

American University in Cairo

AUC Knowledge Fountain

Theses and Dissertations

6-1-2016

Scheduling of pipeline construction projects using simulation

Hany Mohsen Zahran

Follow this and additional works at: <https://fount.aucegypt.edu/etds>

Recommended Citation

APA Citation

Zahran, H. (2016). *Scheduling of pipeline construction projects using simulation* [Master's thesis, the American University in Cairo]. AUC Knowledge Fountain.

<https://fount.aucegypt.edu/etds/280>

MLA Citation

Zahran, Hany Mohsen. *Scheduling of pipeline construction projects using simulation*. 2016. American University in Cairo, Master's thesis. *AUC Knowledge Fountain*.

<https://fount.aucegypt.edu/etds/280>

This Thesis is brought to you for free and open access by AUC Knowledge Fountain. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AUC Knowledge Fountain. For more information, please contact mark.muehlhaeusler@aucegypt.edu.

The American University in Cairo
School of Sciences and Engineering

SCHEDULING OF PIPELINE CONSTRUCTION PROJECTS USING
SIMULATION

A Thesis Submitted to
Department of Construction Engineering

in partial fulfillment of the requirements for
the degree of Master of Science

by Hany Mohsen Zahran

(under the supervision of Dr. Khaled Nassar)
January/2015

ABSTRACT

Scheduling of Pipeline Construction Projects using Simulation

Hany Mohsen Zahran

American University in Cairo

Repetitive Projects represent a large percentage of construction projects. They usually have an immense importance for a nation's economy and future. Highways, tunnels, infrastructure networks, high-rise buildings, housing projects, pipeline networks, airport runways, railways, bridges, sewer mains and mass transit systems are all considered projects of repetitive nature. Research that started to serve industrial purposes for the military efforts in World War II has been revised and improved to be employed for repetitive construction projects. Obtaining an optimum schedule that would be achievable, feasible, and comprehensive by all involved parties besides maintaining minimum overall cost and duration has been an important objective. Another main objective was to maintain an optimal formation of various types of crews and equipment that would avoid idle periods as well as work stoppages. Various examples of mathematical models presented in the literature were presented as an example to show their limitations. This research presents a simulation-based scheduling model for pipeline construction projects. The model was developed with a simulation software called "AnyLogic"; this software supports discrete events, agent based and system dynamics simulation, presents an easy graphical user interface and utilizes Java coding. The model consists of various types of pre-programmed objects that were used and connected together to model the different stages of the project and resources involved within them. The model also contains a simulation experiment that would be used to provide the visual presentation of the construction process including the layout of the project and all kinds of utilized resources moving within it. The final part of the model is the optimization module. This module has the definition of the optimization objective, the optimization parameters and constraints. This module would run the simulation experiment a numerous trials while changing the parameters to get the optimal solution which is the optimal schedule for the project. This simulation model would aid planners in scheduling, tracking and controlling the construction operations over the lifetime of the project. It would present an important tool for top management to visualize the impact of their decisions.

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude and sincere appreciation to my supervisor Dr. Khaled Nassar, for his unconditional support, valuable advice, constructive guidance and encouragement throughout all stages of this study. His effort and suggestions to improve the contents of this thesis are greatly appreciated. I would also like to express appreciation to each of my committee members for the time and effort they gave to my comprehensive and final defense.

I am grateful and will always be indebted to my family, and especially my future wife, who always encouraged me, believed in me and provided unlimited support during the course of this study.

TABLE OF CONTENTS

LIST OF FIGURES.....	vi
----------------------	----

LIST OF TABLES.....	viii
---------------------	------

CHAPTER 1

INTRODUCTION.....	1
1.1. Introduction.....	1
1.2. Pipeline Construction scheduling.....	3
1.3. Considerations in the Planning and Control of Pipeline Construction	4
1.4. Problem Statement.....	5
1.5. Research Objectives.....	5
1.6. Thesis Organization.....	6

CHAPTER 2

LITERATURE REVIEW	7
2.1. Introduction.....	7
2.2. Traditional Scheduling Techniques.....	7
2.2.1. Bar Chart Method	7
2.2.2. Network Techniques.....	8
2.3. Techniques for Scheduling Repetitive Activities	9
2.3.1. Line of Balance (LOB)	10
2.3.2. Linear Scheduling Method (LSM)	11
2.4. Optimized Scheduling.....	11
2.4.1. Operations Research Models	12
2.4.2. Simulation Models	13
2.5. Summary and Conclusion.....	14

CHAPTER 3

STAGES OF PIPELINE CONSTRUCTION	15
3.1 Introduction.....	15
3.2 Stages of Pipeline Construction.....	15
3.3 Summary and Conclusion.....	23

CHAPTER 4

DETERMINISTIC MODELS.....	24
----------------------------------	-----------

4.1	Introduction	24
4.2	Methodology	24
4.3	Proposed Models.....	25
4.3.1	Johnson's Rule	25
4.3.2	Mixed Integer Programming	27
4.3.3	Slope Heuristic	31
4.4	Summary and Conclusion.....	33
CHAPTER 5		
SIMULATION MODEL		34
5.1.	Introduction	34
5.2.	Simulation Software	36
5.3.	Model Development.....	36
5.4.	Summary and Conclusion.....	75
CHAPTER 6		
OPTIMIZATION MODEL		77
6.1.	Introduction	77
6.2.	Optimization Experiment.....	82
6.3.	Case Study Description	89
6.4.	Application of Model	91
6.5.	Validation of Model	96
6.6.	Conclusion	97
CHAPTER 7		
CONCLUSIONS.....		98
7.1.	Summary of Research	98
7.2.	Research Contributions	99
7.3.	Recommendations for Future Research.....	100
Appendix A		102
Appendix B		106
Appendix C		110
REFERNCES		113

LIST OF FIGURES

CHAPTER 1

Figure 1-1 : Nord Stream and South Stream Pipelines.....	2
Figure 1-2: Pipeline Construction Quantities in 2013.....	3

CHAPTER 3

Figure 3-1: Flowchart of Spread Method Activities	16
Figure 3-2 : Pipe Stringing Activity	19
Figure 3-3 : Pipe Welding Activity	20
Figure 3-4 : Trench Excavation Activity.....	21
Figure 3-5 : Lowering pipes in Trenches.....	22
Figure 3-6: Pipeline Construction Operation	23

CHAPTER 4

Figure 4-1 : MS Excel Worksheet Used for (MIP)	30
--	----

CHAPTER 5

Figure 5-1: Typical Interface in AnyLogic.....	37
Figure 5-2: Palette View Window	38
Figure 5-3: Layout of Pipeline Project in AnyLogic	39
Figure 5-4: First Window in “New Java Class” wizard.....	41
Figure 5-5: Second Window in “New Java Class” wizard	42
Figure 5-6: Properties View for “Pline” Resource Pool	46
Figure 5-7: Resource Pools Diagram.....	49
Figure 5-8: Complete Flowchart of Model	49
Figure 5-9: Stage (1) – Creating Entities.....	55
Figure 5-10: Stage (2) - Pipe Stringing	61
Figure 5-11: Stage (3) - Pipe Bending and Welding	65
Figure 5-12 : Stage (4) - Weld Inspection and Repair.....	68
Figure 5-13: Stage (5) - Joints Coating, Trench Excavation and Pipe Lowering	73
Figure 5-14: Stage (6) - Joints Welding and Weld Inspection	75
Figure 5-15: Stage (7) - Trench Backfilling and Hydrotesting	79

CHAPTER 6

Figure 6-1: Properties View of Optimization Model.....	82
Figure 6-2: Setting the Optimization Objective.....	83

Figure 6-3: Defining Optimization Parameters	85
Figure 6-4: Constraints Table in Properties View.....	91
Figure 6-5: Graphical Presentation for Optimization Model	92
Figure 6-6: Map of South Valley Gas Pipeline	93
Figure 6-7: Results of 50 Simulation Runs.....	95
Figure 6-8: Second Stage Optimization Model	97
Figure 6-9: Example of Used Sub-models	98

LIST OF TABLES

CHAPTER 4

Table 4-1: Job durations in man-hours	25
Table 4-2: Completion times for jobs in man-hours	26
Table 4-3: Dividing jobs into two sets.....	26
Table 4-4: Completion times for jobs after applying Johnson's rule (In man-hours).....	27
Table 4-5: Job durations in man-hours	28
Table 4-6: Completion times for jobs in man-hours	29
Table 4-7: Decision Variable Matrix (x _{jk})	29
Table 4-8: Waiting Time Matrix (Wik).....	30
Table 4-9: Idle Time Matrix (lik)	30
Table 4-10: Completion times for jobs in man-hours	31
Table 4-11: Job durations in man-hours	32
Table 4-12: Completion times for jobs in man-hours	32
Table 4-13: Slope Index for jobs	32
Table 4-14: Completion times for jobs in man-hours	33

CHAPTER 5

Table 5-1: Names and Animation Shapes of Model Components	40
Table 5-2: New Fields of Entity Class	43
Table 5-3: Arrays Properties	53

CHAPTER 6

Table 6-1: Properties of "stationProps1"	84
Table 6-2: Values of Optimization Parameter "Strng1"	85
Table 6-3: Properties of Arrays "stationProps2" to "stationProps14"	87
Table 6-4: Values of Optimization Parameters	90
Table 6-5: Definition of 1 st Constraint for Stringing Optimization Parameters	90
Table 6-6: Definition of 2 nd Constraint for Stringing Optimization Parameters	91
Table 6-7: Constraints for Optimization Parameters	92
Table 6-8: Second Stage's Resulting Sequence of Work for Pipeline Stations	97
Table 6-9: Final Sequence of Work for Pipeline Stations.....	98

Chapter 1

Introduction

1.1. Introduction

Pipelines transport crude oil and raw natural gas over long distances from producing regions to refineries and processing plants, where these energy sources are converted into useful fuel types such as gasoline, diesel and commercial-grade natural gas. Pipelines are also used to transport these consumer-ready fuels from refineries and gas processing plants to large terminals on the edge of towns and cities, where they can then be distributed to homes and businesses. Pipelines are utilized for many reasons such as:

- Pipelines are more cost-effective than the alternative transportation options such as tanker trucks or rail cars.
- They are more stable and reliable as they are not affected by any conditions such as road or weather conditions
- They require significantly less energy to operate than operating trucks or rail and thus, have a much lower carbon footprint.

Pipelines for major energy resources (petroleum and natural gas) are not merely an element of trade. They connect to issues of geopolitics and international security as well. The construction, placement, and control of oil and gas pipelines often take an important position in state interests and actions. A notable example of pipeline politics occurred at the beginning of the year 2009, wherein a dispute between Russia and Ukraine seemingly over pricing of sold natural gas led to a major political crisis. Russian state-owned gas company Gazprom cut off natural gas supplies to Ukraine after talks between it and the Ukrainian government failed. In addition to cutting off supplies to Ukraine, Russian gas flowing through Ukraine, which included nearly all supplies to Southeastern Europe and some supplies to Central and Western Europe, was cut off, creating a major crisis in several countries heavily dependent on Russian gas as fuel. To avoid another crisis, two new pipelines, “Nord stream” and “South stream”, were constructed to connect Russia with central and south Europe without passing by Ukraine.

Oil pipelines are made from steel or plastic tubes with inner diameter typically from 4 to 48 inches (100 to 1,220 mm). Most pipelines are typically buried at a depth of about 3 to 6 feet (0.91 to 1.83 m). The oil is kept in motion by pump stations along the pipeline, and usually flows at speed of about 1 to 6 meters per second (3.3 to 19.7 ft. /s). Pipelines could also be used as multi-product pipelines; they are used to transport two or more different products in sequence in the same pipeline.

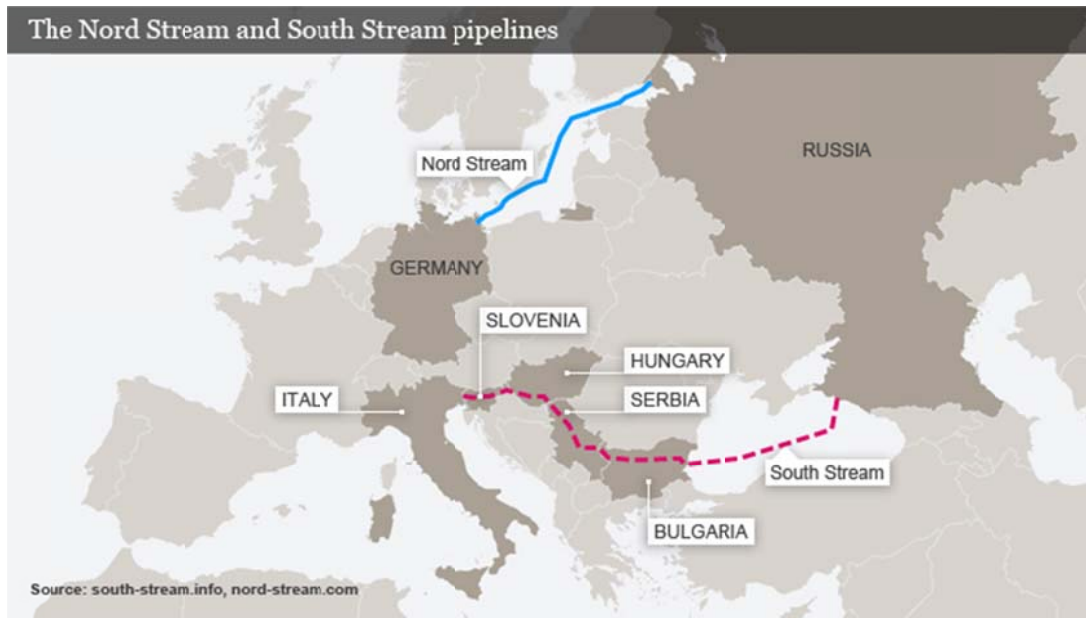


Figure 0-1 : Nord Stream and South Stream Pipelines (Insa Wrede, 2014).

Pipeline networks are composed, other than the pipe itself, of several pieces of equipment that operate together to move products from location to location. The main elements of a pipeline system are:

Initial injection station

It is also known as supply or inlet station, is the beginning of the system, where the product is injected into the line. Storage facilities, pumps or compressors are usually located at these locations.

Compressor/pump stations

Pumps for liquid pipelines and Compressors for gas pipelines are located along the line to move the product through the pipeline. The location of these stations is defined by the topography of the terrain, the type of product being transported, or operational conditions of the network.

Partial delivery station

It is also known as intermediate stations; these facilities allow the pipeline operator to deliver part of the product being transported.

Block valve station

These are the first line of protection for pipelines. With these valves the operator can isolate any segment of the line for maintenance work or isolate a rupture or leak. Block valve stations are usually located every 20 to 30 miles (48 km), depending on the type of pipeline. Even though it is not a design rule, it is a very usual practice in liquid pipelines. The location of

these stations depends exclusively on the nature of the product being transported, the trajectory of the pipeline and/or the operational conditions of the line.

Final delivery station

It is also known as outlet stations or terminals, this is where the product will be distributed to the consumer. It could be a tank terminal for liquid pipelines or a connection to a distribution network for gas pipelines.

The market size for oil and gas pipeline construction experienced tremendous growth. For 2013 only, operators planned to build more than 15,300 miles of oil and gas pipelines worldwide at a cost of more than \$50 billion. For 2012 only, companies had planned nearly 8,900 miles at a cost of more than \$39.6 billion (Smith, 2013). For projects completed after 2013 (Table 2), companies plan to lay more than 44,800 miles of line and spend roughly \$144 billion. It is estimated that \$193 billion will be spent on onshore pipeline projects worldwide through 2015, according to data in energy business analysts Douglas-Westwood's third edition of The World Onshore Pipelines Report 2011-2015. (Worldwide Onshore Pipeline Construction Market Appears Strong through 2015, 2010)

Area	4-10 in.	12-20 in.	22-30 in.	32+ in.	Total
GAS PIPELINES					
US	27	961	835	393	2,216
Canada	—	149	48	76	273
Latin America	—	155	—	—	155
Asia-Pacific ²	—	20	539	1,553	2,512
Europe ³	—	50	230	132	412
Middle East	—	34	227	308	569
Africa	26	—	—	—	26
Total gas	53	1,369	2,279	2,462	6,163
CRUDE PIPELINES					
US	—	164	516	485	1,165
Canada	—	152	129	—	281
Latin America	—	129	—	—	129
Asia-Pacific ²	—	—	570	1,553	2,123
Europe ³	—	20	—	175	195
Middle East	135	255	—	—	390
Africa	—	—	186	—	186
Total crude	135	720	1,401	2,213	4,469
PRODUCT PIPELINES					
US	—	2,426	530	—	2,956
Canada	—	719	—	—	719
Latin America	19	156	606	—	781
Asia-Pacific ²	—	270	—	—	270
Europe ³	—	—	—	—	0
Middle East	—	—	—	—	0
Africa	—	—	—	—	0
Total product	19	3,571	1,136	0	4,726
WORLD TOTAL					
Gas	53	1,369	2,279	2,462	6,163
Crude	135	720	1,401	2,213	4,469
Product	19	3,571	1,136	—	4,726
Total	207	5,660	4,816	4,675	15,358

¹Projects planned to be completed in 2013. ²Regions east of the Ural Mountains and south of the Caucasus Mountains, excluding the Middle East. ³Regions west of the Ural Mountains and north of the Caucasus Mountains.

Figure 0-2: Pipeline Construction Quantities in 2013 (Smith, 2013).

1.2. Pipeline Construction scheduling

Due to the economical and sometimes political significance of pipelines projects, they are usually constructed under a very tight schedule. They require working under difficult conditions in remote locations such as mountains, jungles, deserts, and even the Arctic Circle which causes logistical constraints of deploying equipment, materials and labor. They

sometimes require working within different countries which means different labor and equipment conditions. They involve various types of labor and equipment that work simultaneously or consecutively along the pipeline path. Avoiding delays during the construction phase of pipeline projects can yield significant benefits to owners, pipeline contractors, and the public. Delays in completing pipeline construction projects not only result in higher costs to owners and contractors, but also add to the cost passed down to the end users. Although some of the variables causing delay are difficult to control, good planning and scheduling of pipeline construction projects can reduce the time and cost of construction.

Pipeline construction falls under the category of repetitive construction. Other examples of repetitive construction can be highways, multiple housing projects and dike construction projects. In repetitive construction, the same activities are repeated for a similar number of units at different locations. Pipeline path is divided into number of segments or stations which are covered by number of construction base camps. Accordingly, each station can be considered a separate unit for which all activities of the project are performed.

Pipeline construction projects require resources to perform the same work in the various stations by moving from one station to the next in the project. Because of this frequent resource movement, an effective schedule is important to ensure the uninterrupted usage of resources of repetitive activities between stations. Consequently, the waste from resource waiting for preceding resources to finish their work should be eliminated to maintain continuity of work. Maintaining work continuity leads to maximizing the learning curve effect and minimizing the idle time of each resource.

Various scheduling techniques have been used for repetitive construction projects. However, they have all proven not to be capable of providing all the benefits simultaneously. Each technique had to overlook some aspects to reach an optimal schedule. Further discussion of the disadvantages of using these techniques will be fully shown in Chapter Two.

1.3. Considerations in the Planning and Control of Pipeline Construction

There are various challenges that should be considered in the development of an effective model for the planning and control of pipeline construction. During the construction phase of a project, job superintendents place resource utilization as a priority, thus creating resource driven schedules will help reflect the actual construction process.

The first challenge is to develop a resource-driven scheduling model that incorporates utilization of different kinds of resources. Due to The linear shape of the construction site that spread over hundreds of kilometers, all the resource work together in each station from one end of the segment, served by the construction camp, to the other. If any resource is detained in one station for any reason, it shall hold the rest of resources used in the following activities

and leave them idle. In addition, several types of resources utilized in pipeline construction are expensive and sophisticated that only one crew would be available for the whole segment. Therefore, the utilization of each crew of every resource separately among the various stations would help in obtaining the optimum schedule.

The second challenge is to develop a model that deals with the geographic nature of the pipeline construction site. Pipelines are usually located in remote areas that are not served with any type of infrastructure. There are a number of base camps that contain labor housing, technical offices, material storage and equipment's shelters and workshops. The location and number of base camps that serve the project is decided based on the nearby populated areas. An optimum schedule would take into account the travel time of the resources to the site and back to the camp based on its location relative to the different stations.

The third challenge is considering realistic activity durations in the model. As the stations of the pipeline varies in their conditions such as the soil type and the topographic nature, the productivity of various resources and quantity of needed work are affected. As a result, the activities duration becomes function not only of resources productivity but also the conditions of each station.

1.4. Problem Statement

Scheduling of repetitive construction projects is done by several techniques. These techniques shall be shown in the literature review in the next chapter. These techniques have limitations in accomplishing some or all of the following requirements:

- Finding the optimal number of units that should be utilized simultaneously within an activity
- Visualizing the entire project
- Ensuring work continuity.
- Satisfying resource constraints
- Dealing with probabilistic durations.
- Maintaining logical interconnections between various activities
- Providing assistance for planners and the management in justifying their decisions.
- Answering "what-if" questions.

1.5. Research Objectives

The main objective of this research is to study planning and scheduling of pipeline projects and develop a model for scheduling and control of pipeline projects that addresses the challenges outlined in the earlier section. In order to develop this model, the objectives of this study are:

- 1) To study related literature focusing on scheduling of repetitive activities and determine the problems that faced researchers in this field.
- 2) To develop simple models based on the literature to solve the scheduling problem addressed previously and illustrate the limitations of these models
- 3) To develop a simulation model that represents the process of pipeline construction and considers the special characteristics and unique features of pipeline projects.
- 4) To develop an optimization module that produces a resource-driven schedule
- 5) To implement the model on a real case study and study the difference between the real and proposed schedule.

