



SAIIE29 Proceedings, 24th - 26th of October 2018, Spier, Stellenbosch, South Africa © 2018 SAIIE

A REAL-TIME SCHEDULING SYSTEM FOR A SENSORISED JOB SHOP USING CLOUD-BASED SIMULATION WITH MOBILE DEVICE ACCESS

S. Snyman^{1*}, J. Bekker² & J.P. Botha³

¹Department of Industrial Engineering
Stellenbosch University, South Africa
snymans@sun.ac.za

²Department of Industrial Engineering
Stellenbosch University, South Africa
jb2@sun.ac.za

³Department of Electrical and Electronic Engineering
Stellenbosch University, South Africa
jp17619084@gmail.com

ABSTRACT

Scheduling is a challenge that continues to trouble management of the operational phase of the manufacturing life cycle and can be attributed to the complex, dynamic and stochastic nature of a manufacturing system. Computer simulation is often used to assist with scheduling, as it can sufficiently mimic complex, discrete, dynamic, stochastic processes. We propose and develop a prototype real-time simulation scheduling system for a sensorised factory, which is to serve as a decision support tool for real-time rescheduling of machine steps in a job shop.

¹The author was enrolled for an MEng(Industrial) degree in the Department Industrial Engineering, Stellenbosch University, South Africa.

¹The author was enrolled for an MEng(Electrical and Electronic) degree in the Department Electrical and Electronic Engineering, Stellenbosch University, South Africa.

*Corresponding author

1. INTRODUCTION

Manufacturing is defined by Groover [1] as the transformation of materials into items of greater value by one or more processing and/or assembly operations. The transformation is accomplished by combining machinery, tools, power, and manual labour. From this description of manufacturing, the process can be considered a complex engineering endeavour, where the coordination of people, material, equipment, and information to accomplish a manufacturing goal demands considerable time and effort.

The manufacturing system life cycle can be divided into several phases which include the design, planning, implementation, operation and termination phases [2]. The coordination challenges faced in the design and implementation phases can be overcome by careful planning; however, such challenges continue to persist over the operational phase. For many manufacturing systems, scheduling is such a challenge, which can be attributed to the complex, dynamic, and stochastic environments exhibited by these systems. Many technological advances, including cloud-based computing and the improved capabilities of sensor networks, have offered the opportunity of designing a real-time scheduling system that can overcome the scheduling challenges, as well as the opportunity for the creation of software architectures to support these real-time scheduling systems.

This paper forms part of a research project with the objective to develop a prototype real-time simulation scheduler of a sensorised job shop, which is to serve as a decision support tool so that unexpected disturbances in the shop can be overcome by the generation of new schedules with real-time data from the shop. This paper was preceded by [3], which provided the relevant literature and initial architecture for the proposed solution. This paper will focus on the development and implementation of the proposed solution, as well as the testing and validation of the solution.

We subsequently provide a brief summary of the literature, as documented in [3], in Section 2, which is followed by a revised architecture of the proposed scheduler in Section 3. The development and implementation of the proposed solution will be described in Section 4, while the validation and operational testing will be discussed in Sections 5 and 6. Finally, the conclusion of the research project will be discussed in Section 7.

2. LITERATURE

The literature for this project was comprehensively discussed in [3], therefore only the most important literature will be reiterated. When considering scheduling in a job shop environment, there are a number of factors that need to be taken into account. These factors are described by [4] as:

- If more work is accepted per day than the organisation can complete per day, then the overall work in progress inventories will increase, causing shop congestion, an erosion of the firm's output rate, and a lengthening of job completion times.
- If completion times or dates are promised to customers, then estimates of lead times for each job must be determined, and jobs must be started early enough to complete the job by the promised date.
- Facilities can finish more jobs per period and satisfy more customers if they work on the shortest jobs first. However, longer jobs will ultimately be completed late or behind schedule.
- More highly valued customers may require earlier completion dates which will then be given processing priority in the shop. This will in turn make it more difficult to estimate accurate completion dates for other jobs.
- Interruptions such as machine breakdowns, employee absence, poor raw-material quality, and processing errors, can cause unforeseen delays in processing.

To address these factors, machine-level controlling measures, such as sequencing, dispatching rules and performance indicators, must be in place. The dispatch rules guide the production sequence of the jobs within the shop, and ensure that operators know the sequence in which the jobs must be processed. Dispatching rules, as mentioned by [4] and [5], include:

1. shortest process time,
2. first-come-first-served,
3. most-important-job-first,
4. earliest due date,
5. critical ratio, and
6. minimum slack time per operation.

The decision of which dispatching rule should be implemented is dictated by the job shop performance indicators that need to be optimised. The most common performance indicators were identified by [4] and [5] as:

- Average flow time, where the flow time begins when a job arrives at the shop and ends when it leaves. This flow time is averaged over a number of jobs.
- Average queue time, where the queue time is the total flow time minus the process time of the job. The queue time is then averaged over a number of jobs.
- Average job lateness, where lateness is the difference between the completion date and the due date. The lateness is averaged over a number of jobs.
- Average job tardiness, where tardiness is the amount of time a job finishes beyond the due date. Tardiness is averaged over a number of jobs.
- Makespan, where makespan is the total elapsed time to complete a number of jobs.

These dispatching rules and performance indicators are then used to define the job shop scheduling problem. The problem was previously mathematically defined and illustrated and therefore the reader is referred to [3]. This concludes the summary of the literature, and the following section will present the revised system architecture.

