation*, http://www.opcfoundation.org/, last accessed in November 2008.

Glossary of Terms

Agent Oriented Software Engineering: Principles, techniques and methodologies relating to the design of software based on multi-agent systems.

Agent-Based: The software or systems that are based on the agent paradigm.

Architectural Design: The third step of the Tropos design phases in which the system-to-be actor is decomposed into separate sub-actors, each fulfilling a specific role that was initially assigned to the system-to-be actor.

Artificial Intelligence: The intelligence of machines and the branch of computer science that aims to create it.

Artificial Neural Networks: A mathematical model or computational model that tries to simulate the structure and/or functional aspects of biological neural networks.

Assembly Line: A manufacturing process in which parts (usually interchangeable parts) are added to a product in a sequential manner using optimally planned logistics to create a finished product much faster than with handcrafting-type methods.

Automation: The use of control systems and information technologies reducing the need for human intervention.

Autonomous Guided Vehicle: A mobile robot that follows markers or wires in the floor, or uses vision or lasers. They are most often used in industrial applications to move materials around a manufacturing facility or a warehouse.

Bayesian Network: A probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG).

Business Logic integration Platform: *Drools* 5 introduced the Business Logic integration Platform which provides a unified and integrated platform for Rules, Workflow and Event Processing.

Component Object Model: A binary-interface standard for software componentry introduced by Microsoft in 1993.

Condition-Based Maintenance: Condition-based maintenance was introduced to try to maintain the correct equipment at the right time.

Corrective Maintenance: The maintenance that is required when an item has failed or worn out, to return it to working order.

Customer Rank: A metric used to arrange customers into a specific order.

DC motor: A DC motor is an electric motor that runs on direct current (DC) electricity.

Decision Analysis: Decision Analysis (DA) is the discipline comprising the philosophy, theory, methodology, and professional practice necessary to address important decisions in a formal manner.

Degradation: Used in engineering to describe what occurs to machines which are subject to constant, repetitive stress.

Detailed Design: The fourth stage of the Tropos development lifecycle in which the capabilities of the system-to-be actors/sub-actors and the interdependencies on these capabilities are defined.

Deterministic: A system in which no randomness is involved in the development of future states of the system. Deterministic models thus produce the same output for a given starting condition.

Distributed Component Object Model: A proprietary Microsoft technology for communication among software components distributed across networked computers.

Distributed Control Systems: A control system usually of a manufacturing system, process or any kind of dynamic system, in which the controller elements are not central in location (like the brain) but are distributed throughout the system with each component sub-system controlled by one or more controllers.

Early Requirements: The first stage in the Tropos development lifecycle in which the domain stakeholders are identified and the dependencies on one another are modelled.

Efficiency: The ability to do something well or achieve a desired result without wasted energy or effort.

E-maintenance: Refers to the integration of the information and communication technologies (ICT) within the maintenance strategy and/or plan.

Equipment Criticality: A metric that defines how critical a device is compared to another in order to prioritise it according to importance.

Ethernet Encapsulation: The process of providing access to a device on one protocol (serial, modbus, etc.) through Ethernet by making use of middleware that translates between the appropriate protocols.

Expert System: Software that attempts to provide an answer to a problem, or clarify uncertainties where normally one or more human experts would need to be consulted.

Extensible Markup Language: A set of rules for encoding documents in machine-readable form.

Feasible Solution: A solution presented by the solver that is not necessarily the best possible solution, but a solution that does not break any of the hard constraints set by the rules.

Forager: Gathers all accepted moves and picks the next step from them.

Fuzzy Logic: A form of multi-valued logic derived from fuzzy set theory to deal with reasoning that is approximate rather than precise.

Genetic Algorithm: A search heuristic that mimics the process of natural evolution.

Gigabit Ethernet: A term describing various technologies for transmitting Ethernet frames at a rate of a gigabit per second, as defined by the IEEE 802.3-2008 standard.

Hard Constraint: A constraint that is not allowed to be broken. A broken hard constraint basically indicates that the solution is not feasible.

Hidden Markov Model: A statistical model in which the system being modelled is assumed to be a Markov process with unobserved state. A HMM can be considered as the simplest dynamic Bayesian network.

Implementation: The fifth and final stage of the Tropos methodology in which the completed diagrams and models are translated into working code.