1.6. Thesis Organization

Chapter 2 presents a literature review of available scheduling techniques for repetitive construction projects.

Chapter 3 explains the stages of pipeline construction projects. The sequence of activities is explained showing the utilization of the different kinds of resources in each activity.

Chapter 4 presents the stages of creating the deterministic models found in the literature that could be used for repetitive construction projects and the limitations of their use.

Chapter 5 presents the stages of constructing the simulation model which would represent the different stages of pipeline construction

Chapter 6 presents the stages of defining the optimization module which would depend on the simulation model to come out with the optimum schedule

Chapter 7 presents the results of the case study, summarizes the results of this research, highlights its contributions and advises the recommendations for future research.

Chapter 2

Literature Review

2.1. Introduction

The main purpose of Construction Management is to deliver a project on time, within a certain budget and in accordance to pre-defined quality standards. Time, cost and quality create a triangle, which is called the fundamental triangle of project management. The planning of a project is carried out in a manner to accommodate these criteria.

The creation of a realistic schedule also serves purposes other than the one stated above, in fact its use is not just limited to the construction stage, and it is extended to the pre-construction and post-construction stages as well. The schedule provides the necessary insight for the project manager or his/her representative to identify the required resources and plan for their timely allocation ahead of time. Cash flows, the assignment of work crews, delivery of material and equipment allocation are such considerations. Schedules are also appropriate tools for project control. In the post construction stage, project schedules serve as a reference to facilitate construction claims and disputes.

Different types of construction projects are planned and scheduled according to their characteristics, in order to achieve an optimum schedule in respect to *the fundamental triangle of construction management*. Among the available categories existing in construction, pipelines construction fall into the category of repetitive construction projects. Repetitive construction projects are made up of a number of similar or identical units (El Rayes, 1997). Examples of repetitive construction could be high-rise buildings, housing projects, highways, airport runways, railways, bridges, tunnels, wind energy farms, water pipes and civil infrastructure. Repetitive projects may be divided into two categories: (1) projects that are repetitive due to a uniform repetition of a unit work throughout projects such as multiple similar houses and high rise building; (2) projects that are repetitive due to their geometrical layout such as highways, tunnels and pipelines.

This chapter presents a review of recent literature in traditional scheduling techniques for construction projects in general, and special scheduling techniques for repetitive construction projects in particular.

2.2. Traditional Scheduling Techniques

2.2.1. Bar Chart Method

Bar Chart method utilizes graphical approach to represent the project schedule by plotting the activities against time. It was invented and developed by Henry L. Gantt during World War I.

The duration of each activity is represented by the length of the bar in accordance with the time scale of the chart. Bar charts are still popular and are used in construction till now. Due to its graphical nature, it is easily understood by all levels of management and supervision, thus becoming an effective means of communication between engineers and foremen. It is also used as a tool to identify the required resources. Resource allocation and leveling is often done using Bar charts. The major deficiency of this method is that it cannot illustrate the interrelationships between activities, thereby failing to identify the critical activities, which actually control the project duration (Chazanowski and Johnston, 1986; Stradal and Cacha, 1982).

2.2.2. Network Techniques

Network techniques were the next step after bar charts. Network diagrams had the ability to graphically represent the activities and their relationships. By displaying the relationships between activities, these diagrams effectively eliminated the main disadvantage of bar charts. This way the Network techniques enabled the identification of critical activities that control the project duration. They are either deterministic or probabilistic. (Chazanowski and Johnston, 1986)

The Arrow Diagram Method (ADM) and the Precedence Diagram Method (PDM) are the two common deterministic network techniques available. These methods are also known as the Critical Path Method (CPM). In ADM, activities are represented by arrows and nodes connecting these arrows are considered events or milestones. In PDM, nodes represent activities and connecting arrows represent the interrelationship among these activities. PDM has a number of advantages over ADM; there is no need for 'dummy activities' in PDM, and ADM can consider only one type of relationship namely finish to start whereas PDM can consider four different types of relationships namely, Finish to Start, Finish to Finish, Start to Start and Start to Finish with lag and lead times.

As for probabilistic network scheduling techniques, one method, Program Evaluation and Review Technique (PERT), considers three different durations for each activity, the most optimistic, the most likely and the most pessimistic durations. This characteristic helps the scheduler in modeling the uncertainty associated with the duration of each activity. Besides the fact that PERT has the same limitations of deterministic network methods, its use is limited due to the assumptions it is based on.

As for repetitive construction, there are limitations for the network scheduling techniques. The main concern is that these techniques do not consider effective resource utilization and for this reason it is widely criticized in literature (Birrell, 1980; Kavanagh, 1985). Network techniques emphasize on minimizing the total project duration and thus make the fundamental, unrealistic assumption that resources are unlimited and centrally controlled. Top management can relate to such goal. On the other hand, site superintendents focus on

minimizing the resource input and maximizing resource utilization rather than critical paths or early project completion (Birrell, 1980; Kavanagh, 1985). That's why they are reluctant to use it in spite of management encouragement (Tavakoli & Riachi, 1990).

Moreover, these techniques produce large and complex schedules when applied to repetitive activities and the complexity increases with the increase in repetitions (Carr and Meyer, 1974). Hence it becomes practically inapplicable for projects that comprise a large number of repetitive activities such as a housing development project with 100 houses. For example, if a housing project with 50 typical houses is to be considered and if the work of each house can be broken down into only 20 activities, the project network would consist of 1000 activities, which complicates the understanding of the schedule and control process.

Another important shortcoming of traditional techniques is its inability to maintain crew work continuity. Its application during scheduling is to schedule work in repetitive units in an order that enables well-timed movement of crews from one unit to the next, avoiding crew idle time. This is known as the 'crew work continuity constraint'. Crew work continuity makes maximum use of the learning curve effect for each crew, maintains a constant workforce by reducing the number of hires and fires, minimizes the crew and equipment idle time, retains skilled labor and last but not least it has proven to be an effective resource utilization strategy for repetitive construction (Birrell, 1980; ElRayes and Moselhi, 1993 (a)).

In addition to all of the mentioned shortcomings of network scheduling, there are hidden interrelationships between activities due to resource constraints which are not shown in the actual network. The methods used to calculate the float of activities cannot depict this constraint, therefore there is actually a 'Phantom Float' which alters the network calculations and perhaps even the total project time (Kim and de la Garza, 2003).

They also complicate the implementation of multiple-crew strategies. They cannot provide data for the progress of individual crews alongside the progress of the project itself.

2.3. Techniques for Scheduling Repetitive Activities

Due to the limitations of the network techniques mentioned in the earlier section, a number of techniques were proposed in the literature for scheduling. Repetitive activities generally can be divided into two categories: 'typical' and 'non-typical' or 'atypical' activities. In the typical repetitive category, common activities in all repetitive units are assumed to have identical durations, such as the paving activity. In the non-typical category, activities need not have identical durations, as in the earth moving activity.

Methods of scheduling projects with repetitive activities can be grouped into two main categories. The first category comprises of methods, which were developed to schedule typical repetitive activities only. These methods are often referred to as 'Line Of Balance'

(LOB) (Al Sarraj, 1990; Carr and Meyer, 1974). The second category includes methods which were developed to schedule both typical and non-typical repetitive activities, and are often referred to as 'Linear Scheduling Method' (LSM) (Russell and Caselton, 1988; Chrazanwski and Johnston, 1986).

There are other techniques proposed in the literature for scheduling repetitive activities utilizing the principles of either LOB or LSM, with the main objective of maintaining crew work continuity. These techniques include 'Vertical production method' (VPM) (O'Brien, 1975; O'Brien et al. 1985), Time-Space Scheduling (Stradal and Cacha, 1982), 'Disturbance Scheduling' (Whiteman and Irwing, 1988), 'Horizontal and Vertical Logic Scheduling', (Thabet and Beliveau, 1994), 'Velocity Diagrams' (Dressler, 1980), Simulation of Repetitive Networks (SIREN) (Kavanagh, 1985), Repetitive Project Modelling (RPM) (Reda, 1990).

2.3.1. Line of Balance (LOB)

Line of Balance (LOB) method was developed by the U.S Navy in 1942 for planning and control of repetitive projects. The method was primarily designed for industrial manufacturing operations. It was used by the industrial engineers to optimize the cost of output by determining the required resources and setting the speed of each stage. However, in industrial manufacturing, products move along a production line. On the other hand, in construction of repetitive projects the products are stationary and machines move along a line. Due to this difference, LOB method was modified in 1966 from its original manufacturing industry purpose to enable its application to housing. The developed method was simple and the schedule could be represented by plotting the number of units in Y-axis and the duration in X-axis. Repetitive activities are represented by separate inclined bars. (Al Sarraj, 1990)

There are several methods proposed in the literature having the same name 'Line of Balance' (LOB) and sharing the same concept (Al Sarraj, 1990; Arditi and Albulak, 1986; Ammar, 2013). Arditi and Albulak (1986) used LOB to schedule a highway project. They concluded that LOB schedule is easy to understand and requires less time and effort. Al Sarraj (1990) developed a formal algorithm for LOB to facilitate scheduling, resource management and project analysis and control in order to provide a mathematical alternative for the graphical LOB method. Ammar (2013) introduced a method that integrates CPM with LOB to make use of the analytical capabilities of CPM in addition to LOB's capabilities in resource utilization

LOB method has been found to have apparent advantages such as maintaining crew work continuity, generating resource driven schedules, incorporating multiple crews and providing clear and easy way to produce schedules (Arditi and Albulak, 1986). However, it has been criticized in the literature for a number of reasons.

Kavanagh (1985) indicated that LOB method was designed to model simple repetitive production process and is not suitable for complex construction projects. Arditi and Albulak (1986) commented about the visual problems associated with the graphical LOB diagram and suggested that different colors can be used to distinguish overlapping activities. They also stated that the schedule is very sensitive to the estimations of activities' man hour requirements and needed crew sizes. Any error in these estimations would be magnified due to repetition. Neale and Raju (1988) stated that the calculations needed in LOB are tedious and requires a lot of trials in order to make the pace of work similar for all activities. Thus, they introduced a way to refine LOB method using a spreadsheet format but they faced complex relationships and concluded that it was practically infeasible to draw the schedule in the form of a diagram. Another major disadvantage of LOB method is its inability to schedule non-typical repetitive activities as well as any non-repetitive activities that occur within the project (Moselhi and El-Rayes, 1993(a) (b)).

2.3.2. Linear Scheduling Method (LSM)

Linear Scheduling Method (LSM) was developed to overcome the limitations of LOB method. LSM is capable of scheduling typical and non-typical repetitive activities along with all the apparent advantages of LOB method (Russell and Caselton, 1988; Chrzanowski and Johnston, 1986; Moselhi and El Rayes, 1993(a)). An important difference between LOB method and LSM is the graphical presentation of the schedule. In LOB method, an activity is represented by two parallel lines with a constant slope, whereas in LSM it is represented by a single line with varying slope.

Johnston (1981) described the basic presentation format of LSM as having two axes. The horizontal axis represents the project duration and the vertical axis represents the number of repetitive units while separate diagonal lines represent repetitive activities. He suggested that LSM schedule is simple and can convey detail work schedule. Chrzanowski and Johnston (1986) employed CPM technique with LSM to schedule a highway project in order to evaluate the capabilities of LSM. They concluded that LSM has several advantages such as its simplicity that helps personnel to understand with minimum training, the ability to extract various types of information such as job progress and resource allocations and the ability to take quick decisions in resource utilization matters. On the other hand, LSM cannot be used for non-repetitive activities and it is a graphical method that cannot utilize numerical computations.

2.4. Optimized Scheduling

As mentioned before, in the practice of professional construction management, time, cost and quality are of essence. For repetitive projects such as pipeline, roads, high rise buildings or housing projects, corporations usually invest a large capital and need the project up-and-running as fast as possible. Thus, minimizing total construction costs along with the duration

is of utmost importance when scheduling construction projects with repetitive activities. In the literature, Attempts made to optimize LSM using mathematically based models can be categorized as follows: 1) operations research models; 2) simulation models; and 3) artificial intelligence (AI) models (Hassanein, 2002).

2.4.1. Operations Research Models

In recent attempts made to optimize repetitive construction, operation research models have proven to be the tool of choice among the researchers. Either linear programming or dynamic programming was employed in these models.

Reda (1990) developed a model called Repetitive Project Model (RPM) to minimize project direct costs. It combined a linear programming formulation with network technique to present a typical stage of the project and a graphical technique to represent the results. This formulation had a number of limitations. One of which is that the productivity rates are constant for all stages of project which limits its application to typical repetitive projects. Another was that the possibility of work interruptions was ignored.

Selinger (1980) was the first to develop a dynamic programming formulation solution to optimize linear schedules. The formulation managed to maintain crew work continuity however it did not consider cost.

Handa and Barcia (1986) presented a model that relied on Optimal Control Theory. The model could take account for variable production rates. The work continuity constraint was maintained but not enforced. Moreover, the model was incapable of considering multiple crews for activities.

Russell and Caselton (1988) built on the works of Selinger (1980) and developed a two-variable N-stage dynamic programming solution that can find the minimum project duration. In order to achieve this, the set of possible interruption vectors were defined for each activity as the second variable where the first is the set of possible durations for the activity. The possibility of work interruption contradicts with the work-continuity constraint; however, it achieves the objective of schedule optimization in respect to time. The limitations of this model are that it does not consider cost like Selinger (1980) and it is incapable of considering multiple predecessors and/or successors.

Moselhi and El Rayes (1993 (a) & (b)) proposed a dynamic programming model that overcame the limitations of previous models of Selinger (1980) and Russell and Caselton (1988). Their model was an object oriented optimization model that used a two-variable N-stage dynamic programming formulation to consider overall project cost as a priority as well as the learning curve effect and the impact of weather on crews' productivity. The model's optimization procedure was executed in two stages, forward and backward paths, and

enforced work continuity. It offered assistance to the user to select an optimum crew formation from a set of possible alternatives.

Eldin and Senouci (1994) also used a two-variable N-stage dynamic programming formulation to minimize total project cost. The two variables represented possible activity resources and acceptable interruptions at each stage. The model, however, could only consider one crew per activity.

El Rayes and Moselhi (1998) developed an algorithm that considers precedence relationships, crew availability and crew work continuity constraints. In addition, it considers the impact of the following practical factors: (i) type of repetitive activity (i.e. typical or atypical); (ii) multiple crews assigned to work simultaneously on an activity; (iii) crew availability period on site; (iv) activity interruption; and (v) order of executing repetitive units. The model has the ability to generate interruption vectors that would minimize total construction cost by itself, unlike the model presented by Eldin and Senouci (1994) where it was necessary to input predefined interruptions.

Moselhi and Hassanein (2003) developed a model that employs a two-variable, N-stage, dynamic programming formulation coupled with a set of heuristic rules. It had the ability to optimize either project duration, total cost or their combined effect (A+B bidding). The model supported multiple crews to work simultaneously on any activity while accounting for: 1) accounts for the presence of transverse obstructions, such as rivers and creeks; 2) utilizes resource-driven scheduling; 3) incorporates repetitive and non-repetitive activities in the optimization procedure; 4) enables the consideration of multiple predecessors and successors for each activity; and 5) accounts for variations in quantity of work and unit length of repetitive activities

2.4.2. Simulation Models

Several simulation models have been developed to introduce computer simulation modeling to scheduling of repetitive projects. Computer simulation models are utilized to change some of the deterministic input elements in the construction process and estimate the consequence.

Ashley (1980) proposed a simulation model for scheduling of repetitive projects that adopts a queuing model to resolve the crew availability problem. The model is implemented using GPSS simulation language, and is based on the concept that repetitive units are organized in a queue to be served by the assigned crew. A main limitation in this model that it doesn't recognize any priority for an activities or units

Kavanagh (1985) presented SIREN (Simulation of REpetitive Networks), a repetitive construction model coded in the GPSS language. The model would first carry out a deterministic analysis and then it employs Monte-Carlo simulation to account for the probability distributions for values of activity durations and weather conditions. The presented

a priority system for assigning crews that is close to a superintendent's priorities. However, this model had some limitations. First, one activity cannot utilize more than one type of crew. Second, it doesn't allow the user to enforce his plan of work and priorities for activities or stages. Finally, the model presumes that the repetitive units are essentially independent.

Pena-Mora et al. (2008) developed a Discrete Event & System Dynamics hybrid simulation model to simulate the combined effect of operational and strategic management decisions on infrastructure projects performance. The model was coded using Extend simulation environment. They concluded that simulation models are useful means for construction managers to consider the impact of their decisions without facing costly consequences.

Hajdasz (2014) utilized MoCCAS (MONolithic Construction Computer Aided System), a comprehensive decision support tool for flexible construction site management in repetitive projects. MoCCAS supports the construction site manager in developing optimal execution scenarios by providing different construction strategies.

Moradi et al. (2015) proposed another hybrid simulation model that uses both Discrete Event & System Dynamics to simulate repetitive construction projects. The model was developed using AnyLogic software. The model used concreting projects as an example to test the performance of the proposed model. They concluded that using hybrid model that employs both Discrete Event & System Dynamics is better than using each of them individually.

2.5. Summary and Conclusion

This chapter presented a review of recent literature on scheduling of construction projects with repetitive activities. Traditional scheduling techniques and their shortcomings in respect to repetitive construction were also discussed. The emergence of linear scheduling methods, the pros and cons of each of the developed techniques, their considerations and limitations were also reviewed. All the proposed models disregarded the linear nature of many repetitive projects such as pipelines, roads and railway projects; a main factor which would highly affect the crew movement and its continuity of work. Another is that these models didn't consider the possibility to change the sequence of work in the project units from one activity to the other. These findings have been effectively used in the development of the proposed model for scheduling, tracking and control of pipeline projects, which is described in the following Chapters.

Chapter 3

Stages of Pipeline Construction

3.1 Introduction

This chapter presents the stages of pipeline construction illustrating the sequence of activities and the resources employed in each of them. Pipeline construction is one of the complex construction projects which employ a large number of specialized crews as well as many types of heavy machinery. Such projects need an accurate schedule to maintain the crew work continuity and minimize idle times.

3.2 Stages of Pipeline Construction

A pipeline can be broken down into three basic elements where different forms of pipeline construction method are used. They are:

- (i) Open cross-country areas, where the spread technique is used
- (ii) Crossings, where specialist crews and civil engineering techniques are used
- (iii) Special sections such as built up urban areas, restricted working areas, difficult terrain sections and environmentally sensitive areas.

The basic method of constructing steel, welded oil and gas onshore pipelines in open cross country areas is generally known as the “spread technique”. The spread technique utilizes the principles of the production line system, but in the case of a pipeline the product (the pipeline) is static and the individual work force, (crews) move along the pipeline track. The implementation of the spread technique is conditional on the pipeline being welded above ground in maximum possible continuous lengths between obstructions/crossings, which can extend to lengths in excess of 10 kilometers. These welded pipe lengths are then immediately installed into unsupported/unobstructed trenches gradually in one continuous length utilizing multiple (three or more) mobile lifting tractors (side-booms) together. The breaks in the continuous main spread method of working result from the location of existing services, roads, railways, tracks, ditches, streams and river crossings, and are also dependent upon restricted working, time constraints and physical features/obstructions. These breaks in the main pipeline spread activities are undertaken by dedicated specialist crews utilizing a variety of special construction techniques and are generally undertaken after the main pipeline sections have been installed.

The main pipeline spread installation is undertaken by dedicated crews undertaking one operation at a time commencing at one end of the pipeline and travelling forward to the other end at anything from 500m to 1,500m per day depending on the diameter of the pipe, terrain,

soils, etc. The program of activities and the start-up of the crews is dependent on available resources and the risk of one crew having an impact upon the following activities.

Pre-construction activities need to be carried out by the Installation Contractor prior to the start of the main pipeline installation activities. These activities include finalizing the pipeline route, detailed design finalization, mobilization, notification of entry to landowners, setting-up of pipe yards and base camps, establishing temporary works requirements, setting-up of geographic positioning stations, design of land drainage in agricultural areas and reinstatement works, construction of temporary access roads, pre-environmental mitigation works, and agreeing with landowners any special requirements prior to entry onto their properties. The Installation Contractor will carry out pre-entry surveys as-and-where required so as to record the condition of the land prior to the start of any work.