3. ARCHITECTURAL DESIGN OF THE PROPOSED SIMULATION SCHEDULER

We explain the concept using an informal schematic diagram, followed by a formal architectural description.

3.1 System overview

Figure 1 shows the system overview of all the components and how they interact with each other. The overview illustrates that there is a shop floor where different machines are installed. At each machine, a sensor is used for logging operation status changes. The sensors can log one of three possible states (*i.e. Waiting, Processing or Completed*). There is also a RFID card linked to a specific job, which can be swiped by the operator of the machine when a status change needs to occur. The state is then transmitted to a gateway which in turn transmits the state to the cloud-based information system. The scheduler (*i.e. Tecnomatix Plant Simulation*), also located in the cloud, consumes data from the information system and generates new schedules. The web interface can also use and log data on the information system. Finally, there is a mobile device that can access both the scheduler and the web interface.

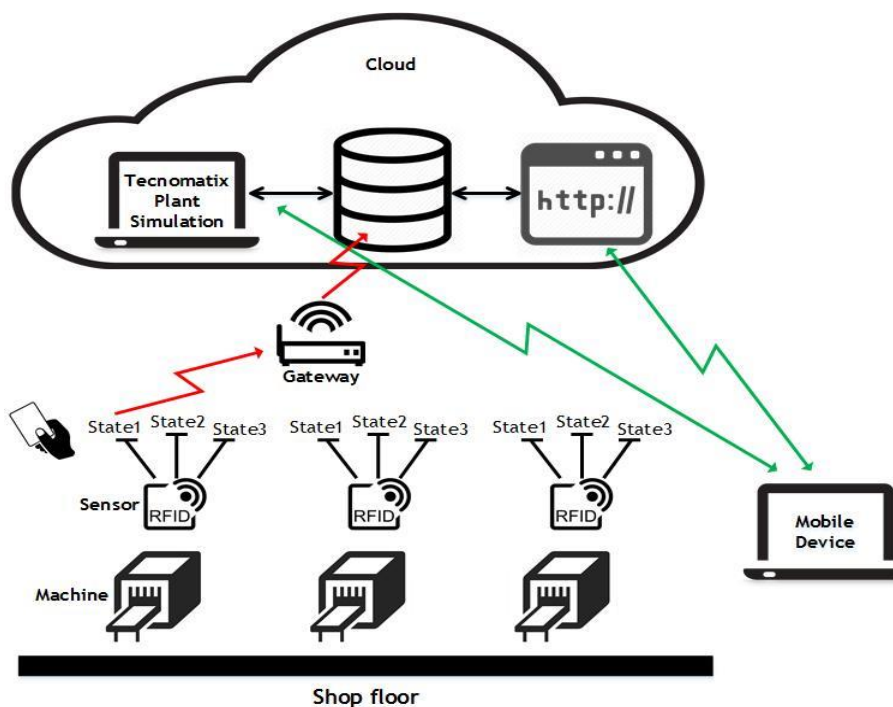


Figure 1: System overview

The initial architecture for the proposed scheduler was developed by [3], however this architecture was revised as illustrated in Figure 2. The architecture is described using the Object-Process Methodology (OPM) [6].

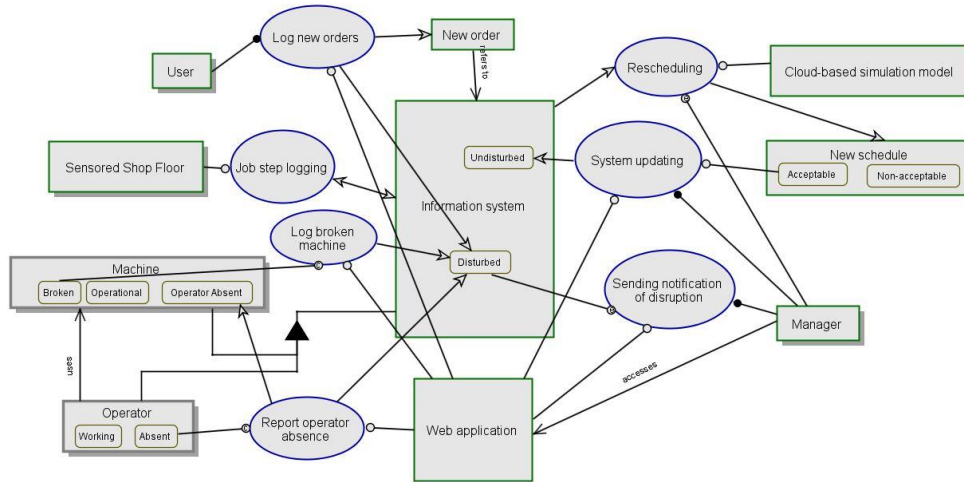


Figure 2: Top-level architecture of proposed scheduler (revised from [3]).

The semantical representation of the revised architecture is as follows:

User is physical.

User handles Log new orders.

New order refers to Information system.

Information system can be Undisturbed by default or Disturbed.

Information system consists of many Machines and many Operators.

Machine is physical.

Machine can be Operational by default, Broken, or Operator absent.

Operator is physical.

Operator can be Working by default or Absent.

Operator uses Machine.

Information system triggers Sending notification of disruption when it enters Disturbed.

Sensored shop floor is physical.

Manager is physical.

Manager accesses Web application.

Manager handles System updating and Sending notification of disruption.

Manager triggers Rescheduling.

