Interest in multi-agent systems (MAS) research has grown in recent years. With this research, new implementation possibilities are being sought on a continuous basis. One potential domain in which MAS could be applied is in the development of an intelligent maintenance management system (IMMS). As with any software development, a methodology is required to design and implement the system. In this paper we describe the development of an IMMS based on MAS using the TROPOS methodology. TROPOS is an agent-oriented software engineering (AOSE) methodology for developing multi-agent systems. The development process of a multi-agent system using TROPOS consists of a number of phases, and most of these phases of the development process of our system are described in this paper. The development process is supported by an interactive development environment (IDE) software tool. The tool also facilitates automatic code generation in a later phase of the design process leading to the implementation of the agents of our system.

Keywords: maintenance management, multi-agent systems (MAS), e-Maintenance, maintenance scheduling, Drools-Planner

1. INTRODUCTION

Device degradation, more often than not, happens over a measurable period of time. Humans often not perceive the degradation and failures that are frequently misconceived as instantaneous occurrences. 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, which in turn can be used to determine device prognosis and generate optimised maintenance schedules. The workload of maintenance personnel is reduced, freeing valuable resources that can be deployed elsewhere. The maintenance information can be readily available to other departments and the synchronisation of maintenance tasks with production schedules can be simplified. The purpose of this paper is to present a method for implementing an intelligent maintenance management system (IMMS) using multi-agent systems (MAS) by means of the TROPOS methodology.

The rest of the paper is organised as follows. Section 2 offers a brief overview of related work. The methodology and tools applied in the design are outlined.
in section 3. Section 4 presents the proposed IMMS. The results are presented in section 5 and the concluding remarks, as well as possibilities for future research, are summarised in section 6.

2. RELATED WORK

Traditional maintenance strategies can be considered as either reactive, whereby the equipment is fixed or replaced after it fails, or blindly proactive, where a certain level of performance degradation is assumed, with no input from the machinery itself, and servicing equipment on a routine schedule regardless of status (Vermaak & Kinyua, 2007). Both these scenarios are wasteful and leave room for improvement. Effective maintenance management can improve the profitability of a manufacturing enterprise (Vermaak & Kinyua, 2007). Machines go through a measurable process of degradation, even though failures are often viewed as instantaneous occurrences. By measuring this degradation in some manner, it is possible to predict when and where a failure of system machinery will occur.

e-Maintenance, a new generation of maintenance, can be attributed to globalisation and the fast growth of the communication, computer and information technologies (Vermaak & Kinyua, 2007). Distributed organisations can benefit from e-maintenance where people, expertise or data are physically separate or isolated. e-Maintenance also facilitates the transition from mere 'predictive maintenance' to intelligent 'prognosis'. e-Maintenance is a “maintenance management concept whereby assets are monitored and managed over the Internet” (Vermaak & Kinyua, 2007). e-Maintenance brings benefits to a distributed organisation, which is where plants, people, expertise or data are physically separate or isolated. 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 online sensing techniques to minimise the system downtime by balancing the risk of failure and achievable profits.

Recently, agent-oriented software development has been used as a paradigm for software engineering in a wide variety of applications (Jennings & Wooldridge, 1998). The agent concept constitutes a powerful abstraction tool when dealing with software development and it provides a good development paradigm in domains that are complex, open and distributed; requiring agents to act autonomously and to cooperate, coordinate, communicate, negotiate and even compete (Jennings & Wooldridge, 1995).

In most cases, the solution to complex distributed problems developed using a multi-agent systems (MAS) approach are composed of multiple interacting agents acting as computing elements to achieve the systems goals. MAS have the traditional advantages of distributed and concurrent problem solving,
in addition to sophisticated patterns of interactions such as cooperation, coordination and negotiation. It is the nature of these interactions that distinguishes multi-agent systems from other forms of software and that has allowed agents to tackle the increasingly complexity of the open software systems where integration, transparency and interoperability among heterogeneous components are essential. Development methodologies, tools and platforms that facilitate the implementation of agent systems have also been developed (Bernon, Cossentino & Pavón, 2005). The methodologies currently available include: GAIA (Zambonelli, Jennings & Wooldridge, 2003), MaSE (Wood & DeLoach, 2001), TROPOS (Bresciani, Perini, Giorgini, Giunchiglia & Mylopoulos, 2004), PASSI (Chella, Cossentino & Sabatucci, 2004), Agent-UML (Bauer, Muller & Odell, 2000) and so on.