Once the pre-construction activities have been completed, then the main construction works can commence. Generally, operations are carried out in three main activities groups as shown in figure (3-1):

1. Preparing Work Area
2. Layout Pipe and Weld above Ground
3. Excavate Trench and Installation of Pipe

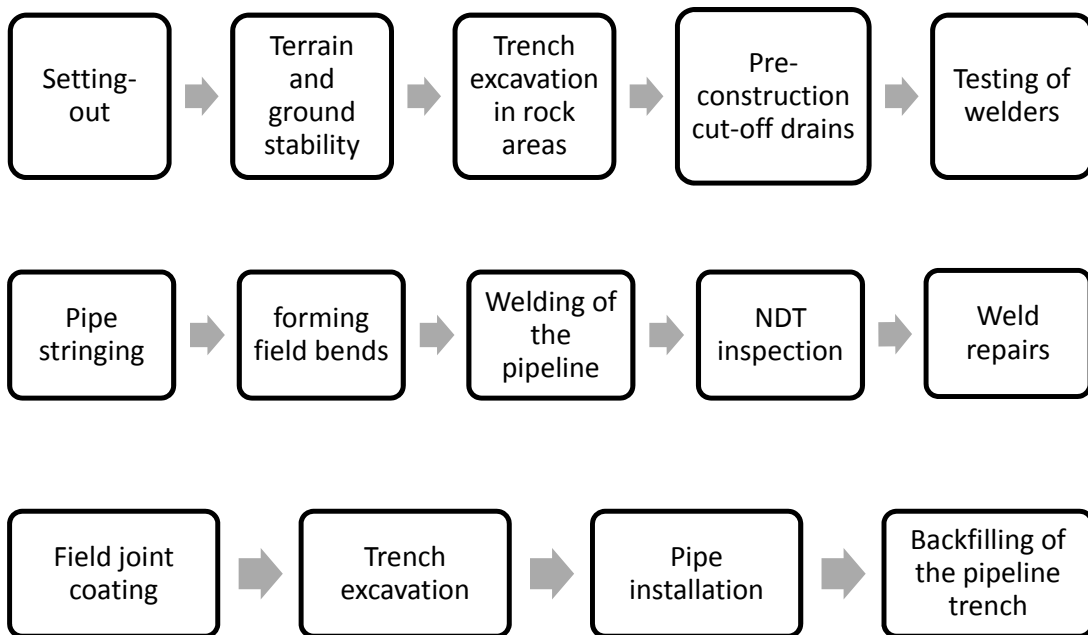


Figure 3-1: Flowchart of Spread Method Activities

The first group which is Preparing Work Area contains many operations, First, Setting-out. The setting-out crews are the first personnel from the construction contractor's workforce to enter the site to commence the main construction activities. The setting out of the works should be scheduled to commence at least four weeks prior to the remainder of the first group activities. This work will be carried out with small four man crews using GPS and surveying instruments. Setting-out pegs will be placed at all boundaries, changes in direction and intermediate sightings on the proposed centre line and the extremities of the working easement. In areas of open country where good and level access is available along the pipeline route and it is anticipated the rock or ground is of sufficient strength that it could impede progress of the trench excavation, then initial ground investigations works will be carried out directly behind the setting-out crew. Part of the setting-out crew's duties is to identify any existing services that cross or are in close proximity to the pipeline and supervise the trial hole crew. The trial hole crew will hand excavate to expose, identify and determine the exact location of all existing services. This data will be recorded and transferred to the engineers for incorporation into the final pipeline design.

The second activity is Pre-construction terrain and ground stability. At locations where there is a risk of ground movement that could result in safety risks to the construction activities and/or undermine the pipe during installation and the period prior to final reinstatement then permanent stability of the affected terrain needs to be undertaken. This work can be separated into two elements; first, Removal of material such as the overburden at the top of ravines and the removal of loose material that could move during the installation works and second, Addition of material such as Bentonite, which is injected under pressure into gravels with high and fast water tables and deep mining areas to provide a protective curtain around the pipe. It also includes the adding (placement) of boulders/ground at the toe of steep gradients on forward and side slopes in the second element.

The third activity is Trench excavation in rock areas. In areas where rock is confirmed as such by the initial ground investigation works then the trench is excavated ahead of any pipe operations. This sequence of working is undertaken to ensure that the excavation of the trench cannot cause any damage to the pipe and/or pipe coating and provide an extended safe working width for the excavation crews allowing double –sided trench working by excavators/ breakers.

Following the review of the data from the initial ripper and trial hole surveys, the ground will be classified in ease of excavation into five groups defined by the method of removal. These are (i) utilizing standard excavation, (ii) larger more powerful excavators (face shovels converted to back-actors), (iii) ripping/hydraulic hammer and excavation, (iv) blasting/hydraulic hammer and excavation and (v) rock trenchers (saw and blade). The finished trench should be to the correct depth and width to suite the pipe diameter, plus any

bedding and pipe cover. The trench should also be in a straight line so that the pipe can lay central in the trench without coming into contact with the trench sides. All loose and jagged outcrops, which could come in contact with the pipe during lay operations, will be removed. The excavation will commence with dedicated crews immediately following the ROW operation. The forward progress will be dependent upon the ground strength, grain structure, terrain, access, method of removal and number of crews/equipment employed.

The fourth and last activity is Pre-construction cut-off drains. All cut-off drainage works, which comprise the connection of existing drains to a new header pipe, will commence immediately after the right of way and fencing operations. Cut-off drainage works will be undertaken at locations where there are existing concentrated drainage schemes on agricultural land and where agreement is reached with the landowners and/or occupiers to their installation. This work will be resourced taking account of the scope of work and the requirement to achieve pipeline installation progress of, say, 500 to 1,500 meters per day along the pipeline route.

The second group of activities is layout pipe and weld above ground. The first activity is Project mechanical procedures/testing of welders. prior to the start of any mechanical works the Contractor will issue for Client approval a full set of mechanical procedures for bending, welding, x-ray and coating. These procedures will address how the Contractor intends to undertake the work in accordance with the project specifications detailing equipment and specific mandatory requirements. The procedures, particularly with regard to welding and x-ray will be sufficient to cover the full ranges of the various parameters characteristic of the project in terms of diameter, wall thickness and technique. Once the documented procedures are approved then full trials for each element of the works will be carried out, fully inspected and witnessed by the Client. The welding will include non-and full destructive testing to ensure that the procedure welds are undertaken in strict compliance with the contract requirements and fully comply with the minimum strength, hardness and quality requirements of the relevant specifications. Once the procedures have been approved then the welders will be tested to ensure that they can comply with the requirements of the procedure welds. A register will be maintained of the welders employed on the project with the various welding techniques they are approved to work on.

The second activity is Pipe stringing. The pipes and pre-formed bends will be scheduled to be delivered to, and stock piled at, the proposed pipeline pipe yards some 4 to 8 weeks in advance of stringing operations. The pipe supply should ensure that the various grades, wall thicknesses and coatings are supplied in sufficient and correct quantities to meet the program. Immediately following ROW or topsoil strip or excavation in rock areas, the pipe stringing operations will commence, which involves laying the pipe lengths along the easement length using pipe trailers. A typical crew will consist of two cranes - one at the base camp loading the pipe trailers and the other on the pipeline easement off-loading the pipe

trailers. In the event that ground conditions do not permit travel down the easement with standard or special heavy-duty pipe trailers then the pipes will be loaded on to tracked pipe carriers at the public roads or at a point where the change in ground conditions occurs and permits the turning of the wheeled pipe trailers. See figure (3-2)



Figure 0-2 : Pipe Stringing Activity

The third activity is forming field bends (cold bending). Once the pipe has been strung along the easement, engineers will follow to determine the location of all bends required in order that the pipeline can follow the contours of the land and the required line and level as detailed on the drawings. There are two types of bends normally used i.e. hot pre-formed or forged bends which are manufactured off site in a factory and are to a radius of 5 or 3 times the pipe diameter and cold bends which are to a radius of 40 times the pipe diameter and are formed in the field. A typical cold bending crew consists of a four-man team together with a bending machine and a side boom tractor. The bending machine is towed along the pipeline route by the side boom and includes “formers” consisting of 20 – 150 ton hydraulic rams, which bend the pipe to the required radius and angle. The side boom acts as a lifting device and has a fixed jib attached to a tracked dozer with a capability of lifting between 15 to 120 tons, dependent upon the size of the machine used. The number of cold bends required depends on the route and contours of the pipeline. Typically, they can range from 1 pipe in 10 in developed regions to 1 pipe in 50 in open country. The cold bend angle that can be achieved ranges from maximum angles of 12 degrees (42” pipe) to 40 degrees (12” pipe).

The fourth activity is Welding of the pipeline. The welding of the pipeline will commence a few days after the cold bending crew. The welding crew will weld the pipeline in continuous lengths between features such as roads, watercourses, tracks, railways, services and other underground obstacles that prevent the pipeline being continuously installed in the trench. There are primarily two methods of welding which are manual or automatic. As the names

imply manual welding involves the welding of the pipe by welders and automatic involves a semi-automatic system. Both systems generally (although certain automatic systems can now do single pass complete welds) operate on a front-end/back-end principle. The front-end consists in a manual operation with, say, 3 separate welding stations placed on CAT D6 carriage consisting of a HIAB for the welding shelter (used in inclement weather or windy conditions), 4 welding bullets and a compressor. The welding stations work on 3 separate joints and complete one pass before moving on with the sequence being the bead (2 - 4 welders), immediately followed by the hot pass (2 – 3 welders) and then hot fill (2 welders). With the automatic process, 1 machine deposits sufficient weld metal equivalent to the 3 manual passes. The weld is allowed to cool after the front-end passes and then sufficient welders working in pairs or multiple automatic machines follow on to fill and cap that day's production. The crew will achieve progress in the order of one weld approximately every 3 to 5 minutes or up to 90 to 150 welds per day, which is equivalent to 1,000 to 1,500 meters of pipeline on 12 meter pipes and up to twice that if double -jointed pipes are used. See figure (3-3).



Figure 0-3 : Pipe Welding Activity

The fifth activity is Non Destructive Tests (NDT) inspection. All welds on the pipeline are generally subjected to inspection by radiography. This is achieved on the main pipeline by an internal x-ray tube travelling along the inside of the pipe carrying out x-rays at each weld for approximately 2 minutes per weld. On completion of the x-ray the film is taken to a dark room and processed in time for the results to be available for inspection at the end of the day or early the next day. Welds, which do not meet the required acceptance criteria, are either repaired or cut out and re-welded. Experienced and qualified x-ray specialists undertake the radiography under controlled conditions. Before the operation is started, the section of pipeline is cordoned off by marker tape to stop entry by non x-ray personnel and audio/flashing warning alarms are activated during all times when the x-ray tube is energized. The x-ray personnel are on constant surveillance to ensure that the workforce and members of the public are aware of the x-ray activities and only authorized access is permitted. Welds

completed by semi-automatic welding processes are examined using automatic ultrasonic testing (AUT) techniques. This consists of an assembly that traverses the circumference of each completed weld in order to detect any defects. The results of each ultrasonically inspected weld are automatically recorded and are used to determine whether a weld repair is required and if so what type.

The sixth activity is Weld repairs. A weld repair crew follows immediately behind the NDT inspection activities to either carry out repairs to or cut out any defective weld. On completion of all repairs a further x-ray is carried out on the weld to ensure that the finished weld conforms to the standard required. The x-ray of repair welds is usually carried out from the outside of the weld by a two-man crew. The last activity is Field joint coating. The coating of the pipeline field joints to prevent corrosion starts a few days after the welding. This extended period is to allow for any repairs or cut-outs to be completed without prejudicing the coating crew's operations.

The third group of activities is excavation of trenches and installation of pipes. The first activity is Trench excavation. In areas other than rock, trench excavation commences a few days after the field joint coating operation. A typical trench excavation crew consists of 5 - 8 excavators working in line. This operation only excavates the length of open cut trench sufficient to install the main line welded pipe; it does not excavate any roads, ditches, services or obstacles. The number of excavators employed will be such that the amount of trench excavated in a single day matches the rate of progress of the welding crew. The spoil from the trench will be stored adjacent to the trench on the opposite side of the ROW from the topsoil stack. The finished trench will be to the correct depth and width to suit the pipe diameter, plus any bedding and pipe cover. As far as possible, the trench should also be in a straight line so that the pipe can lay central in the trench without touching the trench sides. All loose and jagged outcrops, which could come into contact with the pipe during laying operations, will be removed. See figure (3-4).



Figure 0-4 : Trench Excavation Activity

The second activity is Pipe installation. The pipeline will be positioned approximately 5 meters from the trench centre-line and will be installed into the open unobstructed trench utilizing a number of side-booms. This operation will usually be carried out immediately following the excavation crew. As the pipeline is being installed a coating crew will be present who will holiday detect the pipe to detect any damage to the pipe coating just prior to the pipe entering the trench. Any damage detected will be repaired by a fast setting repair coating. In areas of rock, the pipe installation will commence anything from 5 to 15 days after the welding crew. If there are any above ground breaks in the mainline due to access openings across the ROW, expansion breaks or bend breaks, then these will be welded above ground, x-rayed and coated during the excavation and lowered-in as part of the mainline lower & lay operation. This will optimize the use of the side-booms within the lower & lay crew and reduce the number of below ground tie-ins. See figure (3-5).



Figure 0-5 : Lowering pipes in Trenches

The last activity is Backfilling of the pipeline trench. Trench backfill starts immediately following the placement of the pipeline in the trench and the undertaking of a survey of the pipe levels by the engineers to confirm that the required pipe cover has been achieved. There is a requirement that the initial backfill around the pipe and to 300mm above the crown be of loose and relatively fine particles, which can be readily compacted and do not damage the pipe coating. In areas of rock it will be necessary to place the pipe on a 150mm bed of similar material. In order to provide this material it may be necessary to import sand/soft material offsite, sieve the excavated material or crush the excavated material. The sieve and crusher equipment will be portable machines, which will be transported along the pipeline ROW. The pipe is backfilled over the entire length except for, say, 30 meters at each end of the pipeline work section, which is left free to facilitate the tie -in to the crossing/line break pipe work. The whole operation can be summarized in figure (3-6).

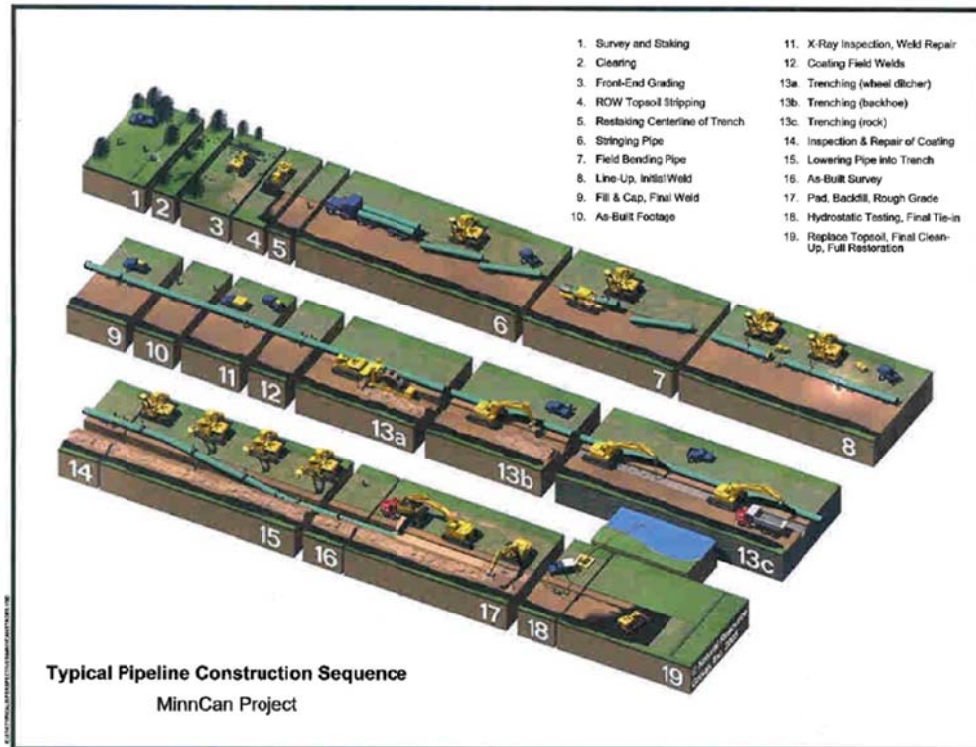


Figure 3-6 : Pipeline Construction Operation

3.3 Summary and Conclusion

This chapter presented a review of the pipeline construction method known as “Spread Technique”. The different stages were shown to show the sequence of work usually followed i.e. the activities precedence, and the resources utilized in each activity. Understanding this sequence is a main step in identifying the scheduling problem and developing the proposed simulation model presented in this research.

Chapter 4

Deterministic Models

4.1 Introduction

This chapter presents the methodology of constructing simple deterministic models based on the literature. There are numerous mathematical models that are used to solve various scheduling problems. Repetitive construction must be classified into a form of the different forms present in the literature in order to choose the appropriate model.

4.2 Methodology

According to the framework established by Pinedo (2011) ($\alpha|\beta|\gamma$), our scheduling problem would be formulated as follows: $FFc \mid prmp, s_{jk}, M_j, prmu \mid C_{max}$. Each part of this formula will be explained in the next paragraphs.

The first part of the formula (α) describes the arrangement of the machines and the sequence in which each job will be processed through these machines. The pipeline project site would be categorized best as *Flexible Flow shop (FFc)*. The construction project consists of a number of stages as it was illustrated earlier (Stringing, Bending, Weldingetc). Each stage utilizes a single or a number of resources (machines) which are either similar to each other or vary in their productivity.

The second part of the formula (β) describes the characteristics of construction stages and the constraints imposed on them. The first characteristic is *preemptions* ($prmp$). It means that after the job is on the machine, it is allowed to stop processing this job for some time and then proceed once again until it's finished. In our problem, this means that after starting a certain activity in one station, it is allowed to stop working in this station, utilize the resources in a different station and then returning the resources to finish working in the first station. The second characteristic is *sequence dependent setup times* (s_{jk}). This means that there is setup time for the resources incurred as the resources are relocated between stations. Pipeline projects extend over hundreds of kilometers and it utilizes heavy machinery such as excavators and side booms. Arrangement of stations in a certain sequence will greatly affect setup times of resources. The third characteristic is *machine eligibility restrictions* (M_j). This means that not all machines in one stage of the project is suitable for all jobs (stations). One major example of that is the (Excavation) activity. As the pipeline passes through different areas, different types of soil starting from loose sand all the way to hard rock may be incurred within the same project. As a result each station would require a different type of excavation equipment depending on the nature of soil. The fourth characteristic is *permutation* ($prmu$). This means that the sequence of work in stations is fixed for all

activities. In linear projects such as construction of pipelines, it is established that work sequence is fixed for all activities. It is even preferred to make the sequence made so that all the equipments move from one end of the pipeline directly to the other end. However, as shown previously in the literature and in the model used in this thesis, *permutation* could be overlooked to get a better schedule.

The third part of the formula (γ) describes the objective that needs to be minimized. Our objective is to minimize the *makespan* (C_{\max}) which is the completion time of the last job. This objective is the common objective in construction projects. In pipeline projects, it is crucial to finish all the stations as early as possible in order to start the operation of it.

4.3 Proposed Models

There have been many approaches to solve scheduling problems for flowshop and flexible flowshops environments. It will be illustrated that these methods are not sufficient to solve the scheduling problem of pipeline construction projects. The methods developed according to Pinedo (2011) will be illustrated in the next part.

4.3.1 Johnson's Rule

This method was developed by Johnson (1954) to minimize the makespan for flowshops with 2 machines ($F^2 \parallel C_{\max}$) problems. It is commonly referred to as Johnson's rule. If there are (n) jobs in one problem. The processing time of job (j) on machine 1 is (p_{1j}) and its processing time on machine 2 is (p_{2j}) . An optimal sequence can be generated as follows. First, divide the jobs into two sets with "Set I" containing all jobs with $(p_{1j}) < (p_{2j})$ and "Set II" containing all jobs with $(p_{1j}) > (p_{2j})$. The jobs with $(p_{1j}) = (p_{2j})$ may be put in either set. Second, the jobs in Set I go first in the sequence and they go in increasing order of (p_{1j}) . "Set I" is referred to as *(SPT)*. Finally, the jobs in "Set II" follow in the sequence in a decreasing order of (p_{2j}) . "Set I" is referred to as *(LPT)*. Such schedule is referred to as "*SPT (1) – LPT (2)*" schedules. The following example illustrates the way this method works.

If we take five jobs (stations) j_1, j_2, \dots, j_5 and two machines (activities) e.g. "Excavation" and "Welding", the following is the durations for the five jobs on the two machines:

Table 4-1: Job durations in man-hours

	Job	j1	j2	j3	j4	j5
Excavation	$P_{1, jk}$	62	86	87	66	56
welding	$P_{2, jk}$	60	75	57	82	76

Where, $p_{1, jk}$ = the duration for job (j_k) on machine (1)

$P_{2, jk}$ = the duration for job (j_k) on machine (2)

The makespan or the total completion time for the five jobs on the two machines equals the completion time for the last job on the second machine ($C_{2,j5}$). The completion time ($C_{i,jk}$) for job (j_k) on machine (i) is calculated using the following formulas:

$$C_{1,j1} = P_{1,j1}$$

(Eq. 4-1)

$$C_{1,jk} = C_{1,j(k-1)} + P_{1,jk} \quad \text{for } k = 2, \dots, 5$$

(Eq. 4-2)

$$C_{2,j1} = C_{1,j1} + P_{2,j1}$$

(Eq. 4-3)

$$C_{2,jk} = \max(C_{1,jk}, C_{2,j(k-1)}) + P_{2,jk} \quad \text{for } k = 2, \dots, 5$$

(Eq. 4-4)

If the jobs were processed on the machines with their default sequence ($j1, j2, j3, j4, j5$), the total completion time equals 459 man-hours as calculated in the following table:

Table 4-2: Completion times for jobs in man-hours

Job Sequence	j1	j2	j3	j4	j5
Completion Time ($C_{i,jk}$)	$C_{1,j}$	$C_{2,j}$	$C_{3,j}$	$C_{4,j}$	$C_{5,j}$
Excavation (machine 1)	62	148	235	301	357
welding (machine 2)	122	223	292	383	459

Next, the makespan is minimized by applying Johnson's rule. The five jobs are divided to two sets (SPT) and (LPT) as follows:

Table 4-3: Dividing jobs into two sets

	Job	j1	j2	j3	j4	j5
Excavation	$P_{1,jk}$	62	86	87	66	56
welding	$P_{2,jk}$	60	75	57	82	76
		LPT	LPT	LPT	SPT	SPT

For (SPT), jobs are arranged in increasing order, so, "j5" comes first and "j4" comes second in the sequence. Followed by these two jobs, comes (LPT) jobs in decreasing order. As a result, the jobs would be arranged in the order "j2", "j1" and finally "j3". Applying this sequence and calculating the total completion time, the resulting total decreases to 414 man-hours.