New schedule can be Acceptable or Non-acceptable.

Log new orders requires Web application.

Log new orders yields Disturbed Information system and New order.

Log broken machine occurs if Machine is Broken.

Log broken machine requires Web application.

Log broken machine yields Disturbed Information system.

Report operator absence occurs if Operator is Absent.

Report operator absence requires Web application.

Report operator absence yields Disturbed Information system and Operator absent Machine.

Sending notification of disruption requires Web application and Disturbed Information system.

Rescheduling requires Cloud-based simulation model and Manager.

Rescheduling consumes Information system.

Rescheduling yields New schedule.

System updating requires Acceptable New schedule and Web application.

System updating yields Undisturbed Information system.

Job step logging requires Sensored shop floor.

Job step logging affects Information system

4. DEVELOPMENT AND IMPLEMENTATION

This section will describe the development and implementation of the different components of the proposed scheduling system. Reference will be made to the integration of the different hardware components into a

sensor prototype, as well as the software components which will include the information flow through the sensor and gateway, the information system, the web interface and the scheduler. Finally, the section will contain a system overview that will graphically illustrate the functionality of the proposed scheduler.

4.1 Hardware development and implementation

In this section the hardware components used to develop the sensor will be discussed with a complete explanation of the sensor integration.

4.1.1 Components

The main components that form part of the sensors and the additional components used to complete the hardware setup are discussed.

Raspberry Pi

The computer used to capture the sensor data and send it to the cloud is a Raspberry Pi, also referred to as the *system gateway*. This computer is used because it is well known and thus has sufficient software support. The Raspberry Pi runs on a *Debian* platform that forms part of the *Linux* operating system (OS) [7]. The *Linux* OS environment integrates seamlessly with the hardware of the Raspberry Pi through Python. The *serial peripheral interface* (SPI) bus of the Raspberry Pi is used to communicate to the radio frequency transceiver, RFM98. The RFM98 is discussed later in this section.

Arduino Pro Mini

The Arduino Pro Mini consists mainly of the Atmel Atmega328P micro-controller [8]. The operating voltage is 3.3V, which is ideal for this low-power sensor due to the use of only 3.3V components. The extremely low cost of this micro-controller with its functionality that supports the sensor outcomes made it the ideal choice [9]. The Arduino serves as the controller of the sensor, which captures the data from the RFID module and buttons, and sends this data to the RFM98 module. The RFM98 and RFID modules communicate via the SPI bus lines to the Arduino. The buttons trigger *general purpose input output* (GPIO) pins on the Arduino for user interface.

LoRa (RFM98)

The RFM98 module is a breakout board that holds the Semtech LoRa *integrated circuit* (IC). This module is used due to its compact design and design support. LoRa, short for Long Range, is a long-range, low-power radio frequency communication platform which is at the forefront of technology in network communication [10]. The confirmed line of sight communication distance is between 15 and 20 km [11]. It makes use of the low-frequency bandwidth that is part of the unlicensed radio spectrum, thus it is free to use, which holds great advantages. One disadvantage with this technology is that due to its low frequency the communication speed is limited [12]. However, this application makes use of low data rates which makes the RFM98 module the ideal choice. The module is used with the Raspberry Pi and Arduino Pro Mini to establish the communication network.

RFID (RC522)

The RFID module that is used in this application is the RC522 module. It identifies and communicates card or tag ID data to the Arduino via the SPI bus. In this application passive cards and tags are used as they are the cheapest and do not require a battery. The cards and tags utilise the radio energy that is transmitted by the reader to energise its circuit and to provide its ID [13].

Programmer

In this application an external programmer is used as it reduces the cost of each sensor [8]. The programmer consists of the FTDI FT232 UST to serial IC. The programmer is used to program the Arduino with its application specific functionality. The programmer makes use of the serial connection on the Arduino to write the developed code onto the Atmel micro-controller.

Power Supply

The sensors were developed to be seamlessly integrated into the proposed system by means of battery power. To make this possible and user-friendly a charging module and 18650 Lithium-Ion battery are added to the

sensors. The battery has a capacity of 2600 mAh and is widely available [14]. A typical power bank charging module is used with the battery. It enables the user to easily charge the sensor through USB.

4.1.2 Gateway design

The gateway consists of the Raspberry Pi and a RFM98 module. A small *printed circuit board* (PCB) breakout board for the RFM98 was designed to connect directly onto the Raspberry Pi. The software used to design this PCB was *Altium Designer*. This was done to eliminate prototype wires and to ensure the stability of the gateway with good component connections. Figure 3 illustrates the PCB with numbers and descriptions as follows:

1. The connection pins that connect to the Raspberry Pi to the LoRa module to receive the data from the module through the SPI bus.
2. The LoRa module that receives the data from the sensors.

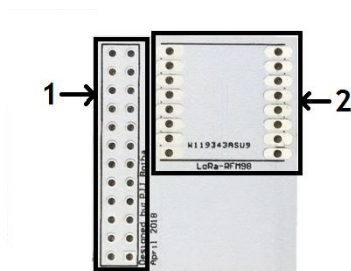


Figure 3: Designed PCB for gateway

4.1.3 Sensor design

The sensor consists of the Arduino Pro Mini, LoRa, RFID and the power supply modules. A PCB was designed to connect all of these components. The PCB serves as the sensor component integration that also has the advantage of good connections, which provide sensor stability. Figure 4 illustrates the front and back of the developed PCB with numbers and descriptions. The components are matched to the numbers as follows:

1. The RFID that reads the ID on the card or tag and sends this data to the Arduino through the SPI bus.
2. The three buttons that enable the user to define the status (*i.e. Waiting, Processing, Completed*) of an operation.
3. The charging module of the power supply that supplies power to the sensor and charges the Lithium-Ion battery.
4. The LoRa module that handles all the communication through low-power and low-frequency communication.
5. The Arduino Pro Mini that serves as the master of all the connected components.