Induction Motor: An induction motor or asynchronous motor is a type of alternating current motor where power is supplied to the rotor by means of electromagnetic induction.

Inference Engine: A computer program that tries to derive answers from a knowledge base.

Infrared Thermography: Examples of infrared imaging science. Thermal imaging cameras detect radiation in the infrared range of the electromagnetic spectrum (roughly 900–14,000 nanometers or 0.9–14 µm) and produce images of that radiation, called thermograms.

Kalman Filter: A mathematical method named after Rudolf E. Kalman. Its purpose is to use measurements that are observed over time that contain noise (random variations) and other inaccuracies, and produce values that tend to be closer to the true values of the measurements and their associated calculated values.

Late Requirements: The second step of the Tropos methodology in which the system-to-be is introduced as a single actor and the dependencies between this new actor and the existing environment actors are modelled.

Maintenance: Fixing any sort of mechanical or electrical device should it become out of order or broken (known as repair, unscheduled or casualty maintenance). It also includes performing routine actions which keep the device in working order (known as scheduled maintenance) or prevents trouble from arising (preventive maintenance).

Maintenance Threshold: A metric that indicates when a device is considered eligible to be maintained.

Means-end Relationship: Shows how the goal can be achieved. For example it can be used to connect task to a goal.

Methodology: Methodology may be a description of process, or may be expanded to include a philosophically coherent collection of theories, concepts or ideas as they relate to a particular discipline or field of inquiry

Middleware: Computer software that connects software components or applications. The software consists of a set of services that allows multiple processes running on one or more machines to interact.

Model-Driven Architecture: A software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models.

Multi-Agent System: A class of computational models for simulating the actions and interactions of autonomous agents (either individual or collective entities such as organisations or groups) with a view to assessing their effects on the system as a whole.

Multiresolution Analysis: The design method of most of the practically relevant discrete wavelet transforms (DWT) and the justification for the algorithm of the fast wavelet transform (FWT).

Need Urgency: A metric specifically referring to how urgent a maintenance task is.

Object Linking and Embedding: A technology developed by Microsoft that allows embedding and linking to documents and other objects.

Object-Oriented Programming: A programming paradigm that uses “objects” – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs.

Oil analysis and Tribology: Oil analysis (OA) is the sampling and laboratory analysis of a lubricant’s properties, suspended contaminants, and wear debris. OA is performed during routine preventive maintenance to provide meaningful and accurate information on lubricant and machine condition. Tribology is the science and engineering of interacting surfaces in relative motion. It includes the study and application of the principles of friction, lubrication and wear.

OLE for Process Control: Stands for Object Linking and Embedding (OLE) for Process Control and is the original name for a standards specification developed in 1996 by an industrial automation industry task force. The standard specifies the communication of real-time plant data between control devices from different manufacturers.

OPC Data Access: A group of standards that provides specifications for communicating real-time data from data acquisition devices such as PLCs to display and interface devices like Human-Machine Interfaces (HMI). The specifications focus on the continuous communication of data.

OPC XML-DA: Builds on the OPC Data Access specifications to communicate data in XML; incorporates SOAP and Web services.

Open-Source: Practices in production and development that promote access to the end product's source materials.

Pareto-Improvement: A performed action that harms no singular entity and helps at least one instance of the predefined entities.

Particle Swarm Optimisation: A method for performing numerical optimisation without explicit knowledge of the gradient of the problem to be optimised.

Pattern Matching: The act of checking for the presence of the constituents of a given pattern.

Perfect Solution: A solution to a planning problem in which none of the hard or soft constraints are considered broken.

Preventative Maintenance: Maintenance, including tests, measurements, adjustments, and parts replacement, performed specifically to prevent faults from occurring.

Proactive: Proactive behaviour (or proactivity) by individuals refers to anticipatory, change-oriented and self-initiated behaviour in the work place. Proactive behaviour involves acting in advance of a future situation, rather than just reacting.

Prognostics: An engineering discipline focused on predicting the future condition or estimating remaining useful life of a component and/or system of components.

Programmable Logic Controller: A digital device that can be programmed for the automation of electromechanical processes, such as control of machinery on factory assembly lines.