**Table 4-4: Completion times for jobs after applying Johnson's rule
(In man-hours)**

Job	j5	j4	j2	j1	j3
Completion time (C _{i,jk})	C _{5,j}	C _{4,j}	C _{2,j}	C _{1,j}	C _{3,j}
Excavation (machine 1)	56	122	208	270	357
welding (machine 2)	132	214	289	349	414

In conclusion, Johnson's rule is suitable for getting optimal schedule for (F2 || C_{max}) problems. However, it cannot be generalized to problems with more than two machines (activities). In addition, it cannot be used for flexible flowshops (FFc). Another disadvantage of this method is that the sequence of jobs is fixed through both machines i.e the schedule must be *permutation* (prmu).

4.3.2 Mixed Integer Programming

Wagner (1959) developed a method in order to solve problems with more than two machines (F_m | prmu | C_{max}). His method was based on formulating the problem as a Mixed Integer Program (MIP). First, the variables are defined. The decision variable (x_{jk}) equals 1 if job (j) is the k th job in the sequence and 0 otherwise. The auxiliary variable (I_{ik}) denotes the idle time on machine (i) between the processing of the jobs in the k th position and ($k + 1$)th position and the auxiliary variable (W_{ik}) denotes the waiting time of the job in the k th position in between machines (i) and ($i+1$). Wagner (1959) stated that minimizing the makespan is equivalent to minimizing the total idle time on the last machine, machine m . hence, the problem was formulated as follows:

$$\min \left(\sum_{i=1}^{m-1} \sum_{j=1}^n x_{j1} p_{ij} + \sum_{j=1}^{n-1} I_{mj} \right) \quad (Eq. 3-5)$$

With the following constraints:

$$\sum_{j=1}^n x_{jk} = 1 \quad \text{for } k = 1, \dots, n \quad (Eq. 3 - 6)$$

$$\sum_{k=1}^n x_{jk} = 1 \quad \text{for } j = 1, \dots, n \quad (\text{Eq. 3 - 7})$$

$$\begin{aligned} I_{ik} + \sum_{j=1}^n x_{j,k+1} p_{ij} + W_{i,k+1} - W_{ik} - \sum_{j=1}^n x_{jk} p_{i+1,j} - I_{i+1,k} \\ = 0 \\ \text{for } k = 1, \dots, n-1; \quad i \\ = 1, \dots, m-1 \end{aligned} \quad (\text{Eq. 3 - 8})$$

$$\begin{aligned} W_{i1} = 0 \quad \text{for } i \\ = 1, \dots, m-1 \end{aligned} \quad (\text{Eq. 3 - 9})$$

$$\begin{aligned} I_{1k} = 0 \quad \text{for } k \\ = 1, \dots, n-1 \end{aligned} \quad (\text{Eq. 3 - 10})$$

The first set of constraints (Eq. 3-6) specifies that exactly one job has to be assigned to position (k) for any (k). The second set of constraints (Eq. 3-7) specifies that job (j) has to be assigned to exactly one position. The third set of constraints (Eq. 3-8) relates the decision variables (x_{jk}) to the physical constraints. These physical constraints enforce the necessary relationships between the idle time variables and the waiting time variables. The fourth set of constraints (Eq. 3-9) insures that the waiting time for the first job equals zero on all machines. The last set of constraints (Eq. 3-10) insures that the idle time for the first machine equals zero for all jobs.

The following example illustrates the way this method works. If we take five jobs (stations) j1, j2, ..., j5 and three machines (activities) e.g. "Excavation", "Welding" and "Lowering of pipes", the following is the durations for the five jobs on the three machines:

Table 4-5: Job durations in man-hours

	Job	j1	j2	j3	j4	j5
Excavation	P _{1,jk}	62	86	87	66	56
welding	P _{2,jk}	60	75	57	82	76
Lowering	P _{2,jk}	78	82	77	87	80

Following the default sequence (j1, j2, j3, j4, j5) for processing the jobs on all three machines, the total completion time equals 550 man-hours as calculated using equations (3-1) to (3-4) in the following table:

Table 4-6: Completion times for jobs in man-hours

Job Sequence	j1	j2	j3	j4	j5
Completion Time (C _{i,jk})	C _{1,j}	C _{2,j}	C _{3,j}	C _{4,j}	C _{5,j}
Excavation (machine 1)	62	148	235	301	357
welding (machine 2)	122	223	292	383	459
Lowering (machine 3)	200	305	382	470	550

Next, the Mixed Integer Program (MIP) is used to minimize the makespan. A MS Excel worksheet (see figure (1)) is set as follows. First, the matrix of job durations (p_{ij}) [table (4-5)] is set as the input matrix. Second, three matrices are set as variables matrices: the decision variable matrix (x_{jk}) [table (4-7)], the waiting time matrix (W_{ik}) [table (4-8)] and the idle time matrix (I_{ik}) [table (4-9)]. In table (4-7), Cells (D11:H15) can take one of two values (0 or 1) so that each the total of each row and column equals one to satisfy equations (3-6) and (3-7). Cells (U36: X37) in table (8) and (L37: O38) in table (4-9) can take any value \geq zero. However, these values must satisfy the set of constraints (Eq. 3-8) which in this example would add up to 8 equations e.g. for the second machine ($i=2$) and third job ($k=3$), the constraint would be:

$$I_{2,3} + \sum_{j=1}^n x_{j,4} p_{2,j} + W_{2,4} - W_{2,3} - \sum_{j=1}^n x_{j,3} p_{3,j} - I_{3,3} = 0$$

Table 4-7: Decision Variable Matrix (x_{jk})

Excel Cell Designation	C	D	E	F	J	H	I
10	X_{jk}	j1	j2	j3	j4	j5	Σ
11	1	1	0	0	0	0	1
12	2	0	0	0	0	1	1
13	3	0	0	1	0	0	1
14	4	0	0	0	1	0	1
15	5	0	1	0	0	0	1
16	Σ	1	1	1	1	1	

Table 4-7: Waiting Time Matrix (Wik)

Excel Cell Designation	S	T	U	V	W	X	Y
35		j1	j5	j3	j4	j2	Σ
36	W _{1,k}	0	4	0	2	1	7
37	W _{2,k}	0	2	18	2	11	33
38	W _{3,k}	0	0	0	0	0	0
39	Σ	0	6	18	4	12	40

Table 4-8: Idle Time Matrix (lik)

Excel Cell Designation	K	L	M	N	O	P	Q
35		j1	j5	j3	j4	j2	Σ
36	I _{1,k}	0	0	0	0	0	0
37	I _{2,k}	0	7	11	3	0	21
38	I _{3,k}	0	0	0	0	0	0
39	Σ	0	7	11	3	0	21

The target cell, for which the objective is to be minimized, would contain the equation:

$$\min(\sum_{i=1}^2 \sum_{j=1}^5 x_{j1} p_{ij} + \sum_{j=1}^4 I_{3,j})$$

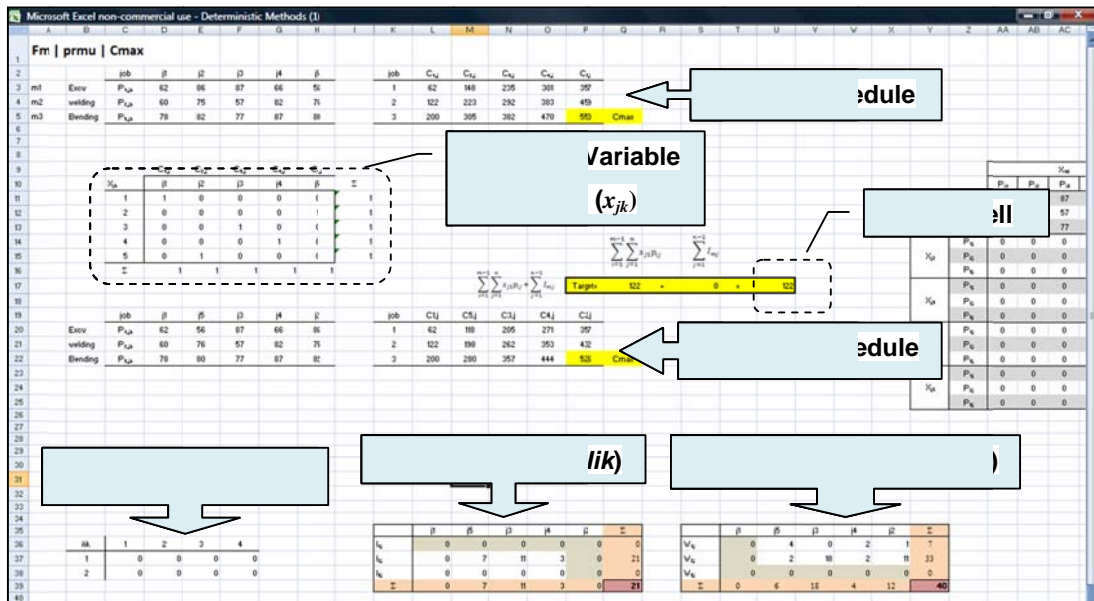


Figure 4-1 : MS Excel Worksheet Used for (MIP)

The resulting schedule would decrease the makespan to 526 man-hours as the following sequence is followed:

Table 4-9: Completion times for jobs in man-hours

Job Sequence	j1	j5	j3	j4	j2
Completion Time ($C_{i,jk}$)	$C_{1,j}$	$C_{5,j}$	$C_{3,j}$	$C_{4,j}$	$C_{2,j}$
Excavation (machine 1)	62	118	205	271	357
Welding (machine 2)	122	198	262	353	432
Lowering (machine 3)	200	280	357	444	526

In conclusion, MIP is suitable for $(F_m \mid prmu \mid C_{max})$. It can be used to any number of machines and jobs. However, like Johnson's rule, it cannot be used for flexible flowshops (FFc) and the sequence of jobs is fixed through all machines i.e. the schedule must be *permutation* (prmu). In addition, $(F3 \mid \mid C_{max})$ problems were found to be strongly NP-Hard (Pinedo, 2011). Thus, any problem with more than three machines would also be strongly NP-Hard.

4.3.3 Slope Heuristic:

Another method to solve scheduling problems of type $(F_m \mid prmu \mid C_{max})$ was developed by Palmer (1965). It was based on the same principle used by Johnson (1954). Jobs with small processing times on the first machine and large processing times on the second machine should be positioned more towards the beginning of the sequence, while jobs with large processing times on the first machine and small processing times on the second machine should be positioned more towards the end of the sequence. According to this heuristic a slope index (A_j) is computed for each job. It is defined as:

$$A_j = -\sum_{i=1}^m (m - (2i - 1))p_{ij} \quad (\text{Eq. 3-5})$$

Where, m = total no. of machines

i = machine number

p_{ij} = processing time of job (j) in machine (i)

The jobs are then arranged in a decreasing order of the slope index. The following example illustrates the way this method works. If we take five jobs (stations) j1, j2, ..., j5 and three machines (activities) e.g. "Excavation", "Welding" and "Lowering of pipes", the following is the durations for the five jobs on the three machines:

Table 4-10: Job durations in man-hours

	Job	j1	j2	j3	j4	j5
Excavation	P _{1,jk}	62	86	87	66	56
welding	P _{2,jk}	60	75	57	82	76
Lowering	P _{2,jk}	78	82	77	87	80

Following the default sequence (j1, j2, j3, j4, j5) for processing the jobs on all three machines, the total completion time equals 550 man-hours as calculated using equations (3-1) to (3-4) in the following table:

Table 4-11: Completion times for jobs in man-hours

Job Sequence	j1	j2	j3	j4	j5
Completion Time (C _{i,jk})	C _{1,j}	C _{2,j}	C _{3,j}	C _{4,j}	C _{5,j}
Excavation (machine 1)	62	148	235	301	357
welding (machine 2)	122	223	292	383	459
Lowering (machine 3)	200	305	382	470	550

Next, the slope heuristic method is applied by calculating the slope index (A_j) for each of the five jobs using equation (3-5). For example, for (j1):

$$A_{j1} = -(3 - ((2 \times 1) - 1) \times 62 - (3 - ((2 \times 2) - 1) \times 60 - (3 - ((2 \times 3) - 1) \times 78$$

$$A_{j1} = 32$$

The following table presents values of (A_j) for jobs j1 to j5:

Table 4-12: Slope Index for jobs

Job	j1	j2	j3	j4	j5
Slope Index (A_j)	32	-8	-20	42	48

According to the jobs' slope index (A_j), the jobs are arranged in the sequence (j5, j4, j1, j2, j3). Applying this sequence and calculating the total completion time, the resulting total decreases to 538 man-hours.

Table 4-13: Completion times for jobs in man-hours

Job Sequence	j5	j4	j1	j2	j3
Completion Time ($C_{i,jk}$)	$C_{5,j}$	$C_{4,j}$	$C_{1,j}$	$C_{2,j}$	$C_{3,j}$
Excavation (machine 1)	56	122	184	270	357
welding (machine 2)	132	214	274	349	414
Lowering (machine 3)	212	301	379	461	538

In conclusion, slope heuristic is suitable for $(F_m \mid \text{prmu} \mid C_{\max})$. It can be used to any number of machines and jobs. However, like MIP, it cannot be used for flexible flowshops (FFc) and the sequence of jobs is fixed through both machines i.e. the schedule must be *permutation* (prmu).

4.4 Summary and Conclusion

After testing all the methods demonstrated by Pinedo (2011), they all have proven, as shown above, to be inadequate to solve the scheduling problem of pipeline construction projects due to the following reasons. First, all three methods are not suitable for flexible flowshops (FFc). They only deal with one machine per stage which is not applicable. In pipeline construction, each stage usually utilize a number of machines (crews and equipments), which may be different in their productivity. Second, the large number of stages involved would, if they were reduced to a single machine, result in a huge problem for which an optimum solution would be impossible to find. Third, all three methods inflict the *permutation* (prmu) condition in the schedule. As mentioned earlier, overlooking this condition would result in a much optimal schedule.

CHAPTER 5

SIMULATION MODEL

5.1. Introduction

As addressed in the previous chapters, heuristics and analytical methods are incapable of finding the optimal schedule for linear projects, involving various activities and resources. Simulation modeling, which represents a powerful alternative, would be illustrated in this chapter.

The simulation method used is Discrete Event Simulation. This method is based on modeling the operation of any system as a discrete sequence of events occurring in different instances of time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation can directly jump in time from one event to the next.

5.2. Simulation Software

The simulation software used is called “AnyLogic”. It is a general-purpose modeling and simulation tool for discrete, continuous and hybrid systems. It supports all three well-known modeling approaches: System dynamics, Discrete event simulation, Agent-based modeling in addition to any combination of these approaches within a single model. AnyLogic includes a graphical modeling language i.e. the model is built in a graphical editor that allows the user to edit the diagram of the model graphically. It also allows the user to extend simulation models using Java code.

The “Active objects” are the main building blocks of AnyLogic models. Active objects can be used to model very diverse objects of the real world such as processing stations, resources, and various operations. Active objects may encapsulate other active objects to any desired depth. This enables building the model from as many levels of details as required; each active object typically represents a logical section of the model. Each AnyLogic model has a main active object which contains embedded objects which, in turn, may contain their embedded objects, and so on. These embedded objects serve as tools to facilitate modeling the events of the process.

These objects are assembled in a number of libraries. One of the main libraries is called the “Enterprise Library”. The Enterprise Library supports discrete-event, or, to be more precise, process-centric modeling paradigm. This library’s tools are used to create discrete event patterns frequently used in process-centric modeling such as queuing, resource usage and entity generation. Using the Enterprise Library objects, the real-world systems can be modeled in terms of entities (transactions, customers, products, parts, vehicles, etc.), processes (sequences of operations typically involving queues, delays, resource utilization),

and resources. The Enterprise Library contains a set of objects specifically designed for “Network Based Modeling”.

Network-based or layout-based modeling is used to model processes that take place in a certain physical space, referred to as a Network, with moving entities and resources. To use the “Network” set of objects, the network topology needs to be defined. A network is a set of nodes interconnected with segments. It may have parts that are not connected to each other. The entities and resources are automatically animated moving along the network segments or staying at nodes. Movement always is done along the shortest path between the origin and the destination nodes. Entities and resource units may have individual speeds; moreover, those speeds may change dynamically. For example, you can set different speed for loaded and unloaded trucks. It is assumed that segments have unlimited capacity, so entities moving along a segment do not interfere.

There are two main classes that are used in discrete event models: *Entity* and *ResourceUnit*. *Entity* is a base class for all entities, that are generated, access resources and take part in the process flow in process-centric models. An entity may represent a person, a document, a piece of information or a vehicle. Entity is a regular Java class with functionality sufficient for the Enterprise Library objects to handle and animate it. Its functionality could be extended by creating a costume entity subclass, adding custom fields to it accessing them from the process model. Enterprise Library objects are used to handle entities through the process whether by generating them like **Source**, **Combine** and **Split**, disposing of them like **Sink**, handling resources like **Seize**, **Release** and **Service**, controlling their flow through the process like **Queue**, **Hold** and **SelectOutput** or Network-based objects such as **NetworkMoveTo**, **NetworkSeize** and **NetworkSendTo**.

The corresponding class used in models is *ResourceUnit*. *ResourceUnit* is a base class for all types of resources. Each resource type belongs to either a *ResourcePool* object or *NetworkResourcePool* object. Like the Entity class, *ResourceUnit* is a regular Java class with functionality sufficient for the Enterprise Library objects to handle and animate it. Its functionality could be extended by creating a costume entity subclass, adding custom fields to it accessing them from the process model. *NetworkResourcePool* is a resource pool that is used in Network-based Modeling. Its resource units are similar to the “regular” ones, those that are defined with *ResourcePool* object, but have additional properties that help in managing them within the network. Each resource unit has its home node in the network which could be, for instance, a storage yard for equipment or a base camp for labor. The resource units can be static, moving, or portable. Static resources are bound to a particular location, i.e. a node, within the network and cannot move or be moved. An example of a static resource would be tower crane or workshop machinery. Moving resources can move on their own; they can represent workers or vehicles. Portable resources can be moved by entities or by moving resources. Portable devices or construction materials would be an example of

portable resources. Moving and portable resources have their home locations where they can optionally return or be returned.

Resource units are utilized by entities during the operation of the process. The resource management in a network is done centrally. The Network object maintains the queue of requests from entities that want to seize the network resources and processes them from front to back. Requests are arranged, by default, according to the rule "First In, First Out" (FIFO), but optionally it can be a priority queue where requests are arranged based on priorities of request which depend on the entities. If a request can be satisfied (i.e. all requested resource units are simultaneously available), the units will be allocated, otherwise the units that are available will be "reserved" by that request and the request stays in the queue. This means that a request from the middle of the queue can be satisfied only if it does not conflict with any request in front of it.

5.3. Model Development

As mentioned previously, AnyLogic depends on a graphical interface; it allows the user to build the model using the libraries of active objects by "Drag & Drop". The interface [figure (5-1)] consists of a number of views as follows:

- *Graphical Editor*: Each active object class has a graphical editor associated with it. The graphical editor is the place where the structure of the active object class is defined. It plays several roles:
 - Defines the interface of the active object class.
 - Defines a presentation and icon for the active object using presentation shapes and controls. Graphical editor links shape properties to active object data and embedded objects.
 - Defines behavior elements, such as events and state charts.
 - Defines the embedded objects and their interconnection.
- *Project View* provides access to projects currently opened in the workspace. The workspace tree provides easy navigation throughout the models. As models are organized hierarchically, they are displayed in a tree structure.
- *Palette View* lists the model elements grouped in palettes. An element is added to the model by dragging it from the palette to the graphical editor.
- *Properties View* is used to view and modify the properties of the selected model item(s).

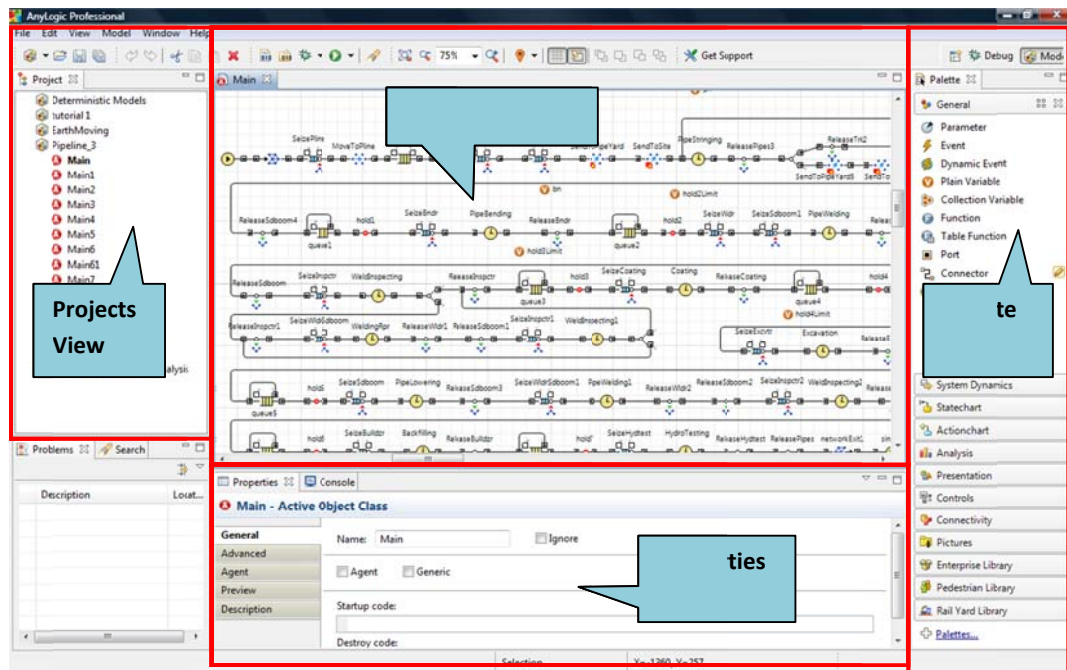



Figure 5-1: Typical Interface in AnyLogic

Building the model will go through five main stages as follows:

- 1- Creating the model animation by drawing the network that represents the actual space of the project including all the nodes and the paths linking them. In addition, the shapes, that represent the entities and resources used in the model, are created.
- 2- Creating the costume classes for entities and resources as needed i.e. adding additional characteristics to entities' and resources' classes
- 3- Creating the diagram of the model by adding the needed objects from the libraries in the graphical editor, modifying the objects' properties and defining the relationships between them to fit the logic of the actual process.
- 4- Creating the "Simulation" experiment which runs model simulation with animation displayed and model debugging enabled. The first experiment in each model is automatically created.
- 5- Creating the "Optimization" experiment which is used to find the optimal combination of conditions resulting in the best possible solution by making decisions about system parameters and/or structure.