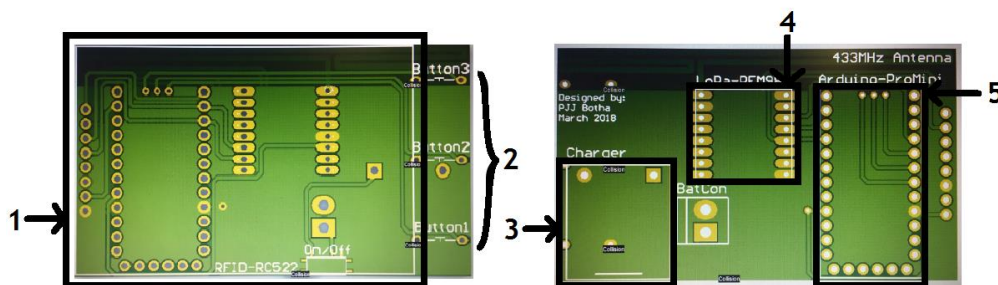


Figure 4: Front and back view of the developed PCB for sensor

A casing was also designed in *Autodesk Inventor*, as illustrated in Figure 5, to enclose the sensor and all its components. The casing holds the battery safely underneath the sensor. Push buttons are designed to enable the easy use of the input buttons on the sensor within the case. The casing is 3D printed.

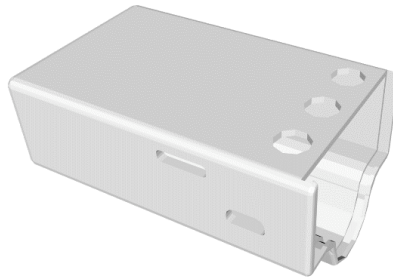


Figure 5: CAD design of sensor casing

4.2 Software development and implementation

The development of the software components of the proposed scheduler will now be discussed.

4.2.1 Information flow in sensor and gateway

The information flow through the sensor is illustrated by the flow diagram in Figure 6. The information flow starts off when the operator presses the button which indicates the new status and processes the ID of the button to the Arduino. The operator can then swipe the RFID card linked to the job that is then read by the RFID reader which processes the ID of the card to the Arduino. Only when both the button ID and card ID are received by the Arduino can a data package be created. The data package is then communicated to the LoRa module, which will transmit the package to the gateway. The data package will consist of a string of numbers that includes the sensor ID, card ID and button ID, where the sensor ID corresponds to the machine ID it is located at.

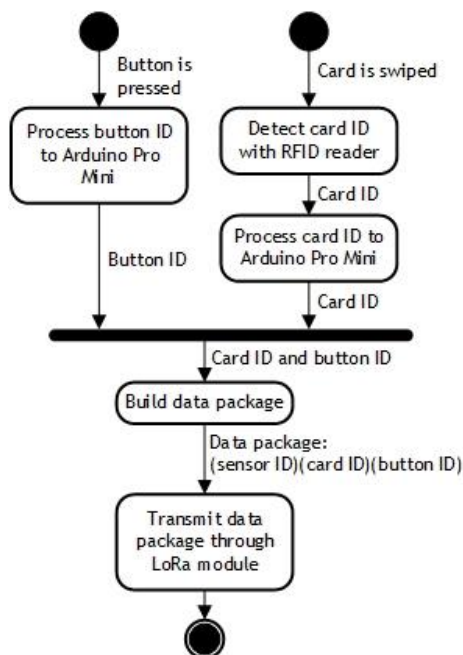


Figure 6: Information flow through sensor

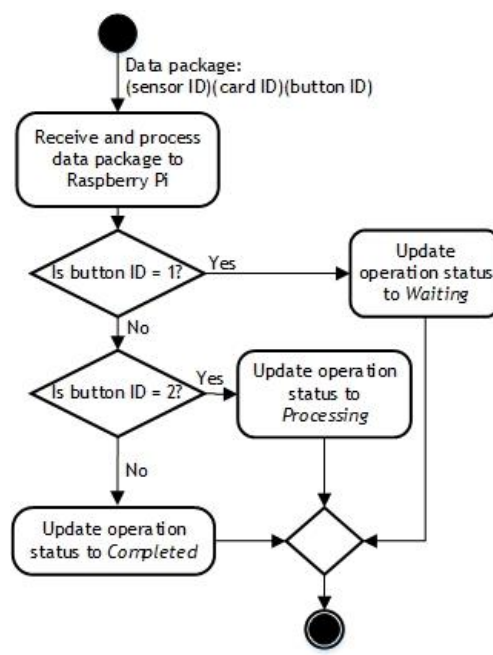


Figure 7: Information flow through gateway

The information flow through the gateway is also illustrated through the flow diagram in Figure 7 and starts off when the LoRa module receives the data package from the sensor. This package is processed by the LoRa module to the Raspberry Pi which then logs the data change in the cloud-based information system. The Raspberry Pi performs this data change at the operation that is associated with the machine ID and card ID provided in the data package. If the button ID = "1", the status of the operation will be updated to *Waiting*; while if the button ID = "2", the status of the operation will be updated to *Processing*. If neither of these conditions is met, the button ID must be "3" and the status of the operation will be updated to *Completed*.