Traditional maintenance management systems are mainly based on corrective ("fail" and "fix") and preventative (scheduled time-based schemes) maintenance strategies. The traditional maintenance management systems can be converted to intelligent maintenance management systems (IMMS) enabling systems to achieve near-zero breakdown performance, and ultimately transform the traditional maintenance practices from a 'fail and fix' to 'predict and prevent' methodology. Current production machines contain sophisticated sensors and it is now possible to rapidly sense performance indicators, and thus assess and predict system performance (Djurdevanovic, Yang, Qiu, Lee & Ni, 2003).

Solutions based on intelligent software agents or multi-agent systems (MAS) have been used to solve problems in a wide variety of domains such as e-mail filtering, manufacturing, process control, e-commerce, health care and air traffic control (Jennings & Wooldridge, 1998). In section 3, we present the development of our intelligent maintenance management system (IMMS) using multi-agent systems.

3. THE IMMS SYSTEM

The intelligent maintenance management system (IMMS) we are developing aims to monitor health degradation of devices in an industrial automation platform in order to generate optimised maintenance schedules. The system is a MAS, consisting of several intelligent software agents. These agents work collectively to accomplish the overall goals of the system. Object linking and embedding for process control (OPC) is used as communications medium between our software agents and the industrial automation platform (OPC Task Force, 1998). As our system is focussed on providing maintenance services, the agents will only be making use of the monitoring facility that OPC provides.

In order to store and analyse the health status of hardware devices, the information is stored in a database. The information is used to detect trends in
device health degradation and predict when a failure might occur. Generated schedules need to be verified by maintenance personnel before they can be considered as final. It is important to note that the physical maintenance tasks are still performed by maintenance personnel. Our system only aims to provide the maintenance personnel with the information they require to make more informed decisions.

The following subsections below present the different design phases of the IMMS system based on the TROPOS methodology. A number of design diagrams created with the tool for agent-oriented modelling (for Eclipse TAOM4E) are included. Unfortunately due to space constraints all of the diagrams could not be included.

3.1. Early requirements

The domain stakeholders and their dependencies on one another were identified as per specification of the early requirements. Seven stakeholders in total were identified:

- Maintenance personnel - The maintenance team consisting of humans that perform the physical maintenance tasks.
- Domain expert - If a maintenance scenario arises that cannot be solved by the maintenance technicians on site, the technical support department of the device in question is contacted to resolve the issue.
- Factory manager - The manager of the entire factory.
- Automation - platform The industrial platform consisting of all the hardware devices.
- Operator - The operator in charge of monitoring the industrial processes.
- Suppliers - The supply companies for the various maintenance parts.
- Procurement personnel - The department that negotiates with the suppliers to obtain the required maintenance parts.

Figure 1 depicts the early requirements actor diagram in which the relationships and dependencies of these stakeholders is illustrated.

The circles represent the actors, whilst the shapes connecting the actors are the goals. The actors depend on one another for the achievement of the goals that connect them. For example, the maintenance personnel depend on the domain expert for the achievement of the 'supply domain expertise' goal. It is important to note that the system-to-be as an actor is not included in the TROPOS early requirements diagram. The current system, as it is, is modelled in early requirements in order to provide some clarity on current processes and interactions. This in turn helps the designer to determine how the existing actors are to interact with the system-to-be designed and vice versa.
3.2. Late requirements

In late requirements the system-to-be is introduced as a single actor. The dependencies between the existing actors and the newly added system actor are added. The intelligent maintenance management system (IMMS) actor that symbolises the system-to-be is depicted in Figure 2. All the original goals from early requirements are still present. The IMMS actor and the new set of goals represent the system-to-be implemented and its goals.
In addition to the actor diagram, each actor has their own goal diagram depicting their goals, plans, resources, etc. An example of one such goal diagram is illustrated by Figure 3.

Figure 3 also depicts goal decomposition. The decomposition illustrated is known as AND-decomposition. If a goal is AND-decomposed into sub-goals it implies that a goal is deemed successfully achieved if all of the sub-goals connected in the relationship return successful. Another form of goal-subdivision is OR-decomposition, in which any of the sub-goals must return successful in order for the main goal to be declared completed.