Radio Frequency Identification: Use of an object (typically referred to as an RFID tag) applied to or incorporated into a product, animal, or person for the purpose of identification and tracking using radio waves. Some tags can be read from several meters away and beyond the line of sight of the reader.

Reasoner: A piece of software able to infer logical consequences from a set of asserted facts or axioms. The notion of a semantic reasoner generalises that of an inference engine, by providing a richer set of mechanisms to work with.

Reconfigurable Manufacturing System: A reconfigurable manufacturing system (RMS) is one designed at the outset for rapid change in its structure, as well as its hardware and software components, in order to quickly adjust its production capacity and functionality within a part family in response to sudden market changes or intrinsic system change.

Recursion: A method of defining functions in which the function being defined is applied within its own definition; specifically it is defining an infinite statement using finite components.

Rete Algorithm: An efficient pattern matching algorithm for implementing production rule systems.

RS232, RS422 and RS485: Standards that define serial binary single-ended data and control signals connecting between a DTE (Data Terminal Equipment) and a DCE (Data Circuit-terminating Equipment). Serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus.

Rule Engine: A software system that executes one or more business rules in a runtime production environment.

Schedule: An organised list usually set out in tabular form, providing information about a series of arranged events: in particular, the time at which it is planned these events will take place.

Score Calculation: The process of calculating the total score of a solution based on the hard and soft constraints that the solution in question breaks or adheres to.

Simulated Annealing: A generic probabilistic meta-heuristic for the global optimisation problem of applied mathematics, namely locating a good approximation to the global optimum of a given function in a large search space.

Soft Constraint: A constraint that does not render a solution unfeasible and is used to reason over the quality of feasible solutions.

Software Agent: A piece of software that acts for a user or other program in a relationship of agency. The idea is that agents are not strictly invoked for a task, but activate themselves.

Software Engineering: A profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build.

Solution Initialiser: A separate piece of software that prepares an initial, in most cases feasible, solution which the solver can then improve. This is done to give the solver a head start in the solving process.

Solver: The software in *Drools* that makes single changes to the solution and determines whether the new solution is better than the initial.

Support Vector Machines: A set of related supervised learning methods used for classification and regression.

Transparency: Any form of distributed system should hide its distributed nature from its users, appearing and functioning as a normal centralised system.

Truth Maintenance: The ability of a system to restore consistency.

Turing Completeness: A collection of data-manipulation rules (an instruction set, programming language, or cellular automaton) is said to be Turing Complete if and only if such system can simulate a single taped Turing Machine (a theoretical device that manipulates symbols contained on a strip of tape). Classical Turing-complete systems include context-dependent grammars, recursive functions and lambda calculus.

Ultrasonic Sensor: Ultrasonic sensors generate high-frequency sound waves and evaluate the echo that is received back by the sensor.

Vibration Analysis: Analysis of mechanical oscillations about an equilibrium point. The oscillations may be periodic such as the motion of a pendulum or random such as the movement of a tyre on a gravel road.

Visual Inspection: Visual Inspection typically means inspection using raw human senses and/or any non-specialised inspection equipment.

Wavelet: A wave-like oscillation with an amplitude that starts out at zero, increases, and then decreases back to zero. It can typically be visualised as a 'brief oscillation' like one might see recorded by a seismograph or heart monitor. Generally, wavelets are purposefully crafted to have specific properties that make them useful for signal processing.

Web Ontology Language: A family of knowledge representation languages for authoring ontologies endorsed by the World Wide Web Consortium. They are characterised by formal semantics and RDF/XML-based serialisations for the Semantic Web.

Web-based: A web application is an application that is accessed over a network such as the Internet or an intranet. The term may also mean a computer software application that is hosted in a browser-controlled environment (e.g. a Java applet) or coded in a browser-supported language (such as JavaScript, combined with a browser-rendered markup language like HTML) and reliant on a common web browser to render the application executable.

Weibull Distribution: The Weibull distribution is a continuous probability distribution.

Wireless Communication: The transfer of information over a distance without the use of enhanced electrical conductors or “wires”.

Wireless N: IEEE 802.11n-2009 is an amendment to the IEEE 802.11-2007 wireless networking standard to improve network throughput over the two previous standards — 802.11a and 802.11g — with a significant increase in the maximum raw data rate from 54 Mbit/s to 600 Mbit/s with the use of four spatial streams at a channel width of 40 MHz.