4.2.2 Information system and web interface design

The information system is the database where all data from the job shop is recorded and stored. It is necessary to track the status of the shop floor, as it is required by the digital twin (the simulation model) when scheduling the existing set of jobs in progress. Due to the system requirement which stipulates that the information system must be cloud-based, a cloud server needed to be created. *Google Cloud Platform* (GCP) was chosen as a cloud-computing service provider which provided sufficient functionality required by the authors, and due to GCP making use of *MySQL* as the information system development tool, a *MySQL* server was created. *MySQL* is an open-source, relational database system which enables the delivery of reliable, high-performance and scalable web-based applications [15].

The information system was created for the documentation of information regarding machines, operators, jobs and operations in the shop which can either be used by the scheduler or displayed on the web interface. The web interface is what the user of the system will use to log data changes, view scheduler results and choose new schedules. The web interface was created through *Microsoft Visual Studio* and the functionality of the web pages that were incorporated into the interface include:

- logging of a broken machine,
- logging of a new job entering the shop,
- changing of an operator status,
- displaying of scheduler results,
- manual capturing of sensor data,
- addition or withdrawal of a machine, and
- selection of a new schedule.

4.2.3 Simulation scheduler

The simulation scheduler is the component in the proposed solution that generates new schedules according to the different dispatching rules. New schedules are created each time the shop floor is disturbed by events such as a machine failure, a new order arriving, an operator being absent, *etc.* First of all, the scheduler is required to be in the cloud, therefore a cloud server was created to run the model. Following the creation of the server, the model was created with the use of *Tecnomatix Plant Simulation* which is a simulation software package. The scheduler incorporated different elements, which include:

1. *MySQL* data import, which is used to import data from the *MySQL* server created for the information system. This will ensure that the scheduler uses the current state of the shop.
2. Dispatching rules, which are used by the scheduler to generate machine schedules according to the different dispatching rules.
3. Machine schedules, which are the data tables the machines refer to, to determine the next operation to start processing.
4. Machine behaviour, which includes the various machines and ensures that each machine adheres to the sequences in the generated schedules.
5. Experiment inputs, which are controlled by a variable in the scheduler. The variable informs the scheduler as to which dispatching rule must be used.
6. Experiment outputs, which are the results of the performance indicator calculations for each dispatching rule. These values will be compared to the corresponding values of the other dispatching rules, and the best performing dispatching rule can then be identified.

5. DEVELOPMENT VALIDATION

This section will serve to describe the validation and testing of the developed system. Reference will be made to the validation of the scheduler as well as the validation of the sensor and gateway.

5.1 Scheduler validation

The validation of the scheduler starts off by defining test scenarios that are used to determine whether the dispatching rule generated the correct schedules. Thereafter the results of the generated schedules can be used to determine the best-performing dispatching rule for each performance indicator. The test scenarios that were defined include:

- **Scenario one:** five jobs are entered into the system, where four jobs have similar expected processing times. There is, however, one job which is a clear outlier because the processing times of its operations are considerably longer than those of the other jobs. (A small number of jobs is selected in order to be able to analyse the results manually.)
- **Scenario two:** five jobs are entered into the system, each having similar expected processing times.
- **Scenario three:** five jobs are entered into the system, each with exactly the same expected processing time.
- **Scenario four:** 50 jobs are entered into the system, having similar expected processing times. This scenario is used to test the robustness of the scheduler, i.e. whether it can handle a relatively large number of jobs.

It was decided that there will be no machine failures or absent operators during these tests. Each of these scenarios was then run by the scheduler and a schedule was generated for each dispatching rule. For explanation purposes and due to space limitations, only the schedule generated for the shortest processing time dispatching rule of test scenario one will be compared to a manually drawn schedule, to determine whether it generated the correct sequence. First the manually drawn schedule was created using the data from the information system, where meticulous care was taken to follow the dispatching rule. Figure 8 illustrates the schedule that was manually drawn. The schedule that was generated for test scenario one using the shortest processing time dispatching rule is illustrated in Figure 9, where the different machines are denoted by M1-M8. Figure 9 also illustrates the shift start and end time (*i.e.* 08:00 and 17:00), between which the machines will not accept any new operations. If there is an operation that already started on the machine before 17:00, the machine will continue processing until the operation is finished, but it will not accept a new operation. The generated schedule was then analysed to determine whether the sequence in which jobs were processed was correct, by comparing it to a manually drawn schedule for the same test scenario and dispatching rule. Both figures illustrate the same schedule which means that the scheduler did adhere to the dispatching rule and generated the correct schedule. The only difference between the schedules is that the manually drawn schedule does not incorporate the stoppage time between shifts, while the schedule generated by the scheduler does incorporate the stoppage time. This same evaluation process was done for each schedule that was generated, however, due to space limitations it will not be shown.

The same process was repeated for each dispatching rule and test scenarios, which resulted in a total of six schedules that were generated per test scenario. The schedules from the different dispatching rules could then be compared when referring to each performance indicator, and the best-performing dispatching rule could then be identified. The results from each test scenario are shown in Table 1, Table 2, Table 3 and Table 4. The results include the performance indicator value, in “days:hours:minutes:seconds”, for each dispatching rule. Depending on the performance indicator, the user can select the best performing dispatching rule by finding the shortest duration for that performance indicator. This concludes the validation of the scheduler; following is the validation of the sensors and gateway.