5.3.1. Stage (1): Creating of Model Animation

The first stage is to draw the elements required for the animation of the model components. These components include 1) the network i.e. the nodes and paths between them and 2) the resources. The library used is the *presentation* library (palette) (Fig.5-2). The tools with  icons support the “drawing” mode in addition to “Drag & Drop” mode. Nodes are usually represented by rectangles while paths between them are represented by lines or polylines.

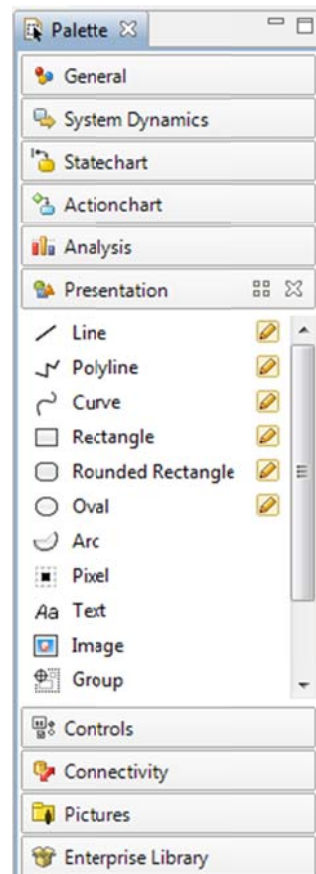


Figure 5-2: Palette View Window

Pipeline construction projects rely on a number of camps set along the pipeline path; these camps are well-equipped for housing of the workers e.g. welders and storage of construction equipment such as trucks and bulldozers and material such as pipes. Each camp is located properly to serve a segment of the pipeline. The model would represent only one construction camp and the segment it covers. The segment would be divided it into ten stations which are equal in length. The construction camp consists of a base camp for the residence of workers and storage of equipment in addition to a pipe yard for storage of pipes. A temporary road is constructed alongside the pipeline path. The length of this road on the model represents 90

km in reality. The speed of each equipment is set to match this path's line on the model. The steps of the first stage are as follows:

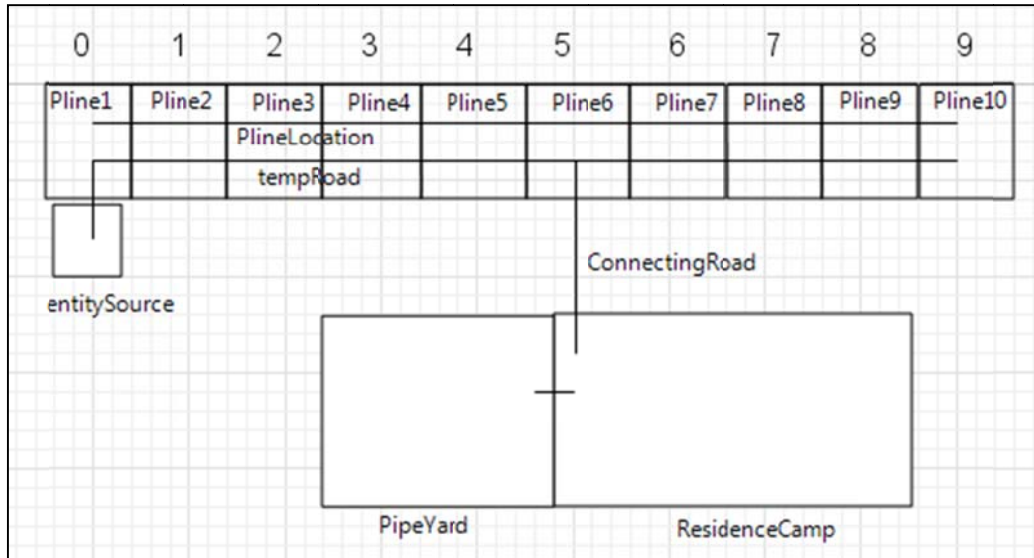


Figure 5-3: Layout of Pipeline Project in AnyLogic

1.1) A rectangle is drawn using the tool *Rectangle* either in “Drag & Drop” mode by dragging the tool from the *Presentation* palette to the graphical editor and modifying its size using the handles on the shape’s border or in Drawing mode by clicking on tool next to the *Rectangle* and drawing the shape with the required size in the graphical editor.

1.2) Modify the name of the rectangle from the properties view after selecting it from the graphical editor into *ResidenceCamp* and modify its colors as needed.

1.3) Repeat steps (1.1) & (1.2) to draw the rectangles with the names *PipeYard*, *entitySource*, *Pline1* through *Pline10*. Rectangles *ResidenceCamp* and *PipeYard* would be used as home nodes for the resources as explained later. Rectangles *Pline1* through *Pline10* represent the ten stations that compose the stretch of the pipeline served by this camp. The point of entry of the entities to the network would be the *entitySource* rectangle. The arrangement of rectangles is shown in figure (5-3).

1.4) Three lines are drawn using the tool *Line* either in “Drag & Drop” or Drawing modes as illustrated in step (1.1). One line is to connect *entitySource* and *Pline1* rectangles. The second connects *ResidenceCamp* and *PipeYard* rectangles. The third line, called *ConnectingRoad*, is used to connect *ResidenceCamp* with *Pline6*.






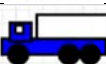




1.5) Two polylines are drawn using the tool *Polyline* in drawing mode as illustrated in step (1.1). The two polylines are used to connect rectangles *Pline1* through *Pline10* by placing one point of the polyline in each rectangle. One polyline would be named *PlineLocation*. It would be set as a home path for a type of resource as explained later.

The other polyline would be named `tempRoad`. It represents the temporary road constructed along the path of the pipeline. This road would be utilized for the transportation of resources and material between the pipeline's stations. The arrangement of lines and polylines is shown in figure (5-3).

1.6) A new group is created using the tool *Group* by grabbing it from the *Presentation* palette to the graphical editor and named `networkGroup`. All shapes, created in steps (1.1) to (1.5), would be put together in the group.

Now that the network is created, the animation shapes used for the resources are drawn. Shapes are grabbed from the *Presentation* palette to the graphical editor and added together in groups as illustrated in steps (1.1) to (1.6) to form the shapes of the different resources as shown in table (5-1). Using these shapes in the animation of the model would be illustrated in the second stage.

Table 5-1: Names and Animation Shapes of Model Components

Description	Name (in the model)	Animation shape
Pipes Truck	TruckShape	
Side boom	SideboomShape	
Pipe bending machine	PipebenderShape	
Excavator	ExcavShape	
Bulldozer	BulldozerShape	
Hydro testing Crew	HydTestShape	
Pipes	PipeShape	
Welder	WelderShape	
Coating Crew	CoatShape	
Inspection Team	InspectorShape	

5.3.2. Stage (2): Creating of Costume Classes

The second stage is creating costume classes for entities and resources. Some models require adding additional characteristics to entities' and resources' classes in order to have a better model of the problem. This process results in a subclass of class `Entity` or `ResourceUnit`. A subclass inherits the properties of its super class in addition to the

properties added by the user. In this model, entities represent stations of the pipeline. These stations have various properties that affect the productivity of resources for each activity. As a result, these properties should be added to a subclass named `Station` of the class `Entity`. On the other hand, all units of each resource type would be assumed to have similar characteristics e.g. productivity, speed...etc. in order to simplify the model. The steps to create the subclass `station` are as follows:

2.1) In the Project view, right-click the model item *Main* which is the main object where the model is built, and choose **New | Java Class** from the popup menu

2.2) The **New Java Class** wizard is displayed. On the first page of the wizard, the name of the new Java class `Station` is specified in the Name field and the superclass name `Entity` is chosen in the Superclass edit box as in figure (5-4)

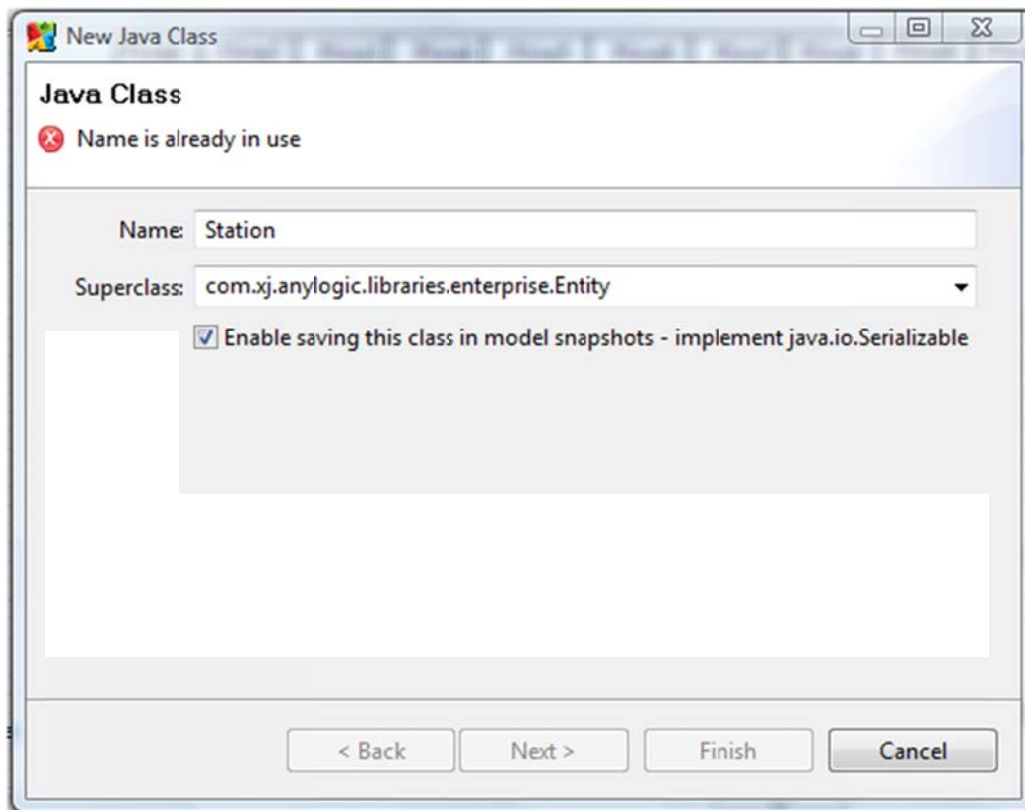


Figure 5-4: First Window in “New Java Class” wizard

2.3) Click **Next** to go to the next page of the wizard. On the second page of the wizard, Java class Fields are specified in the table, each class field is defined in the separate row (figure 5-5). The type of the field is entered in the Type cell, name of the field in the Name cell and optionally name the access modifier in the Access cell and the initial value is specified in the Initial value cell. The data entered in this table are presented in table (5-2).

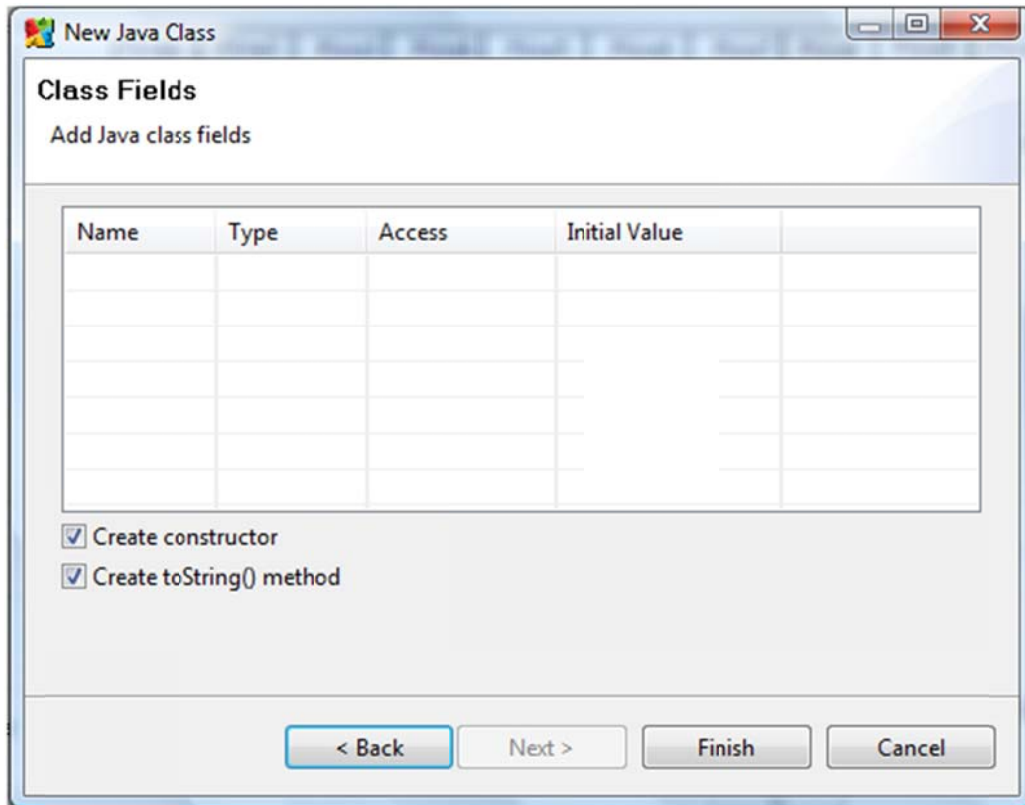


Figure 5-5: Second Window in “New Java Class” wizard

Table 5-2: New Fields of Entity Class

Name	Type	Access	Initial Value	Description
PipeNo	double	Public	0	Counter of the number of pipes sent to a station
StrngPriority	double	Public	0	The station’s priority in entering “Pipe Stringing” activity
TruckNo	double	Public	0	Number of trucks assigned to a station
BendPriority	double	Public	0	The station’s priority in entering “Pipe Bending” activity
BendNo	double	Public	0	Factor to represent the number of bends needed for the pipes of a station. It depends on the station’s topology.
WeldPriority	double	Public	0	The station’s priority in entering “Welding” activity
WelderNo	double	Public	0	Number of welder teams assigned to a station

WeldPipeNo	double	Public	0	Counter of the number of welded pipes in a station
CoatPriority	double	Public	0	The station's priority in entering "Coating" activity
Excvdifff	double	Public	0	Factor that represents the type of soil of a station and the difficulty of excavation in it.
ExcPriority	double	Public	0	The station's priority in entering "Excavation" activity
ExcNo	double	Public	0	Number of excavators assigned to a station
LwrPriority	double	Public	0	The station's priority in entering "Pipe Lowering" activity
BckflPriority	double	Public	0	The station's priority in entering "Backfilling" activity
BldzrNo	double	Public	0	Number of bulldozers assigned to a station
HdrtstPriority	double	Public	0	The station's priority in entering "Hydrotesting" activity

2.4) Using Create constructor and Create toString() method check boxes, default class constructor and toString() method are created automatically.

2.5) Click Finish to complete the process. The code editor for the created class would be opened. The code for subclass `Station` is presented in appendix-A

5.3.3. Stage (3): Creating of Model Diagram

The third stage is constructing the diagram of the model in a proper way to represent the actual events occurring in the actual process. The technique used to complete this stage mainly depends on adding the needed objects from the libraries into the graphical editor, modifying the objects' properties and defining the relationships between them to fit the logic of the actual process. This stage contains a large number of steps, thus it will be divided into several sub-stages.

Sub-stage (1): Definition of Resource Pools

3.1) Add **Network** object by dragging it from the *Enterprise* palette into the graphical editor. The object's properties are set by selecting it and modifying the needed fields in the *properties* view. The properties are as follows:

Name:	network	(The default name)
Group of network shapes:	networkGroup	(It links the Network object to the graphical network created in the first stage. Refer to step (1.6))
Enable priorities:	checked	(This allows entities to be arranged according to their priorities)
Request priority:	0	(The default value)

3.2) Add **NetworkResourcePool** by dragging it from the *Enterprise* palette into the graphical editor. The object's properties are set by selecting it and modifying the needed fields in the *properties* view (see figure 5-6). The properties are as follows:

Name:	Pline	(The name for the resource pool of pipeline stations' location)
Resource type:	Static	
Capacity defined:	By home shape	(The home shape is the PlineLocation polyline. It has ten nodes, thus, the capacity of this resource pool would be also ten). Refer to step (1.5)
Home defined by:	Path across nodes	(It defines the home of resources as a number of nodes on a path specified in the next field)
Home path:	PlineLocation	

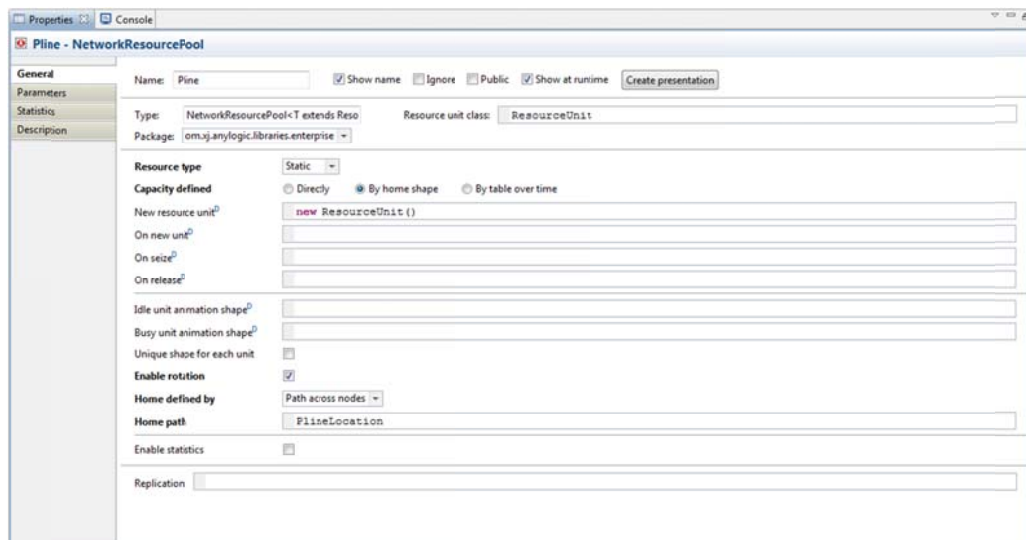


Figure 5-6: Properties View for “Pline” Resource Pool

3.3) Repeat step (3.2) to create resource pools for the resources: Pipes, Trucks, Side booms, Pipe benders, Welders, Excavators, Inspection teams, Coating teams, Bulldozers and Hydro-testing teams. The properties of each resource pool are shown in appendix-B.

3.4) To add the defined resources into the network, the ports of **NetworkResourcePool** objects are connected with the port of the **Network** object as shown in figure (5-7).

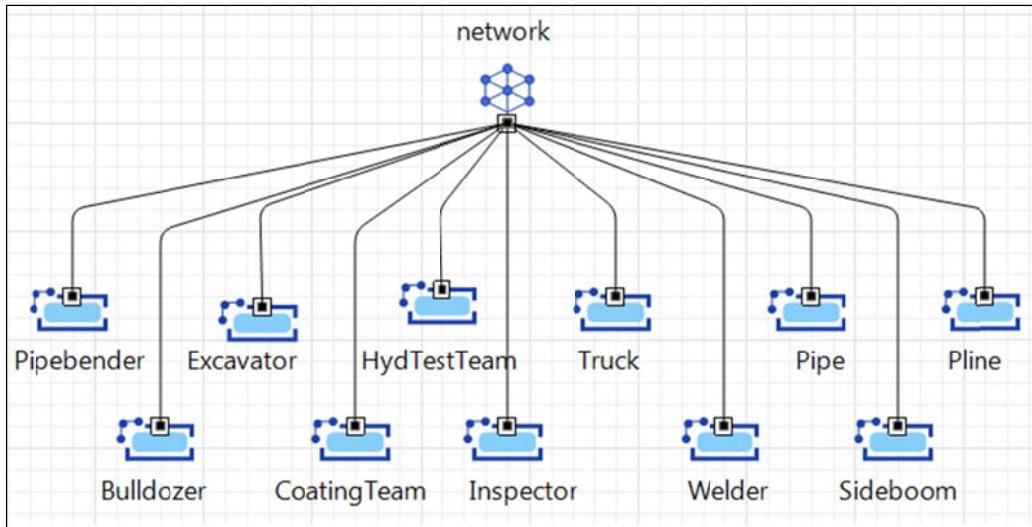


Figure 5-7: Sub-stage (1): Definition of Resource Pools

After finishing the definition of resource pools, the next steps will demonstrate constructing the flowchart describing the process. The final complete flowchart is shown in figure (5-8). The following steps will show the modifications done in each element of the flowchart and the group of objects that represent each activity will be shown later in separate figures.