**Figure 3:** Example of the supplier actor's internal goal diagram

The same type of decomposition can be used for plans where a plan is decomposed into sub-plans, where at least one or all of the sub-plans need to be performed before the originating plan can be considered completed. The connection between a goal and plan is a means-end relationship. This relationship implies that the plan (means) is the means to achieve the goal (end). If a goal can be achieved through multiple plans, these plans can be seen as in an OR-relationship with one another towards achieving the goal.

### 3.3. Architectural design

This phase of the design consists of three steps of which the first defines the overall architecture. New actors, and in some cases sub-actors, are
introduced. Sub-goals and sub-plans can be delegated to sub-actors. Three sub-actors were identified:

- **Solution provider** The functionality of managing the maintenance solutions has been delegated to this actor. These solutions can then aid maintenance personnel in future with similar queries.
- **Scheduler** It is the responsibility of the scheduler agent to oversee the generation of maintenance schedules. The maintenance personnel then verify these schedules.
- **Device health assessor** This actor performs the prognostic analysis of the hardware devices in the system.

It is important to note that the design is decentralised. The IMMS actor is now reduced to merely a design artefact and the responsibility is delegated to the aforementioned sub-actors. The sub-actors interact directly with the other agents in the environment. No messages pass through, or are controlled by the IMMS agent.

Figure 4 depicts the various sub-actors of the IMMS agent along with their inter-dependencies. If any agent is unable to manage the number of requests it must satisfy, this actor can be cloned and processing of requests can then occur in parallel. Each of the sub-actors has an internal goal diagram with goal decomposition and is illustrated by Figure 5.

![Figure 4: Architectural design actor diagram with sub-actors (simplified)](image_url)
4. IMMS SIMULATIONS

4.1. Simulation scenario

In this section we will briefly discuss the scenario this simulation is aimed at achieving. Usually a device will have a maintenance guide in the form of hours worked or operations performed. This is not accurate in all cases as the same device can be used in different configurations. 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. The devices are prioritised according to certain criteria as well as a maintenance threshold that is calculated according to 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.

Data indicating the prognostic state of a device is collected and stored. The scheduling component then retrieves the device information from the database, along with the maintenance schedule template. This template specifies which timeslots are available for maintenance. The scheduler component then proceeds to generate the schedule using the schedule template and then attempts to improve the generated schedule. This process
continues until some set criterion is reached, which is a measurement indicating that an acceptable schedule has been reached.

The data for this simulation was collected from a DC induction motor in one of the practical labs at the Central University of Technology (CUT). The motor is a 500V-, 10kW-, 20A-rated DC motor with an 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 nought at no-load). The motor drives a generator that powers rows of 100W light bulbs. There are four rows of bulbs and the rows can be switched individually. Each row contains five bulbs and can be seen 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.

Two temperature sensors were used to measure the ambient temperature and the temperature of the motor bearings. The measurements were taken at 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 significant heat over time.

A national instruments (NI) compactRIO (cRIO) 9014 real-time (RT) controller was used for the temperature recording. 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 platinum resistance thermometers (the PT100 we used). For the simulation the sampling mode was set to high resolution, which gave very accurate and stable temperature measurements by filtering out any noise generated by the DC motor.

4.2. Simulation concepts

4.2.1. Motor priority

The data collected from the motor was used to simulate 50 motors. These motors were simulated 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) (Mobley, 1999). A rating system of numbers 1 through 4 is recommended. Since most humans think of
number 1 as the first priority to get done, the NUCREC system does number 1 first. See Mobley (1999) for a detailed discussion of proposed NUCREC ratings.

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. When several work requests have the same priority, labour and materials availability, locations, and the scheduling fit may guide what may be done first. “With these predetermined evaluations, it is easy to establish the priority for a work order either manually or automatically by taking the numbers from the equipment card and the customer list and multiplying them by the urgency” (Mobley, 1999). 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 assigned a higher priority.

4.2.2. Maintenance threshold

In addition to priority, another metric is also used to determine the scheduling order. We refer to this metric 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. The device for which the maintenance threshold is calculated will determine the algorithm or technique that is most appropriate. It is wise to implement the methods in such a way that they are interchangeable for additional flexibility.