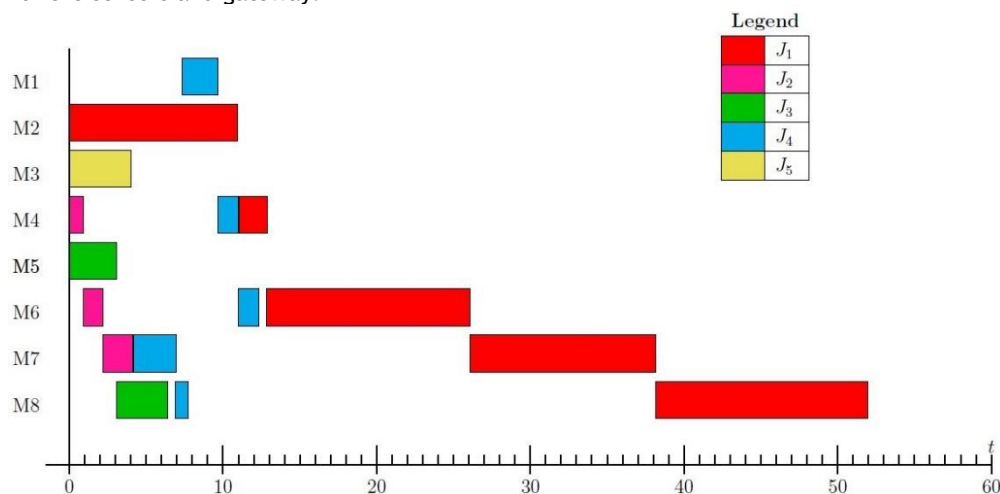


Figure 8: Schedule drawn manually through shortest processing time dispatching rule for Test scenario 1

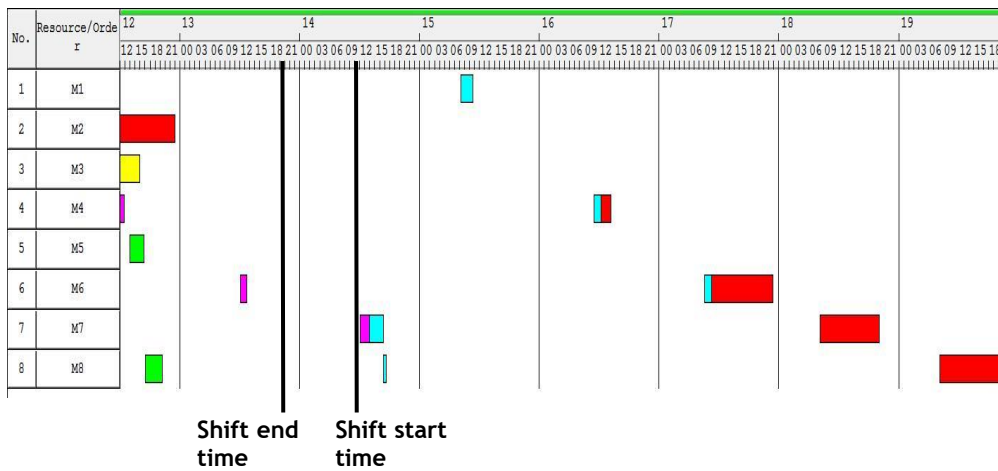


Figure 9: Schedule generated through shortest processing time dispatching rule for test scenario one

Table 1: Test scenario 1 results

Dispatching Rule	Average Flow Time	Average Queue Time	Average Job Tardiness	Average Job Lateness	Makespan
Shortest processing time	9:21:57:13	9:07:02:19	05:36	-2:12:29:38	14:08:08:57
First-come-first-serve	10:05:56:27	9:15:01:33	42:17	-3:00:36:47	11:20:25:03
Most-important-job-first	10:00:20:30	9:09:26:18	00:30	-2:13:01:40	14:07:46:39
Earliest due date	9:05:11:34	8:14:17:36	27:53	-3:02:07:11	11:20:20:43
Critical ratio	10:05:50:39	9:14:55:46	41:59	-3:00:36:49	11:20:19:23
Minimum slack time	9:23:00:27	9:08:05:33	55:15	-2:12:48:52	14:07:48:57

Table 2: Test scenario 2 results

Dispatching Rule	Average Flow Time	Average Queue Time	Average Job Tardiness	Average Job Lateness	Makespan
Shortest processing time	1:07:02:14	16:53	0	-3:02:32:16	2:01:51:34
First-come-first-serve	1:04:07:26	22:05	0	-2:22:58:57	1:22:51:11
Most-important-job-first	1:04:49:25	03:27	0	-3:01:21:32	2:00:57:56
Earliest due date	1:01:19:29	34:00	0	-3:09:36:49	1:09:32:53
Critical ratio	1:05:49:46	04:24	0	-3:04:34:41	1:17:45:17
Minimum slack time	1:04:31:22	46:01	0	-3:01:25:26	2:00:36:51

Table 3: Test scenario 3 results

Dispatching Rule	Average Flow Time	Average Queue Time	Average Job Tardiness	Average Job Lateness	Makespan
Shortest processing time	1:05:33:34	47:28	0	-2:22:17:24	2:00:06:06
First-come-first-serve	1:04:51:38	05:32	0	-2:22:53:06	1:22:32:52