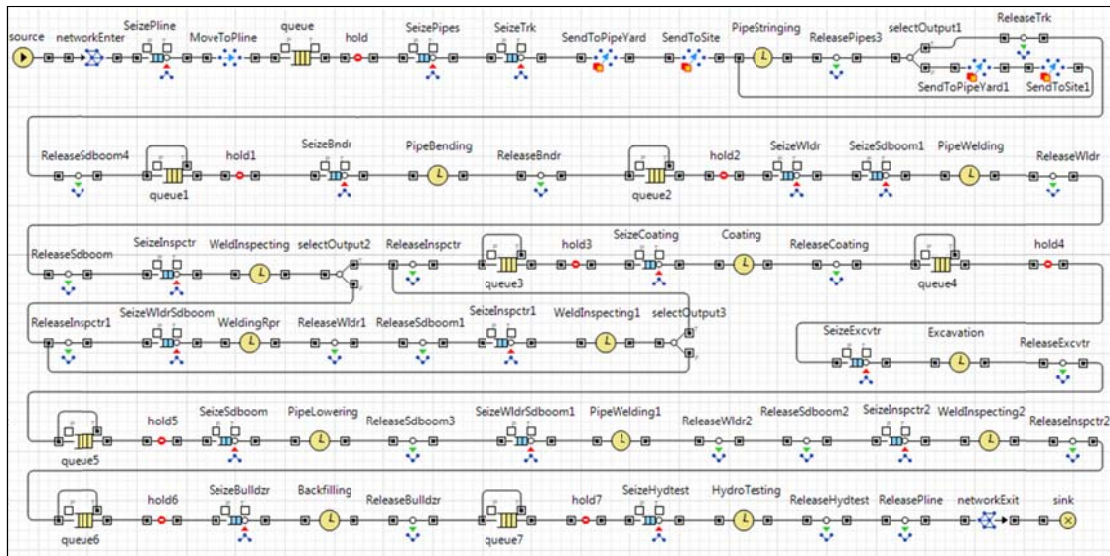


Figure 5-8: Complete Flowchart of Model

Sub-stage (2): Creating Entities

This sub-stage contains the steps of creating the entities that represent the stations, as discussed before, and placing these entities in the network.

4.1) The flowchart starts with **Source** object drawn from the *Enterprise* palette. This object generates entities. It is usually a starting point of a process model. Entities generated will be of the subclass *Station*. The subclass created in the second stage was modified from the super class *Entity* by adding various attributes in order to utilize them in the model. After dragging the object into the graphical editor, the following properties are assigned to it.

Name:	source	(The default name)
Entity class:	Station	(The name of the subclass created in second stage)
Arrivals defined by:	Rate	
Arrival rate:	1	
Entities per arrival:	10	
Limited number of arrivals:	Checked	
Max. number of arrivals:	1	
New entity:	new Station()	

4.2) The next object **NetworkEnter** is drawn next to **Source** object and a connector is drawn between **Source** object's only port and the left port of **NetworkEnter** object. In this object, each generated entity's attributes would be assigned with a value. These values are stored in a number of arrays that would be listed in the next step. This process is done through a Java code written in the **On enter** field of the object. As each entity enters the object, it is assigned with one value from each array that corresponds to its order of entry i.e. the first entity takes the first value of each array and the second entity takes the second value and so on. The following properties are assigned to **NetworkEnter**:

Name:	networkEnter	(The default name)
Entity class:	Entity	(The name of the subclass created in second stage)
Network:	network	(The network defined in step (3.1))
Entry node:	entitySource	(Part of the graphical network. Refer to step (1.3))
On enter:	(The following code is executed as every entity enters the object)	
<pre>int i=j; strngProp1=stationProps1[i]; if(entity instanceof Station) ((Station)entity).StrngPriority = strngProp1; strngProp2=stationProps2[i]; if(entity instanceof Station) ((Station)entity).TruckNo = strngProp2; // bendProp1=stationProps3[i]; if(entity instanceof Station) ((Station)entity).BendPriority = bendProp1; bendProp2=stationProps4[i]; if(entity instanceof Station) ((Station)entity).BendNo = bendProp2;</pre>		

```

//
weldProp1=stationProps5[i];
if( entity instanceof Station ) ((Station)entity).WeldPriority =
weldProp1;
weldProp2=stationProps6[i];
if( entity instanceof Station ) ((Station)entity).WelderNo =
weldProp2;
//
coatProp1=stationProps7[i];
if( entity instanceof Station ) ((Station)entity).CoatPriority =
coatProp1;
//
excvProp1=stationProps8[i];
if( entity instanceof Station ) ((Station)entity).Excvdiff =
excvProp1;
excvProp2=stationProps9[i];
if( entity instanceof Station ) ((Station)entity).ExcvPriority =
excvProp2;
excvProp3=stationProps10[i];
if( entity instanceof Station ) ((Station)entity).ExcvNo =
excvProp3;
//
lwrProp1=stationProps11[i];
if( entity instanceof Station ) ((Station)entity).LwrPriority =
lwrProp1;
//
bckflProp1=stationProps12[i];
if( entity instanceof Station ) ((Station)entity).BckflPriority =
bckflProp1;
bckflProp2=stationProps13[i];
if( entity instanceof Station ) ((Station)entity).BldzrNo =
bckflProp2;
//
hdtstProp1=stationProps14[i];
if( entity instanceof Station ) ((Station)entity).HdtstPriority =
hdtstProp1;
j++;

```

4.3) The arrays used in the previous step are plain variables where ten values are stored in each one. The **Plain Variable** object is drawn from the *General* palette into the graphical editor. Fourteen objects (stationProps1 to stationProps14) will be created and modified as shown in the following table:

Table 5-3: Arrays Properties

Plain Variable 1	Name:	stationProps1	Array for StrngPriority
	Type:	Other: int []	
	Initial Value:	new int [] {1,2,3,4,5,6,7,8,9,10}	
Plain Variable 2	Name:	stationProps2	Array for TruckNo
	Type:	Other: int []	
	Initial Value:	new int [] {2,2,1,2,1,1,1,2,2,1}	
Plain	Name:	stationProps3	Array for

Variable 3	Type:	Other: int []	BendPriority
	Initial Value:	new int [] {1,2,3,4,5,6,7,8,9,10}	
Plain Variable 4	Name:	stationProps4	Array for BendNo
	Type:	Other: int []	
	Initial Value:	new int [] {1,2,1,2,2,3,2,0,1,2}	
Plain Variable 5	Name:	stationProps5	Array for WeldPriority
	Type:	Other: int []	
	Initial Value:	new int [] {1,2,3,4,5,6,7,8,9,10}	
Plain Variable 6	Name:	stationProps6	Array for WelderNo
	Type:	Other: int []	
	Initial Value:	new int [] {2,2,1,2,1,1,1,2,2,1}	
Plain Variable 7	Name:	stationProps7	Array for CoatPriority
	Type:	Other: int []	
	Initial Value:	new int [] {1,2,3,4,5,6,7,8,9,10}	
Plain Variable 8	Name:	stationProps8	Array for Excvdif
	Type:	Other: int []	
	Initial Value:	new int [] {1,2,1,2,2,3,2,1,1,2}	
Plain Variable 9	Name:	stationProps9	Array for ExcVPriority
	Type:	Other: int []	
	Initial Value:	new int [] {1,2,3,4,5,6,7,8,9,10}	
Plain Variable 10	Name:	stationProps10	Array for ExcVNo
	Type:	Other: int []	
	Initial Value:	new int [] {2,2,1,2,1,1,1,2,2,1}	
Plain Variable 11	Name:	stationProps11	Array for LwrPriority
	Type:	Other: int []	
	Initial Value:	new int [] {1,2,3,4,5,6,7,8,9,10}	
Plain Variable 12	Name:	stationProps12	Array for BckflPriority
	Type:	Other: int []	
	Initial Value:	new int [] {2,2,1,2,1,1,1,2,2,1}	
Plain Variable 13	Name:	stationProps13	Array for BldzrNo
	Type:	Other: int []	
	Initial Value:	new int [] {2,2,1,2,1,1,1,2,2,1}	
Plain Variable 14	Name:	stationProps14	Array for HdrtstPriority
	Type:	Other: int []	

	Initial Value:	new int [] {1,2,3,4,5,6,7,8,9,10}	
--	-----------------------	--------------------------------------	--

4.4) The next object **NetworkSeize** is drawn from the *Enterprise* palette next to **NetworkEnter** object and a connector is drawn between its right port and the left port of **NetworkSeize** object. **NetworkSeize** object is used to seize a given set of network resources and optionally attaches them to the entity. It is considered as a queue for the entities waiting for the required resources. As explained earlier, assigning resources to the entities follows either the “First In, First Out” (FIFO) rule or based on priorities of request which depend on the entities. This object would be used many times in the model to seize different sets of resources depending on the activity performed. The first usage of **NetworkSeize** object would be to seize a station for each entity. The following properties are assigned to it:

Name: SeizePline
Entity class: Entity
List of Resources: {Pline} (The name for the resource pool of pipeline stations' location)

4.5) The next object is **NetworkMoveTo** drawn from the *Enterprise* palette next to **NetworkSeize** object and a connector is drawn between **NetworkSeize** object's right port and the left port of **NetworkMoveTo** object. This object is used to move the entity from its current location in the network to a new location. The new location is identified either directly as a node in the network or as the location of a certain seized resource. The first **NetworkMoveTo** object in the model will move the entities from their home node `entitySource` to the seized resource Pline which is the stations' location on the network i.e. each entity would move to its station. This step is not a real activity but rather a dummy step to distribute the entities, which are the stations, on the pipeline path. The properties of the object would be modified as follows:

Name: MoveToPline
Entity class: Entity
Destination is: Seized resource unit

Resource: Pline (The name for the resource pool of pipeline stations' location)

On exit: `count1++;` (a code that counts the number of entities that pass through the current object)

4.6) The next object to be implemented in the model is **Queue**. It is drawn from the *Enterprise* palette next to **NetworkMoveTo** object and a connector is drawn between the latter's right port and the left port of **Queue** object. A **Queue** is a buffer for entities waiting to be accepted by the next object in the process flow, or a general-purpose storage for the entities. As in the **NetworkSeize** object, entities inside the **Queue** are either arranged according to (FIFO) rule or based on their priority. The priority may be explicitly stored in the

entity or calculated based on the entity properties and external conditions. One of three exit ports set in the **Queue** object is used by entities to exit. The default port is **out** port which would be used if the next object allows for it. The second port **outPreempted** would be used in a priority-based queue. In a priority queue any incoming entity is always accepted, its priority evaluated and the entity is placed at the corresponding position in the queue. If the queue is full, the new entity may then cause the last entity to be thrown out of the queue via **outPreempted** port. If an entity is associated with a maximum waiting time, it will exit via the third port **outTimeout** will be used if the maximum waiting time is reached.

The current queue is used to stop all the entities that exit from **NetworkMoveTo** object when they reach their station from proceeding to the next phase of the model until the last entity reaches its station. The objective of this step is to give all the entities the same arrival time in the next object. This way they all have the same chance in seizing the resources needed for the first activity depending only on their priority. In order for the queue to hold the entities until they all arrive, the **Queue** is followed by another object which is **Hold**. The **Hold** object acts as a gate that can block the flow along a particular path. It will be closed until the number of entities inside the queue that precedes it reaches 10. Then, it will be opened through a java code written in the **Queue** object properties window as follows:

Name:	queue	(the default name)
Entity class:	Entity	
Capacity:	100	(any value above 10)
On enter:	<code>if (count1==10) hold.setBlocked(false)</code>	(a code that opens Hold object when number of entities reaches 10)

The properties of the **Hold** object are to be as follows:

Name:	hold	(the default name)
Entity class:	Entity	
Initially blocked:	checked	

The above steps would sum up the first stage of the model. In this stage, entities which represent the pipeline stations are created and assigned to their physical location on the network, which represents the layout of the pipeline construction site. The layout of objects in graphical editor will be as shown in Figure (5-8).

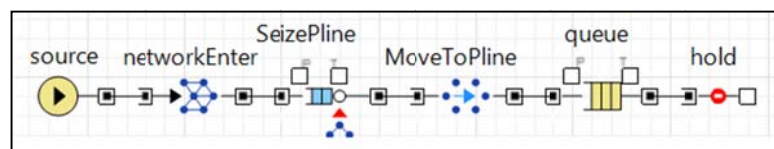


Figure 5-9: Sub-stage (2) – Creating Entities

Sub-stage (3): Modeling of “Pipe Stringing” Activity

This sub-stage of the model will represent the first activity of pipeline construction, Pipe Stringing, starting from seizing the pipe trucks to pick up the pipes from the pipe yard and sending them to the corresponding station until the side booms unload them from the trucks along the pipeline path.

5.1) The first object to be drawn from the *Enterprise* palette next to **Hold** object is **NetworkSeize** and is named “SeizePipes”. This object allows each entity to seize number of resource units, e.g. 10, from resource pool “Pipe” along with a “Side boom” resource. All the **NetworkSeize** objects from now on will be priority based. The entity’s priority was assigned to it in step (3.6) as a number from 1 to 10. As each entity enters “SeizePipes” object, it shall wait for its turn to acquire all the resources, whenever they are available, based on its priority e.g. if the entity with priority equal to 8 arrives first to the “SeizePipes” object, it shall remain in the queue waiting for the entities with priority equal to 10 and 9 to arrive and seize the needed resources before it is allowed to seize its resources, if they are still available. “SeizePipes” object’s properties are modified as follows:

Name:	SeizePipes	
Entity class:	Entity	
List of Resources:	{Sideboom, Pipe, Pipe, Pipe, Pipe, Pipe, Pipe, Pipe, Pipe, Pipe, Pipe, Pipe}	(The name for the resource pool of pipes and side booms)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity).StrngPriority	(The entity’s property on which the priority of queue is based)

5.2) The next object is also a **NetworkSeize** object. It is called “SeizeTrk”. This object allows entities to seize trucks to use them in transporting pipes from the pipe yard to the pipeline construction site. The queue of entities is arranged based on their priority *StrngPriority*. In addition, the object allows each entity to seize either 1 or 2 trucks based on a property called *TruckNo*. Each entity is assigned with a number, either 1 or 2, in step (3.6). Therefore, as each entity enters the “SeizeTrk” object, it takes its place in the queue based on its priority and when its turn comes, the entity seizes either 1 or 2 trucks based on the number assigned to its property *TruckNo*. In addition, the variable *time1* stores the time at which the entity with the highest priority i.e. (10) enters the “SeizeTrk” object. This variable will be used to calculate the total time of the construction process. Another action done by this object is to distribute the seized pipes on the trucks according to number of the latter. “SeizeTrk” object’s properties are modified as follows:

Name: SeizeTrk

Entity class: Entity

List of Resources: (The following code is used to decide whether to seize one or two trucks for the entity)

```
((Station)entity).TruckNo == 1 ? new NetworkResourcePool[] {
Truck } : new NetworkResourcePool[] { Truck, Truck }
```

On enter (The following code is executed as every entity enters the object. It is used to store the time at which the entity with priority equal to 10 enters the “SeizeTrk” object)

```
if (((Station)entity).StrngPriority == 10)
    time1=time();
```

On exit (The following code is executed as every entity exits the object. it is used to distribute the number of seized pipes on the number of trucks used)

```
if (((Station)entity).TruckNo == 1)
    ((Station)entity).PipeNo = 1;
else
    ((Station)entity).PipeNo = 2
```

Enable Preemption: Checked (An option that specifies the queue as priority based)

Entity priority: ((Station)entity).Strng Priority (The entity's property on which the priority of queue is based)

5.3) The next object **NetworkSendTo** is drawn from the *Enterprise* palette next to “SeizeTrk” object. This object is used to send network resources from their current location(s) to a new location in the network. It can only move moving resources such as trucks or portable resources such as pipes but they have to be escorted by moving resources. The resource units sent by this object may be at different locations. The entity will exit this object once the last unit arrives at the destination location, therefore the time spent by the entity in this object equals the longest travel time of the unit being sent. The speed of each group of units sent together equals the speed of the slowest moving resource in that group. The resource units will be animated moving along the shortest path from their origin to the destination. The current object, called “SendToPipeYard”, is used to send the trucks from the base camp to the pipe yard where the pipes would be loaded on it. “SendToPipeYard” object’s properties are modified as follows:

Name: SendToPipeYard

Entity class: Entity

Resources to send: (The following code is used to send the seized trucks whether they are one or two trucks)

```
((Station)entity).TruckNo == 1 ? new
NetworkResourcePool[] { Truck } : new
```

```
NetworkResourcePool[] { Truck, Truck }
```

Destination is: Seized resource unit (The option selected to identify the destination as the current location of a certain seized resource)

Resource: Pipe (The seized resource whose location is specified to send the resources to)

5.4) The next object is also a **NetworkSendTo** object. It is called “SendToSite”. It is used to send all the seized resources to their entity i.e. one or two trucks, one or two pipes and a side boom. “SendToSite” object’s properties are modified as follows:

Name: SendToSite

Entity class: Entity

Resources to send: (The following code is used to send the seized trucks whether they are one or two trucks)

```
((Station)entity).TruckNo == 1 ? new
NetworkResourcePool[] { Truck,Pipe,Sideboom } : new
NetworkResourcePool[] { Truck,Truck,Pipe,Pipe,Sideboom
}
```

Destination is Entity (The option selected to identify the destination as the current location of the entity that seizes the resources)

5.5) The next object in the model is **Delay**. It is drawn from the *Enterprise* palette next to “SendToSite” object. This object delays entities for a given amount of time. The delay time is may be stochastic and may depend on the entity as well as on any other conditions. Multiple entities, depending on the **Delay’s** capacity, can be delayed simultaneously and independently. Delay objects are used in this model to represent the actual time spent in executing the activity after all the resources are gathered together in the pipeline station. The current object, called “PipeStringing”, would represent the time of execution of pipes stringing activity. “PipeStringing” object’s properties are modified as follows:

Name: PipeStringing

Entity class: Entity

Delay time is Specified explicitly

Delay time: triangular(10, 12.5, 15) (The duration of the activity in hours takes a triangular distribution function)

5.6) The next object to be added to the model from the *Enterprise* palette next to “PipeStringing” object is **NetworkRelease**. This object is used to Releases all or some network resources previously seized by the entity. If a moving resource is released, there are

two options; it either returns to its home location or to stays where it is. However, after the resource is released, the network checks if the released resources have been requested by other entities and, if yes, the moving resource will be seized and not go to its home location regardless of the chosen option. A portable resource after its release will stay at its current location. If it needs to be returned to its home location, it should be moved either with the entity or with a seized moving resource. The current object "ReleasePipes" is used to release the seized one or two pipes, depending on number of trucks used for the entity, in the station's location in the network. "ReleasePipes" object's properties are modified as follows:

Name:	ReleasePipes	
Entity class:	Entity	
Release	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	(The following code is used to release the seized pipes whether they are one or two)	
	<pre>((Station)entity).TruckNo == 1 ? new NetworkResourcePool[] { Pipe } : new NetworkResourcePool[] { Pipe,Pipe }</pre>	
Moving resources:	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

5.7) The next object in the model is **SelectOutput**.This object helps in routing the incoming entities to one of two output ports depending on probabilistic or deterministic condition. The condition may depend on the entity as well as on any external factors. The two output ports are **OutTrue** and **OutFalse** and as each entity enters the object, according to its compliance to the condition, exits from one output port in zero time. The current object is used to decide whether the trucks should go back to the pipe yard to transport another batch of pipes to the station or not. The condition put in the object depends on each entity's property PipeNo which was set to 1 or 2 in step (3.12). A java code is executed when the entity exits through the **OutFalse** port to accumulate the number of delivered pipes in the entity's property PipeNo. The properties of the object are modified as follows:

Name:	SelectOutput	(The default name)
Select True output:	If the condition is true	(The option selected to specify the condition used)
Condition:	((Station)entity).PipeNo>=10	
On exit (false):	(The following code is used to count the number of pipes delivered to the entity and adds them to the entity's PipeNo)	
	<pre>((Station)entity).PipeNo=((Station)entity).PipeNo+((Station)entity).TruckNo*1)</pre>	

5.8) As for the **OutFalse** port of the **SelectOutput** added previously, the next object is a **NetworkSendTo** object called “SendToPipeYard1”. Similar to the object “SendToPipeYard” added in step (3.13), this object is used to return the trucks to the pipe yard to get another batch of pipes and transport them to site. Its properties are as follows:

Name: SendToPipeYard1

Resources to send: (The following code is used to send the seized trucks whether they are one or two trucks)

```
((Station)entity).TruckNo == 1 ? new
NetworkResourcePool[] { Truck } : new
NetworkResourcePool[] { Truck, Truck }
```

Destination is: Seized resource unit (The option selected to identify the destination as the current location of a certain seized resource)

Resource: Pipe (The seized resource whose location is specified to send the resources to)

5.9) Following “SendToPipeYard1” object, another **NetworkSendTo** object is added and named “SendToSite1”. Its properties and function are similar to those of “SendToSite” object added in step (3.14). This object is used to send both the trucks and pipes from the pipe yard to the construction site. The output port of the object is connected with the input port of the **Delay** object “PipeStringing” added in step (3.15) in order to repeat the Pipe Stringing activity once more. its properties are as follows:

Name: SendToSite1

Resources to send: (The following code is used to send the seized trucks whether they are one or two trucks)

```
((Station)entity).TruckNo == 1 ? new
NetworkResourcePool[] { Truck,Pipe } : new
NetworkResourcePool[] { Truck,Truck,Pipe,Pipe }
```

Destination is: Entity (The option selected to identify the destination as the current location of the entity that seizes the resources)

5.10) The entity shall pass through the objects “PipeStringing”, “ReleasePipes”, then the **OutFalse** port of **SelectOutput**, then “SendToPipeYard1” and finally “SendToSite1” for as many cycles as needed to fulfill the condition in **SelectOutput** object i.e. the number of pipes delivered to a certain station (entity) reaches ten pipes. In that case, the entity shall pass through **OutTrue** port. The next object, that the entity enters is a **NetworkRelease** object called “ReleaseTrk”. This object is connected to the **OutTrue** port; it is used to release the seized trucks to be sent to the base camp or seized by the entity next in the queue at “SeizeTrk” object in step (3.12). “ReleaseTrk” object’s properties are modified as follows:

Name: ReleaseTrk

Release: Specified resources (The option selected to specify that only the selected resources would be released)

List of Resources: (The following code is used to release the seized trucks whether they are one or two)

```
((Station)entity).TruckNo == 1 ? new
NetworkResourcePool[] { Truck } : new
NetworkResourcePool[] { Truck, Truck }
```

Moving resources: Return to home location (The option selected to specify where the released moving resources, if any, should go)

5.11) The last object in the “Pipe Stringing” activity is another **NetworkRelease** object. It is called “ReleaseSdboom1”. It releases the seized side booms and let them stay where they are waiting for the next entity to seize them. The object’s properties are modified as follows:

Name: ReleaseSdboom1

Release: Specified resources (The option selected to specify that only the selected resources would be released)

List of Resources: {Sideboom} (The name for the resource pool of side booms)

Moving resources: Stay where they are (The option selected to specify where the released moving resources, if any, should go)

The objects that constitute the “Pipe Stringing” activity are shown in figure (5-9).