For this simulation the maintenance threshold was calculated by taking the time units a motor was operational and multiplying it by the normal operational value, which was in this case load level three. This gives you a base value to be used in a calculation to determine the percentage of time that has elapsed towards the full maintenance term. 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 (10). The sum of the actual load levels is equal to 29 and the maintenance threshold as a percentage is then: 29/30 x 100 ≈ 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 in close proximity 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.
4.2.3. Maintenance personnel

For this simulation the regulations as specified by the South African Department of Labour (DoL) were used as a guideline. The employer may require the employee to work a certain number 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. 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 above-mentioned regulations. Four maintenance technicians are available to be scheduled for maintenance tasks in this simulation.

4.2.4. 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. In this section the schedule for this simulation is introduced. 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 concepts. The schedule is the same for all working days (Monday through Friday) of the week and is illustrated by Figure 6. Production occurs for a period of 14 hours a day, from 6am till 8pm. During this time no maintenance is scheduled that will disrupt production. Unavoidable, emergency maintenance is not considered in this simulation. Normal maintenance can be scheduled from 2am till 6am and 8pm till 10pm respectively. It is assumed that the maintenance technicians work in shifts during production. Any maintenance scheduled between 10pm and 2am of the following day is considered to be overtime.

Figure 6: Scheduled timeslot assignment
4.3. Agent roles

The IMMS actor is not listed here due to the fact that it was reduced to a design artefact in architectural design and the simulation as well. 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 device status information is then used to generate a maintenance schedule based on the maintenance schedule template. This process in the simulation is performed by Drools-Planner. The schedule generation process runs for a specified period of time or until the finalisation criteria is 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. The solution provider is merely an interface for maintenance staff to manage and organise solutions to previously resolved maintenance conditions.

4.4. Drools-Planner

4.4.1. Building a solver

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 timeslot in that day).
- Maintenance task Combination of other concepts that specify when maintenance is scheduled, who will perform said maintenance and on which device.
- Maintenance technician A technician that can perform maintenance-related duties.
- Motor Device on which maintenance is to be performed.
- Timeslot Period in which maintenance is scheduled (in our case 60 minutes/1 hour)

The solver configuration required us to choose options for setting the ScoreDefinition, selector, acceptor and foragerType as explained here. When setting the ScoreDefinition in the .xml configuration file you have 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 your own version of the *ScoreDefinition*, but the built-in score definitions should suffice for most needs. In the current version of Drools-Planner (5.0) the selector generates a list of moves by making use of *MoveFactories*. The number of possible moves determines the duration it takes the planner to select the next move. There exists a trade-off relationship between the two values and some time should be taken to reach an acceptable balance.

An acceptor filters out unacceptable moves. It can also weigh a move it accepts. An acceptor 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 change. A move can be rejected based on a score that determines acceptability. You can build your own and also combine multiple acceptors. The current version of Drools-Planner provides two built-in acceptor types (Drools Solver, 2009): a taboo search acceptor and simulated annealing acceptor.

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. You can also implement your own custom forager. The Drools-Planner provides several pre-defined foragers (Drools Solver, 2009):

- Maximum score of all forager: Allows all selected moves to be evaluated and picks the accepted move with the highest score.
- First best score improving forager: Picks the first accepted move that improves the best score.
- First-last step score improving forager: Picks the first accepted move that improves the last step score.
- 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 acceptor as suggested by the documentation (Drools Solver, 2009).

4.4.2. 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 timeslot) as another task.

- **scheduleDuringProduction** This rule was entrusted to ensure that no maintenance is scheduled during periods of production.
- **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 (SA Department of Labour, 2010).
- **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 (SA Department of Labour, 2010).

4.4.3. Soft constraints

The following constraints do not invalidate a solution, aid in ranking feasible solution 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 section 4.2.1). Values closer to one (1) signify highest motor priority. Motors with high priorities should be scheduled first.

- **maintenanceThresholdConstraint** 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.

It is important to note that these two constraints are not necessarily aligned and often contradict one another. This is why this solution will never have a nought (0) soft score, which would indicate a perfect solution. It is rather a question of finding the best possible solution for the supplied parameters. Two accumulate rules exist that have the sole responsibility of calculating the sum of the other constraints. It is important to note that an accumulate rule does not alter the score, but it reflects the total of the corresponding soft or hard score.