Most-important-job-first	1:04:16:50	30:45	0	-3:01:27:08	2:00:11:56
Earliest due date	1:03:48:45	02:39	0	-3:04:45:58	1:17:02:09
Critical ratio	1:07:20:16	34:11	0	-3:02:50:03	1:23:46:05
Minimum slack time	1:04:11:05	24:59	0	-3:01:27:31	2:00:04:44

Table 4: Test scenario 4 results

Dispatching Rule	Average Flow Time	Average Queue Time	Average Job Tardiness	Average Job Lateness	Makespan
Shortest processing time	7:14:05:41	7:06:19:44	21:28	-13:12	14:22:40:41
First-come-first-serve	7:15:35:53	7:07:49:56	30:00	34:21	13:22:19:15
Most-important-job-first	7:23:10:02	7:15:24:03	22:28	09:52	14:20:53:59
Earliest due date	8:11:02:44	8:03:16:51	0	-3:34:30	15:06:17:49
Critical ratio	7:19:52:56	7:12:06:59	0	-4:03:59	14:06:09:51
Minimum slack time	6:12:47:58	6:05:02:01	11:09	-3:34:29	15:03:51:36

5.2 Sensor and gateway validation

The validation of the sensors and gateway entails testing whether the status of an operation changes in the IS when the sensor is evoked. To accomplish the validation, a job was entered into the system which was then allocated several operations and linked to the ID of a RFID card. All the operations of the job will have a status of *Pending* at the start of the test. Having the job arrive at the machine of its first operation, it is expected that the status of that operation should change to *Waiting*. To accomplish the status change, the button corresponding to the status *Waiting* was selected and the RFID card linked to the job was swiped. The information system was consulted after the card was swiped, and it could be concluded that the sensor succeeded in changing the operation status. This concludes the development validation section. The following section will discuss the operational testing of the scheduler.

6. OPERATIONAL TESTING

For the operational testing of the system, the authors logged disturbances in the information system to determine what effect they will have on the schedules. The disturbances that were tested, include:

- logging a machine as broken,
- logging an operator as absent, and
- adding a new job with many operations.

For the purpose of the operational testing, a test scenario was created with ten jobs that have exactly the same expected processing times. The results of this scenario, where there are no disruptions, are illustrated in Table 5. These results can be used as reference values for the results of the tests where disruptions occurred. The results include the performance indicator value, in “days:hours:minutes:seconds”, for each dispatching rule.

Table 5: Performance indicator results with no disruptions

Dispatching Rule	Average Flow Time	Average Queue Time	Average Job Tardiness	Average Job Lateness	Makespan
Shortest processing time	2:09:49:58	2:02:03:52	9:21:37:52	11:29	4:02:15:18
First-come-first-serve	1:22:59:25	1:15:13:20	9:13:25:35	2:14:32:35	3:03:15:51

Most-important-job-first	1:20:09:22	1:12:23:16	9:08:26:43	26:38	3:21:54:20
Earliest due date	1:12:56:36	1:05:10:30	9:03:53:09	21:57	2:19:39:05
Critical ratio	1:23:20:17	1:15:34:11	9:11:54:31	34:16	3:19:39:05
Minimum slack time	1:18:22:08	1:10:36:03	9:08:55:00	19:33	3:00:09:30

When the authors logged a machine as *Broken*, it is expected that the scheduler will start with the broken machine as unavailable and not utilise the machine before it is operational again. The jobs will therefore be allocated to the identical machine, if applicable. However, when the machine is operational again, the scheduler should divide the remaining jobs allocated to the identical machine between the two machines. Table 6 provides the results of the different performance indicators when one of the milling machines (*i.e.* M3) was broken for four days. From the table it is evident that the scheduler changed the schedule to accommodate the disruption, because the values are longer than those where no disruption occurred.

Table 6: Performance indicator results when M3 is broken for four days

Dispatching Rule	Average Flow Time	Average Queue Time	Average Job Tardiness	Average Job Lateness	Makespan
Shortest processing time	3:04:57:13	2:21:11:07	10:17:44:59	10:17:44:59	5:00:55:18
First-come-first-serve	2:05:19:51	1:21:33:45	9:19:28:48	13:34	3:17:09:10
Most-important-job-first	2:08:01:38	2:00:15:33	9:20:53:08	9:20:53:08	4:17:04:22
Earliest due date	2:06:15:33	1:22:29:27	9:20:22:52	24:20	3:17:59:05
Critical ratio	2:02:32:48	1:18:46:42	9:16:41:49	02:25	3:17:59:05
Minimum slack time	2:03:12:48	1:19:26:43	9:19:03:00	37:14	3:01:24:15

When the authors logged an operator as *Absent*, it is expected that the scheduler will start with the machine where that operator is located as unavailable and not utilise the machine before the operator is available again. The jobs will therefore be allocated to the identical machine, if applicable. However, when the machine is operational again, the scheduler should divide the remaining jobs allocated to the identical machine between the two machines. Table 7 provides the results of the different performance indicators when an operator at a grinding machine (*i.e.* M6) was absent for three days. From the table it is evident that the scheduler changed the schedule to accommodate the disruption, because the values are longer than those where no disruption occurred.