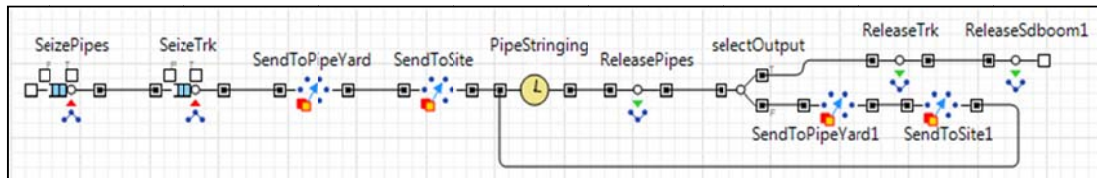


Figure 5-10: Sub-stage (3) - Pipe Stringing

Sub-stage (4): Modeling of “Pipe Bending” and “Welding” Activities

This sub-stage of the model represents the second and third activities of the project; they are Pipe Bending and Pipe Welding. Both activities follow the same sequence of objects to simulate them.

6.1) The first object in this stage is **Queue** object “queue1”. This object is used to store the entities with lower priority in “Pipe Bending” activity i.e. have `((Station)entity).BendPriority` less than 10 ,if they passed through previous objects, before they enter the next **NetworkSeize** object in order to let the entity with `BendPriority = 10` have the priority to seize the needed resources first. Entities entering “queue1” object are arranged based on their priority. A **Hold** object will be placed after “queue1” to block the flow until the entity with highest priority arrives. The condition to open **Hold** Object is placed in “queue1”. As each entity enters “queue1” object, its priority is

compared with a plain variable named “hold1Limit” which has an initial value of 10. If this entity has any priority less than 10, it exits from **outTimeout** port and return to input port of the **Queue**. Otherwise, the **Hold** object opens to pass this entity then closes again and “hold1Limit” value decreases by one. Then, the entity with priority equal to 9 takes its turn in this process and so on. This process is executed via a Java code put in “queue1”. Its properties are modified as follows:

Name:	queue1	
Capacity:	100	(any value above 10)
On at exit:	if (((Station) entity) .BendPriority == hold1Limit) hold1.setBlocked(false)	(a code that opens hold1 object when the entity with priority equals hold1Limit)
On exit:	hold1.setBlocked(true)	(a code that closes hold1 after the entity exits)
Enable exit on timeout:	Checked	(It allows the entity to exit through OutTimeout port)
Timeout:	1	(The maximum waiting time before the entity exits from OutTimeout port)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity Priority:	(((Station) entity) .BendPriority	(The entity’s property on which the priority of queue is based)

The properties of the **Hold** object are to be as follows:

Name:	Hold1	
On enter:	hold1Limit--;	(The value of “hold1Limit” is decreased by 1 as an entity enters)
Initially blocked:	checked	

6.2) The next object added to the model is a **NetworkSeize** object named “SeizeBndr”. It is used to seize a side boom and a pipe bending machine. Its properties are as follows:

Name:	SeizeBndr	
List of Resources:	{ Pipebender, Sideboom }	(The name for the resource pools of side booms and pipe bending machines)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	(((Station) entity) .BendPriority	(The entity’s property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	

6.3) After “SeizeBndr” object, a **Delay** object named “PipeBending” is added to the model. This object represents the time of execution of Pipe bending. The time depends on the topography of the pipeline station which is reflected in the number of pipe bends needed in each station. Therefore, the delay time is a function of entity’s property called BendNo. “PipeBending” object’s properties are as follows:

Name:	PipeBending	
Delay time is:	Specified explicitly	
Delay time:	$\text{triangular}(115, 125, 140) * ((\text{Station})\text{entity}).\text{BendNo} / 3$	(The duration of the activity in hours takes a triangular distribution function)

6.4) The last object of “Pipe Bending” activity is “ReleaseBndr” **NetworkRelease** object. It releases both the side boom and the pipe bending machine. Its properties are modified as follows:

Name:	ReleaseBndr	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{Pipebender, Sideboom}	(The name for the resource pools of side booms and pipe bending machines)
Moving resources	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

6.5) Pipe Welding activity are modeled in the same sequence of objects used for Pipe Bending activity (steps (3.22) to (3.25)). The entities shall pass through **Queue** object “queue2”, followed by **Hold** object “hold2” and two **NetworkSeize** objects “SeizeWldr” and “SeizeSdboom1”, then, a **Delay** object “PipeWelding” and finally two **NetworkRelease** objects “ReleaseWldr” and “ReleaseSdboom2”. “queue2” object’s properties are as follows:

Name:	queue2	
Capacity:	100	(any value above 10)
On at exit:	$\text{if } (((\text{Station})\text{entity}).\text{WeldPriority} == \text{hold2Limit}) \text{ hold2.setBlocked}(\text{false})$	(a code that opens “hold2” object when the entity with priority equals “hold2Limit”)
On exit:	hold2.setBlocked(true)	(a code that closes “hold1” after the entity exits)
Enable exit on timeout:	Checked	(It allows the entity to exit through OutTimeout port)
Timeout:	1	(The maximum waiting time before the entity exits from OutTimeout port)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)

Entity Priority:	((Station)entity).WeldPriority	(The entity's property on which the priority of queue is based)
-------------------------	--------------------------------	---

The properties of the **Hold** object are to be as follows:

Name:	hold2	
On enter:	Hold2Limit--;	(The value of "hold2Limit" is decreased by 1 as an entity enters)
Initially blocked:	checked	

The properties of "SeizeWldr" object are to be as follows:

Name:	SeizeWldr	
List of Resources:	(The following code is used to decide whether to seize one or two welders for the entity) <pre>((Station)entity).WelderNo == 1 ? new NetworkResourcePool[] { Welder } : new NetworkResourcePool[] { Welder, Welder }</pre>	
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity).WeldPriority	(The entity's property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	

The properties of "SeizeSdboom1" object are to be as follows:

Name:	SeizeSdboom1	
List of Resources:	(The following code is used to decide whether to seize one or two side booms for the entity) <pre>((Station)entity).WelderNo == 1 ? new NetworkResourcePool[] { Sideboom } : new NetworkResourcePool[] { Sideboom, Sideboom }</pre>	
On exit:	(The following code is executed as every entity exits the object. it is used to count the number of welded pipes depending on number of welders) <pre>if ((Station)entity).WelderNo == 1) ((Station)entity).WeldPipeNo = 1; else ((Station)entity).WeldPipeNo = 2</pre>	
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity).WeldPriority	(The entity's property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	

As for the **Delay** object "PipeWelding", the delay time is a function of number of welders i.e. the number of welded pipes. The properties of "PipeWelding" object are as follows:

Name:	PipeWelding	
Delay time is	Specified explicitly	
Delay time:	<code>triangular(42, 50, 60)</code> <code>/ ((Station)entity).WeldPipeNo</code>	(The duration of the activity in hours takes a triangular distribution function)

For the **NetworkRelease** object "ReleaseWldr", the properties are modified as follows:

Name:	ReleaseWldr	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	(The following code is used to decide whether to release one or two welders)	
	<pre>((Station)entity).WelderNo == 1 ? new NetworkResourcePool[] { Welder } : new NetworkResourcePool[] { Welder, Welder }</pre>	
Moving resources:	Return to home location	(The option selected to specify where the released moving resources, if any, should go)

And the last object in the sequence, **NetworkRelease** object "ReleaseSdboom2", has the same properties as "ReleaseSdboom1" in step (3.21) as follows:

Name:	ReleaseSdboom2	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{Sideboom}	(The name for the resource pool of side booms)
Moving resources	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

By adding this last object, this sub-stage of the model is finished and would appear in the graphical editor as shown in figure (5-10)

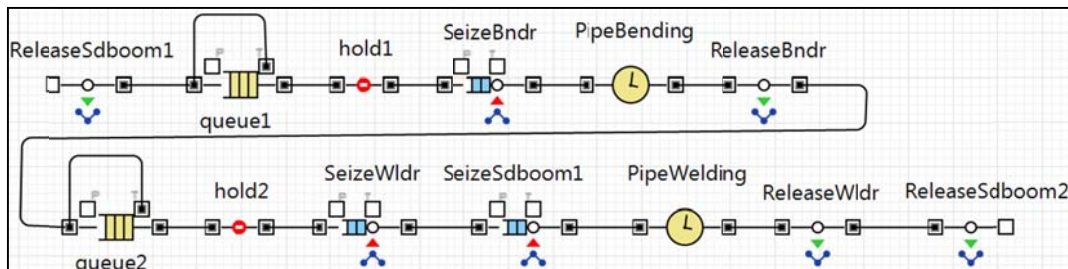


Figure 5-11: Sub-stage (4) - Pipe Bending and Welding Activities

Sub-stage (5): Modeling of “Weld Inspection” and “Weld Repair” Activities

This sub-stage of the model contains the sequence of objects that simulate the fourth and fifth activities of the project. These are Weld Inspection and Weld Repair. As weld inspection for the pipes of a certain station relies on finishing the welding activity for this station, any station, where welding activity is finished and regardless of its `WeldPriority`, shall seize “Inspector” resource first i.e. it will be “First In, First Out” (FIFO) based. However, if two entities requested the “Inspector” resource in the same time, `WeldPriority` of both entities will decide which gets it. This method will also be applied for “Weld Repair” activity.

7.1) The first object in this stage is a **NetworkSeize** object called “SeizeInspctr”. It is used to seize an inspector and sends him to the entity (station). Its properties are modified as follows:

Name:	SeizeInspctr	
List of Resources:	{Inspector}	(The name for the resource pool of inspectors)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity).WeldPriority	(The entity's property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	

7.2) The following object is “WeldInspecting” **Delay** object. It simulates the time taken to inspect the welded joints within one station. Its properties are as follows:

Name:	WeldInspecting	
Delay time is	Specified explicitly	
Delay time:	triangular(75, 83.3, 90)	(The duration of the activity in hours takes a triangular distribution function)

7.3) The **Delay** object is followed by **SelectOutput** object. Object is used to route the entities to one of two paths. The condition for this object is a probability condition. This means that passing entities shall exit through **OutTrue** or **OutFalse** output ports relies on a certain probability. In the current object, the probability condition equals 90% which is the average percent for welded joints passing the inspection successfully. Its properties are set as follows:

Name:	SelectOutput1	
Select True output:	With specified probability	(The option selected to specify the condition used)

Condition

7.4) The **OutTrue** port of “SelectOutput1” object is connected to a **NetworkRelease** object named “ReleaseInspctr”. Entities exiting through **OutTrue** port are the entities that passed the inspection; hence, the inspectors shall be released. “ReleaseInspctr” have the following properties:

Name:	ReleaseInspctr	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{ Inspector }	(The name for the resource pool of inspectors)
Moving resources	Return to home location	(The option selected to specify where the released moving resources, if any, should go)

7.5) The stations, that do not pass the inspection, require welders to repair the welded joints and an inspector to re-inspect the repaired welds. Thus, the OutFalse output port of “SelectOutput1” object is connected to a **NetworkRelease** object named “ReleaseInspctr1”. Its properties are similar to “ReleaseInspctr” object created in the previous step. After that, a **NetworkSeize** object is placed. It is used to seize a side boom and a welder to work on “Weld Repairing”. Its properties is modified as follows:

Name:	SeizeWldrSdboom	
List of Resources:	{ Sideboom, Welder }	(The name for the resource pool of side booms and welders)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity). WeldPriority	(The entity’s property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	

7.6) After seizing the required resources, the entities shall pass through a **Delay** object named “WeldingRpr”. The delay time of this object takes a triangular distribution. Its properties are:

Name:	WeldRpr	
Delay time is	Specified explicitly	
Delay time:	triangular(42, 50, 60)	(The duration of the activity in hours takes a triangular distribution function)

7.7) After finishing the weld repair, the entity releases both the side boom and the welder. This shall be done through two **NetworkRelease** objects “ReleaseWldr1” and “ReleaseSdboom3”. “ReleaseWldr1” object has the following properties:

Name:	ReleaseWldr1	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{Welder}	(The name for the resource pool of welders)
Moving resources	Return to home location	(The option selected to specify where the released moving resources, if any, should go)

While “ReleaseSdboom3” has the following properties:

Name:	ReleaseSdboom3	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{Sideboom}	(The name for the resource pool of side booms)
Moving resources	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

7.8) Each entity need to seize the inspector once more to repeat the inspection for repaired welded joints. This needs adding “SeizeInspctr1” and “WeldInspecting1” objects to the sequence similar to those in steps (3.27) and (3.28). however, for “WeldInspecting1” object, the delay time would be as follows:

Name:	WeldInspecting1	
Delay time is	Specified explicitly	
Delay time:	triangular(7.5, 8.33, 9)	(The duration of the activity in hours takes a triangular distribution function)

7.9) The last object in this stage is “SelectOutput2” object. This object represents the probability of passing the inspection for the repaired joints. The **OutFalse** port shall be connected back with input port of “ReleaseInspctr1” (see step (3.31)) to repeat the process of weld repair. On the other hand, the **OutTrue** port is connected to “ReleaseInspctr” added in step (3.30) i.e. entities exiting through it shall proceed to the next stage. The probability used in “SelectOutput2” is as follows:

Name:	SelectOutput2	
Select True output:	With specified probability	(The option selected to specify the condition used)
Condition	0.9	

The sequence of objects in this sub-stage is shown in figure (5.11).

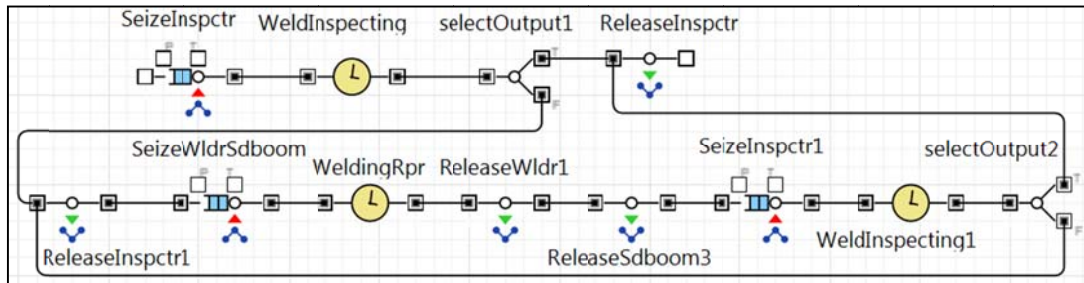


Figure 5-12: Sub-stage (5) - Weld Inspection and Repair Activities

Sub-stage (6): Modeling of “Joints Coating”, “Excavation” and “Pipe Lowering” Activities

This sub-stage contains the sequence of objects that simulate three activities. These activities are Coating of Welded Joints, Pipeline Trench Excavation and Lowering of Pipes into the Trench. The sequence of objects for each of the three activities is the same. The entities shall pass through a **Queue** object, followed by a **Hold** object and a **NetworkSeize** object, then, a **Delay** object and finally a **NetworkRelease** object.

8.1) For the first activity, Joints Coating, the entities (stations) will be held in a priority based **Queue** “queue3” based on their `CoatPriority` value. The **Hold** object “hold3” remains closed until the entity with `CoatPriority` value equal to 10 arrives to the queue. Only then, the **Hold** object will open for this entity to pass and closes again waiting for the entity with `CoatPriority` value equal to 9 to arrive to the queue and so on (see step (3.22)). The entity, which passes through the “hold3”, enters “SeizeCoating” **NetworkSeize** object and seizes a “CoatingTeam” Resource, if available, and sends the resource to the station’s location. Once the resource arrives to its station, the entity exits “SeizeCoating” object and enters the **Delay** object “Coating” to spend the time of execution of the activity. Finally, the entity releases the “CoatingTeam” Resource at “ReleaseCoating” **NetworkRelease** object and sends it to its home location or a new station. The properties of these objects are as follows:

Name:	queue3	
Capacity:	100	(any value above 10)
On at exit:	if (((Station)entity).CoatPriority == hold3Limit) hold3.setBlocked(false)	(a code that opens “hold3” object when the entity with priority equals “hold3Limit”)
On exit:	hold3.setBlocked(true)	(a code that closes “hold3” after the entity exits)
Enable exit on timeout:	Checked	(It allows the entity to exit through OutTimeout port)
Timeout:	1	(The maximum waiting time before the entity exits from OutTimeout port)

Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity Priority:	((Station)entity) . CoatPriority	(The entity's property on which the priority of queue is based)
Name:	hold3	
On enter:	hold3Limit--;	(The value of "hold3Limit" is decreased by 1 as an entity enters)
Initially blocked:	checked	
Name:	SeizeCoating	
List of Resources:	{CoatingTeam}	(The name for the resource pool of coating teams)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity) .Coat Priority	(The entity's property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	
Name:	Coating	
Delay time is	Specified explicitly	
Delay time:	triangular(22, 25, 28)	(The duration of the activity in hours takes a triangular distribution function)
Name:	ReleaseCoating	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{CoatingTeam}	(The following code is used to release the seized pipes whether they are one or two)
Moving resources	Return to home location	(The option selected to specify where the released moving resources, if any, should go)

8.2) In the second activity, Pipeline Trench Excavation, the same sequence of objects is followed. The entities (stations) are held in a priority based **Queue** "queue4" based on their `ExcavPriority` value. The **Hold** object "hold4" remains closed until the entity with `ExcavPriority` value equal to 10 arrives to the queue. Only then, the **Hold** object will open for this entity to pass and closes again waiting for the entity with `ExcavPriority` value equal to 9 to arrive to the queue and so on (see step (3.22)). The entity, which passes through the "hold4", enters "SeizeExcavtr" **NetworkSeize** object and seizes one or two units of "Excavator" Resource based on the entity's property `ExcavNo` and sends the resource to the station's location. Once the resource arrives to its station, the entity exits "SeizeExcavtr" object

and enters the **Delay** object “Excavation” to spend the time of execution of the activity. The delay time has a triangular distribution and is a function in the number of seized excavators along with another property of the entity which is *Excvdiff*. This property stands for the soil type of each station which may vary from loose sand to hard clay. Therefore, *Excvdiff* value will increase as the hardness of soil increases to affect the excavation duration. Finally, the entity releases the “Excavator” Resource at “ReleaseExcvt” **NetworkRelease** object and sends it to its home location or a new station. The properties of these objects are as follows:

Name:	queue4	
Capacity:	100	(any value above 10)
On at exit:	if ((Station)entity).ExcvtPriority == hold4Limit) hold4.setBlocked(false)	(a code that opens “hold4” object when the entity with priority equals “hold4Limit”)
On exit:	hold4.setBlocked(true)	(a code that closes “hold4” after the entity exits)
Enable exit on timeout:	Checked	(It allows the entity to exit through OutTimeout port)
Timeout:	1	(The maximum waiting time before the entity exits from OutTimeout port)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity Priority:	((Station)entity). ExcvtPriority	(The entity’s property on which the priority of queue is based)

Name:	hold4	
On enter:	Hold4Limit--; checked	(The value of “hold4Limit” is decreased by 1 as an entity enters)
Initially blocked:		

Name:	SeizeExcvt	
List of Resources:	(The following code is used to decide whether to seize one or two excavators)	
	<pre>((Station)entity).ExcvtNo == 1 ? new NetworkResourcePool[] { Excavator } : new NetworkResourcePool[] { Excavator, Excavator }</pre>	
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity).Excvt Priority	(The entity’s property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)

Destination is:	Entity	
Name:	Excavation	
Delay time is	Specified explicitly	
Delay time:	<code>((Station)entity).Excvdiff/2 *triangular(250, 300, 330)/((Station)entity).ExcvNo</code>	(The duration of the activity in hours takes a triangular distribution function)
Name:	ReleaseExcvtr	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	(The following code is used to decide whether to release one or two excavators)	
	<pre>((Station)entity).ExcvNo == 1 ? new NetworkResourcePool[] { Excavator } : new NetworkResourcePool[] { Excavator, Excavator }</pre>	
Moving resources	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

8.3) The third activity that follows the same sequence of objects is Lowering of Pipes. The entities (stations) are held in a priority based **Queue** “queue5” based on their `LwrPriority` value. The **Hold** object “hold5” remains closed until the entity with `LwrPriority` value equal to 10 arrives to the queue. Only then, the **Hold** object will open for this entity to pass and closes again waiting for the entity with `LwrPriority` value equal to 9 to arrive to the queue and so on (see step (3.22)). The entity, which passes through the “hold5”, enters “SeizeSdboom” **NetworkSeize** object and seizes five units of “Sideboom” Resource and sends the resource to the station’s location. Once the resource arrives to its station, the entity exits “SeizeSdboom” object and enters the **Delay** object “PipeLowering” to spend the time of execution of the activity. Finally, the entity releases the “Sideboom” Resource at “ReleaseSdboom4” **NetworkRelease** object and sends it to its home location or a new station. The properties of these objects are as follows:

Name:	queue5	
Capacity:	100	(any value above 10)
On at exit:	<pre>if (((Station)entity).LwrPriority == hold5Limit) hold5.setBlocked(false)</pre>	(a code that opens “hold5” object when the entity with priority equals “hold5Limit”)
On exit:	<code>hold5.setBlocked(true)</code>	(a code that closes “hold5” after the entity exits)
Enable exit on timeout:	Checked	(It allows the entity to exit through OutTimeout port)
Timeout:	1	(The maximum waiting time before the entity exits from OutTimeout port)

Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity Priority:	((Station)entity) . LwrPriority	(The entity's property on which the priority of queue is based)
<hr/>		
Name:	hold5	
On enter:	hold5Limit--;	(The value of "hold5Limit" is decreased by 1 as an entity enters)
Initially blocked:	checked	
<hr/>		
Name:	SeizeSdboom	
List of Resources:	{Sideboom, Sideboom, Sideboom, Sideboom, Sideboom}	(The name for the resource pool of side booms)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity) . LwrPriority	(The entity's property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	
<hr/>		
Name:	PipeLowering	
Delay time is	Specified explicitly	
Delay time:	triangular(13, 15, 18)	(The duration of the activity in hours takes a triangular distribution function)
<hr/>		
Name:	ReleaseSdboom4	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{Sideboom, Sideboom, Sideboom, Sideboom, Sideboom}	(The name for the resource pool of side booms)
Moving resources	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

The sequence of objects in this sub-stage is shown in figure (5.12).