4.4.4. Solution initialiser

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 said starting solution. In this simulation we evaluate two starting solution initialisers:
• 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 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 the simulation only the motor scheduling differs between the two initialiser implementations. Once the motor list has been retrieved, 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.

5. RESULTS

We present the results of the simulation scenario discussed in section 4. As in section 4, the priority is calculated by the product of the Need Urgency, Customer Rank and Equipment Criticality parameters. 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). The maintenance threshold 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.

5.1. Solution initialiser comparison

In this section we look at how the score of the solution and solver performance is affected by using a different solution initialiser, as discussed in section 4.4.4. The results of the benchmarks are presented in Table 1, and the accompanying graph shown in Figure 7. The starting scores were 0hard/1280soft and 0hard/921soft for the default- and structured initialisers, respectively.
Table 1: Benchmark scores of default versus a structured initialiser

<table>
<thead>
<tr>
<th>Run#</th>
<th>Seconds</th>
<th>Minutes</th>
<th># of Steps</th>
<th>Default Init.</th>
<th>Struct. Init.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>0.5</td>
<td>122</td>
<td>-1198soft</td>
<td>-914soft</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>1</td>
<td>243</td>
<td>-1168soft</td>
<td>-914soft</td>
</tr>
<tr>
<td>3</td>
<td>120</td>
<td>2</td>
<td>486</td>
<td>-1165soft</td>
<td>-882soft</td>
</tr>
<tr>
<td>4</td>
<td>300</td>
<td>5</td>
<td>1214</td>
<td>-1043soft</td>
<td>-882soft</td>
</tr>
<tr>
<td>5</td>
<td>900</td>
<td>15</td>
<td>3641</td>
<td>-963soft</td>
<td>-826soft</td>
</tr>
<tr>
<td>6</td>
<td>1800</td>
<td>30</td>
<td>7281</td>
<td>-916soft</td>
<td>-792soft</td>
</tr>
<tr>
<td>7</td>
<td>3600</td>
<td>60</td>
<td>14562</td>
<td>-657soft</td>
<td>-792soft</td>
</tr>
<tr>
<td>8</td>
<td>7200</td>
<td>120</td>
<td>29124</td>
<td>-772soft</td>
<td>-757soft</td>
</tr>
<tr>
<td>9</td>
<td>14400</td>
<td>240</td>
<td>58248</td>
<td>-733soft</td>
<td>-727soft</td>
</tr>
<tr>
<td>10</td>
<td>28800</td>
<td>480</td>
<td>116496</td>
<td>-733soft</td>
<td>-727soft</td>
</tr>
</tbody>
</table>

Figure 7: Benchmark scores of default versus a structured initialiser

Note the initial difference between the initialisers is quite sizable. This directly translates into less processing required by the solver and also highlights the importance of a good, well-balanced solution initialiser. As the length of processing increases, the difference between the initialisers becomes less and less prominent. This is the reason why the initialiser, although important initially, is not responsible for the solving of 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 remain 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 nought (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.
The best possible score given 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 weigh the scores of the constraint rules as to favour 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.

### 5.2. Solution initialiser comparison

Here we look at the solutions after they were processed by the solver. 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 8 and Figure 9 respectively.

**Figure 8:** Solved schedule with default solution initialiser
Both solutions that were created are feasible in the sense that there were no negative hard score. The resulting schedules are almost of equal quality given that 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%. We should 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 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 you ignore other aspects and evaluate 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.

6. FUTURE WORK

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 must also be 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 paper.

7. CONCLUSION

Traditional corrective and blind preventive maintenance (follow schedules and ignore device conditions) methods can be 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. 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 were operated, specifically the levels of loads 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.

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 we simulated DC induction motors operating at different loads in order to get different maintenance thresholds. 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.

The results obtained from the simulation showed the suitability of the proposed e-Maintenance strategy for industrial maintenance applications.

8. ACKNOWLEDGEMENTS

The authors would like to thank the Central University of Technology (CUT, 2010), the Research Group in Evolvable Manufacturing (RGEMS, 2010), the Advanced Manufacturing Technology Strategy (AMTS, 2010) of the Council for Scientific and Industrial Research (CSIR, 2010) and the South African National Research Foundation (NRF, 2010) for supporting this research work.
9. REFERENCES