Table 7: Performance indicator results when an operator is absent for three days

Dispatching Rule	Average Flow Time	Average Queue Time	Average Job Tardiness	Average Job Lateness	Makespan
Shortest processing time	2:10:21:38	2:02:35:32	10:00:00:50	27:02	4:00:35:18
First-come-first-serve	2:22:42:53	2:14:56:47	10:11:50:54	10:11:50:54	4:22:06:04
Most-important-job-first	2:00:35:46	1:16:49:41	9:14:22:59	24:58	3:23:48:46
Earliest due date	2:02:22:54	1:18:36:48	9:16:36:45	13:34	3:19:39:08
Critical ratio	2:02:09:33	1:18:23:27	9:16:26:22	11:26	3:19:26:20
Minimum slack time	2:04:24:39	1:20:38:33	9:18:47:48	21:06	3:18:39:40

Finally, when the authors added a new job with several operations to the system, it is expected that the schedule generated by the scheduler will incorporate the new job into the new schedule. The job that was added has a total processing time that is similar to those of the other jobs. Table 8 provides the results of the different performance indicators when a new job was added to the system, and from the results it is evident that the scheduler changed the schedule to accommodate the disruption, because the values are longer than those where no disruption occurred.

Table 8: Performance indicator results when a new job was added

Dispatching Rule	Average Flow Time	Average Queue Time	Average Job Tardiness	Average Job Lateness	Makespan
Shortest processing time	2:15:22:12	2:07:10:18	10:16:38:43	2:09:50:55	4:21:30:00
First-come-first-serve	2:01:51:21	1:17:39:26	10:05:31:54	3:19:34:50	3:19:00:00
Most-important-job-first	2:19:47:19	2:11:35:25	10:20:37:28	10:20:37:28	5:16:29:10
Earliest due date	2:02:50:59	1:18:39:04	10:04:48:49	09:32	3:17:36:41
Critical ratio	2:05:04:38	1:20:52:43	10:06:24:40	57:54	4:00:51:08
Minimum slack time	2:02:27:55	1:18:16:01	10:03:51:11	42:36	4:00:34:20

Each of these disturbances that were tested, proved that the scheduler adapted and generated a new schedule that incorporated each disturbance. The scheduler can therefore be assumed to be working correctly. This concludes the operational testing of the scheduler and the following section will provide the conclusion of the research project.

7. CONCLUSION

This paper described the development and testing of a prototype of a real-time simulation scheduling system for a sensorised job shop. A revised architecture of the proposed system was provided, which describes the working of the system. The different hardware and software components required for the development of the system, were also identified and discussed. Finally, the validation and operational testing of the system was discussed. It was our purpose to build a real-time simulation scheduler of a sensorised job shop so that unexpected disturbances in the shop can be overcome by the generation of new schedules with real-time data from the shop. The results from the testing of the system proved that the authors were successful in creating the desired system, and ultimately succeeded in fulfilling the purpose of the research project.

REFERENCES

- [1] Groover, M. P. 2013. *Principles of modern manufacturing*, 5th Edition, US: John Wiley and Sons.
- [2] Mitsubishi, M., Ueda, K. & Kimura, F. 2008. *Manufacturing systems and technologies for the new frontier: The 41st CIRP Conference on Manufacturing Systems May 26-28, 2008, Tokyo, Japan*. London: Springer.
- [3] Snyman, S., Bekker, J. 2017. Real-time scheduling in a sensorised factory using cloud-based simulation with mobile device access, *South African Journal of Industrial Engineering*, 28(4), pp 161-169.
- [4] Wisner, J. D. 2016. *Operations Management: A supply chain process approach*. California: SAGE Publications.
- [5] Dominic, P. D. D., Kaliyamoorthy, S. & Kumar, M. S. 2004. Efficient dispatching rules for dynamic job shop scheduling, *International Journal of Advanced Manufacturing Technology*, 24(1), pp. 70-75.
- [6] Dori, D. 2002. *Object-Process Methodology: A Holistic Systems Paradigm*. New York: Springer.
- [7] Raspberry Pi. 2014. Teach, learn, and make with Raspberry Pi, [accessed 20 November 2017], available at: <https://www.raspberrypi.org/help/faqs/>
- [8] Arduino. 2017. Getting started with the Arduino Pro Mini, [accessed 1 November 2017], available at: <https://www.arduino.cc/en/Guide/ArduinoProMini>
- [9] Scott Tattersall. 2018. How to build custom IoT hardware with Arduino, [accessed 10 February 2018], available at: <https://opensource.com/article/17/12/how-build-custom-iot-hardware-arduino>
- [10] Semtech. 2018. Lora transceivers, [accessed 10 February 2018], available at:



SAIIE29 Proceedings, 24th - 26th of October 2018, Spier, Stellenbosch, South Africa © 2018 SAIIE

- [11] <https://www.semtech.com/products/wireless-rf/lora-transceivers>
Murata. 2018. Lora module FAQ, [accessed accessed 10 February 2018], available at: <https://www.murata.com/support/faqs/products/lpwa/lora/hardware/0008>
- [12] RF Wireless World. 2017. Advantages and Disadvantages of Lora or LoRaWAN, [accessed 5 November 2017], available at: <http://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Lora-or-LoRaWAN.html>
- [13] NXP. 2016. MFRC522 data sheet, [accessed 1 December 2017], available at: <https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf>
- [14] AA Portable Power Corp. 2004. Specification Cylindrical Li-ion 18650 2600mAh Rechargeable cell, [accessed 1 March 2018], available at: http://www.batteryspace.com/prod-specs/18650_2600.pdf
- [15] Oracle. 2013. MySQL, [accessed 10 October 2017], available at: <http://www.oracle.com/technetwork/database/mysql/index.html>