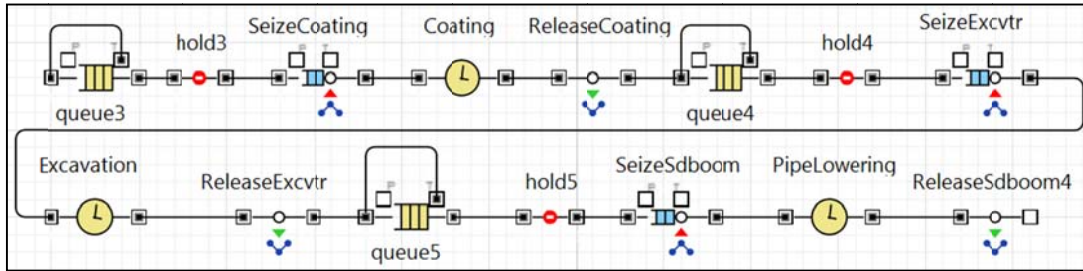


Figure 5-13: Sub-stage (6) - Joints Coating, Trench Excavation and Pipe Lowering Activities

Sub-stage (7): Modeling of “Joint Welding” and “Weld Inspection” Activities

This sub-stage contains the sequence of objects that simulate two activities. These activities are Joints welding and Weld inspection. Both activities have the same sequence of objects. Entities seize the needed resources through a **NetworkSeize** object, then, they spend the duration of the activity in a **Delay** object. Finally, they release the seized resources via a **NetworkRelease** object.

For “Joints Welding” activity, entities seize a welder and a side boom in “SeizeWldrSdboom1” object. Similar to “Joint Inspecting” activity in step (3.27), any station, where lowering activity is finished and regardless of its *WeldPriority*, shall seize “Welder” resource first i.e. it will be “First In, First Out” (FIFO) based. However, if two entities requested the “Welder” resource in the same time, *WeldPriority* of both entities will decide which one gets it. This method will also be applied for “Weld Inspection” activity in the next step. Next, entities enter **Delay** object “PipeWelding1” for the duration of the activity which takes a triangular distribution. Finally, entities release the seized resource “Welder” and send it to its home location by “ReleaseWldr” object while “ReleaseSdboom5” object is used to release “Sideboom” resource and keeps it where it is waiting for the next seize. The properties of these objects are as follows:

Name:	SeizeWldrSdboom1	
List of Resources:	{Sideboom,Welder}	(The name for the resource pool of side booms and welders)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity) . WeldPriority	(The entity’s property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	

Name: PipeWelding1

Delay time is	Specified explicitly	
Delay time:	triangular(8, 10, 12)	(The duration of the activity in hours takes a triangular distribution function)

Name:	ReleaseWldr2	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{Welder}	(The name for the resource pool of welders)
Moving resources	Return to home location	(The option selected to specify where the released moving resources, if any, should go)

Name:	ReleaseSdboom5	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{Sideboom}	(The name for the resource pool of side booms)
Moving resources	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

The second activity, Weld Inspection, follows the same sequence of objects. Each entity shall enter a **NetworkSeize** object “SeizeInspctr2” to seize an “Inspector” resource. When he arrives to the station location, the entity enters the **Delay** object “WeldInspecting2” as its delay time represents the duration of inspection activity. Finally, the entity releases the seized resource and sends it to its home location via **NetworkRelease** object “ReleaseInspctr2”. Figure (5-13) shows the sequence of objects of this stage. The properties of these objects are as follows:

Name:	SeizeInspctr2	
List of Resources:	{Inspector}	(The name for the resource pool of inspectors)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity) . WeldPriority	(The entity's property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	

Name:	WeldInspecting2	
--------------	-----------------	--

Delay time is	Specified explicitly	
Delay time:	triangular(1.5, 1.67, 1.9)	(The duration of the activity in hours takes a triangular distribution function)

Name:	ReleaseInspctr2	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{Inspector}	(The name for the resource pool of welders)
Moving resources	Return to home location	(The option selected to specify where the released moving resources, if any, should go)

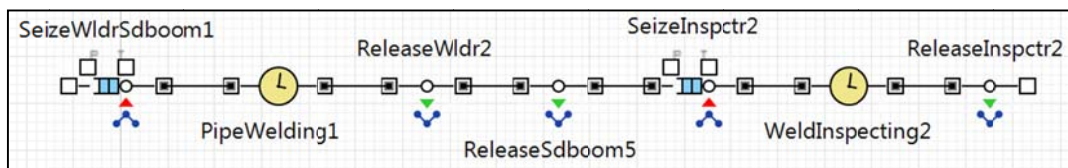


Figure 5-14: Sub-stage (7) – Joint Welding and Weld Inspection Activities

Sub-stage (8): Modeling of “Trench Backfilling” and “Hydro-testing” Activities

This is the last sub-stage of the stage of creating the model diagram. It contains the sequence of objects that simulate the last two activities in the pipeline construction project; they are Pipeline Trench Backfilling and Hydro testing of Pipeline. Both activities have a sequence of objects similar to that of sub-stage (6) activities. The entities shall pass through a **Queue** object, followed by a **Hold** object and a **NetworkSeize** object, then, a **Delay** object and finally a **NetworkRelease** object.

For the first activity, Trench Backfilling, the entities (stations) will be held in a priority based **Queue** “queue6” based on their `BckflPriority` value. The **Hold** object “hold6” remains closed until the entity with `BckflPriority` value equal to 10 arrives to the queue. Only then, the **Hold** object will open for this entity to pass and closes again waiting for the entity with `BckflPriority` value equal to 9 to arrive to the queue and so on (see step (3.22)). The entity, which passes through “hold6”, enters “SeizeBulldzr” **NetworkSeize** object and seizes one or two units of “Bulldozer” resource based on the entity’s property `BldzrNo` and sends the resource to the station’s location. Once the resource arrives to its station, the entity exits “SeizeBulldzr” object and enters the **Delay** object “Backfilling” to spend the time of execution of the activity. Finally, the entity releases the “Bulldozer” Resource at “ReleaseBulldzr” **NetworkRelease** object and sends it to its home location or a new station. The properties of these objects are as follows:

Name:	queue6	
Capacity:	100	(any value above 10)
On at exit:	if ((Station)entity).BckflPriority == hold6Limit) hold6.setBlocked(false)	(a code that opens "hold6" object when the entity with priority equals "hold6Limit")
On exit:	hold6.setBlocked(true)	(a code that closes "hold6" after the entity exits)
Enable exit on timeout:	Checked	(It allows the entity to exit through OutTimeout port)
Timeout:	1	(The maximum waiting time before the entity exits from OutTimeout port)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity Priority:	((Station)entity).BckflPriority	(The entity's property on which the priority of queue is based)

Name:	hold6	
On enter:	hold6Limit--;	(The value of "hold6Limit" is decreased by 1 as an entity enters)
Initially blocked:	checked	

Name:	SeizeBulldzr	
List of Resources:	(The following code is used to decide whether to seize one or two bulldozers)	
	<pre>((Station)entity).BldzrNo == 1 ? new NetworkResourcePool[] { Bulldozer } : new NetworkResourcePool[] { Bulldozer, Bulldozer }</pre>	
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity).BckflPriority	(The entity's property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	

Name:	Backfilling	
Delay time is	Specified explicitly	
Delay time:	triangular(160, 180, 200) / ((Station)entity).BldzrNo	(The duration of the activity in hours takes a triangular distribution function)

Name: ReleaseBulldzr

Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	(The following code is used to decide whether to release one or two bulldozers)	
<pre>((Station)entity).BldzrNo == 1 ? new NetworkResourcePool[] { Bulldozer } : new NetworkResourcePool[] { Bulldozer, Bulldozer }</pre>		
Moving resources:	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

In the second activity, Hydro testing of Pipeline, the same sequence of objects is followed. The entities (stations) are held in a priority based **Queue** “queue7” based on their `HdrtstPriority` value. The **Hold** object “hold7” remains closed until the entity with `HdrtstPriority` value equal to 10 arrives to the queue. Only then, the **Hold** object will open for this entity to pass and closes again waiting for the entity with `HdrtstPriority` value equal to 9 to arrive to the queue and so on (see step (3.22)). The entity, which passes through “hold7”, enters “SeizeHydtest” **NetworkSeize** object and seizes a “Hydro-testing team” Resource, if available, and sends the resource to the station’s location. Once the resource arrives to its station, the entity exits “SeizeHydtest” object and enters the **Delay** object “HydroTesting” to spend the time of execution of the activity. Finally, the entity releases the “Hydro-testing team” Resource at “ReleaseHydtest” **NetworkRelease** object and sends it to its home location or a new station. In addition, the variable `time2` stores the time at which the entity with the lowest priority i.e. (1) enters the “ReleaseHydtest” object. This variable along with `time1` variable in step (3.12) will be used to calculate the total time of the construction process. The properties of these objects are as follows:

Name:	queue7	
Capacity:	100	(any value above 10)
On at exit:	<pre>if (((Station)entity).HdrtstPri ority == hold7Limit) hold7.setBlocked(false)</pre>	(a code that opens hold7 object when the entity with priority equals hold7Limit)
On exit:	<code>hold7.setBlocked(true)</code>	(a code that closes hold7 after the entity exits)
Enable exit on timeout:	Checked	(It allows the entity to exit through OutTimeout port)
Timeout:	1	(The maximum waiting time before the entity exits from OutTimeout port)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity Priority:	<code>((Station)entity).HdrtstPriority</code>	(The entity’s property on which the priority of queue is based)

Name:	hold7	
On enter:	hold7Limit--;	(The value of "hold7Limit" is decreased by 1 as an entity enters)
Initially blocked:	checked	
Name:	SeizeHydtest	
List of Resources:	{HydTestTeam}	(The name for the resource pool of hydro-testing teams)
Enable Preemption:	Checked	(An option that specifies the queue as priority based)
Entity priority:	((Station)entity).HdrtstPriority	(The entity's property on which the priority of queue is based)
Send seized resources:	Checked	(An option that sends the seized resources to a given destination)
Destination is:	Entity	
Name:	HydroTesting	
Delay time is	Specified explicitly	
Delay time:	triangular(45, 50, 55)	(The duration of the activity in hours takes a triangular distribution function)
Name:	ReleaseHydtest	
Release:	Specified resources	(The option selected to specify that only the selected resources would be released)
List of Resources:	{HydTestTeam}	(The following code is used to release the seized pipes whether they are one or two)
Moving resources	Return to home location	(The option selected to specify where the released moving resources, if any, should go)
On enter	(The following code is executed as every entity enters the object. It is used to store the time at which the entity with priority equal to 1 enters the object)	
	<pre> if (((Station)entity).HdrtstPriority == 1) time2=time(); time=time2-time1; </pre>	

As all the activities are finished in all stations, each entity must release any resources seized by it. At this point, the only remaining seized resource is "Pline" resource, which was seized in step (3.8). However, another option in the **NetworkRelease** object, called "ReleasePline", is used; this option is to release all seized resources whatever they are. As a result, the object's properties is modified as follows:

Name:	ReleasePline	
Release:	All seized resources	(The option selected to release all resources)
Moving resources	Stay where they are	(The option selected to specify where the released moving resources, if any, should go)

In order to finalize the simulation process, two objects must be added after the last object. The first object is **NetworkExit**. This object unregisters the entity from the network. The entity will also no longer be animated by the network. The second object is **Sink**. This object is considered the end point of the model. It is used to dispose entities. Both objects' default properties would be used. Figure (5-14) shows the sequence of objects of the last stage.

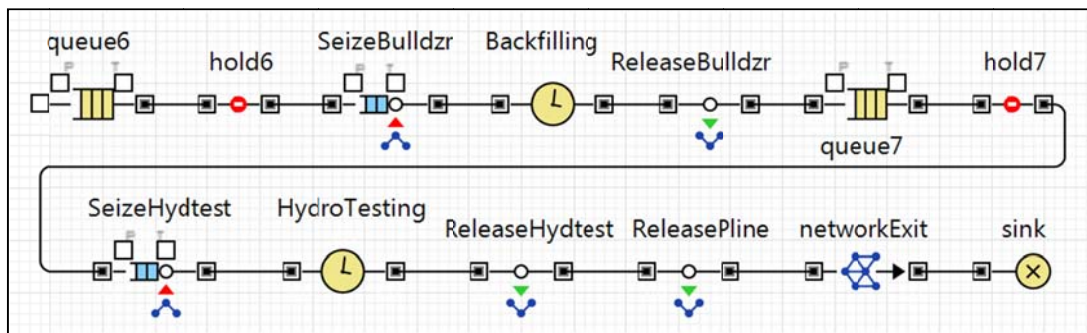


Figure 5-15: Sub-stage (8) - Trench Backfilling and Hydro-testing Activities

The final step to finish the model is to connect all stages together to get the layout of objects shown in figure (5-7).

5.3.4. Stage (4): Creating the “Simulation” experiment

The fourth main stage is creating the “Simulation” experiment which runs model simulation with animation displayed and model debugging enabled. The first experiment in each model is automatically created. Therefore, no steps would be presented to its development. The fifth main stage which is creating “Optimization” experiment will be demonstrated in detail in next chapter.

5.4. Summary and Conclusion

In this chapter, the steps of constructing the simulation model needed for this research were presented. These steps' purpose was managing the positions and properties of different objects constituting the model, defining relationships between them and creating an animated presentation to the whole process. Dealing with the simulation software “AnyLogic” included interacting with graphic view as well as Java code lines. The resulting model could be used

individually to simulate a typical pipeline project, modify any number of variables and observe the resulting change in project's total duration.

CHAPTER 6

OPTIMIZATION MODEL

6.1. Introduction

In the previous chapter, the simulation model was created using various objects from the enterprise library, introducing a number of Java code lines and creating a graphical network that resembles the project's landscape. The model employs Discrete Event Simulation to simulate the activities of pipeline construction starting from pipe stringing all the way to Hydro-testing of pipeline. It shows the real time sequence of events that the project would run through to finish all activities for all stations using the needed resources, based on their availability, and maintaining different relationships between the activities.

In this chapter, the final step of constructing the model is demonstrated. The optimal schedule for the project and the number of resources utilized in each station for each activity shall be obtained by an "Optimization Experiment". An optimization experiment is one of many experiments, available in AnyLogic, which can be done on the model. As stated previously, an optimization experiment is the process of finding the best possible solution for a certain problem by getting the optimal combination of conditions that affect the result.

AnyLogic optimization process is built using an optimization engine called **OptQuest**. The **OptQuest** Engine automatically finds the best parameters of a model, with respect to certain constraints. The optimization process consists of repetitive simulations of a model with different parameters. Using sophisticated algorithms, the **OptQuest** Engine varies controllable parameters from simulation to simulation to find the optimal parameters for solving a problem. In addition, AnyLogic provides a convenient graphical user interface to set up and control the optimization. The optimization experiment in AnyLogic relies on defining several factors such as the objective function, which needs to be maximized or minimized, the optimization parameters and the optimization constraints. The steps of defining these factors will be demonstrated later.

In the final part of this chapter and after finishing the definition of the optimization problem, a case study will be illustrated to show the applicability of the simulation model and how much does it resemble the real process. The schedule generated for the construction of South Valley (Ganoub Elwadi) Gas pipeline would be presented. This case study was used primarily to assist in planning, testing and validating the model and its functions. This chapter also presents three stages of model runs and evaluation of their results.

6.2. Optimization Experiment

Constructing an optimization experiment is done using the same user interface used for the simulation model (see figure (5-1)). However, unlike the simulation model, optimization

experiment doesn't need much effort on defining the different elements it relies on. This is the result of AnyLogic employing an optimization engine. This engine has within it all the needed procedures to run the simulation numerous times, while altering the values for a number of parameters, until it reach the optimal solution for the objective function. Thus, the major part of creating the optimization experiment is done through modifying its various properties in the *properties view*. In the different tabs shown in figure (6-1) all the elements needed to define the experiment. As for the graphical editor, all the objects inserted in it would be to illustrate any sort of input used or output resulting from the experiment such as graphs or charts. In addition, some additions and modifications will be done inside the main active object **Main** which contains the model.

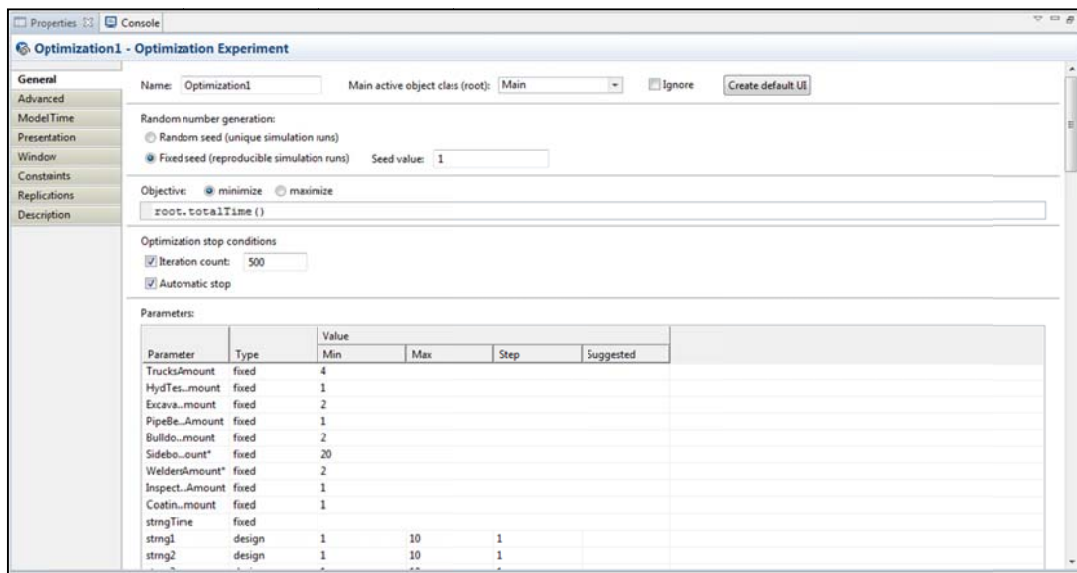


Figure 6-1: Properties View of Optimization Model

Creating the experiment shall undergo four main stages:

- 1- Defining the objective function
- 2- Defining the optimization parameters
- 3- Defining the optimization constraints
- 4- Forming the graphical presentation for the experiment.

The first stage after adding a new optimization experiment, since it is not added to each new model by default as in the case of simulation experiment, is to define the objective function. It is defined in the first page of *properties view* which is *General* page. The objective function which is to minimize the total time of construction process shall be defined according to the following steps:

1.1) A new function is created in the main model. The **function** object is drawn from the *General* palette into the graphical editor. This function is used to get the

value of variable `time` which is calculated in the model in step (3.42) of the previous chapter. The properties of this object are as follows:

Name: `totalTime`

Return type: `double`

Function body (found in “Code” Tab): `return time;` (The code is used to get the value of time variable)

- 1.2) Returning to optimization window, the objective function i.e. minimize **totalTime** is set in General page of the experiment’s properties as shown in figure (6-2).

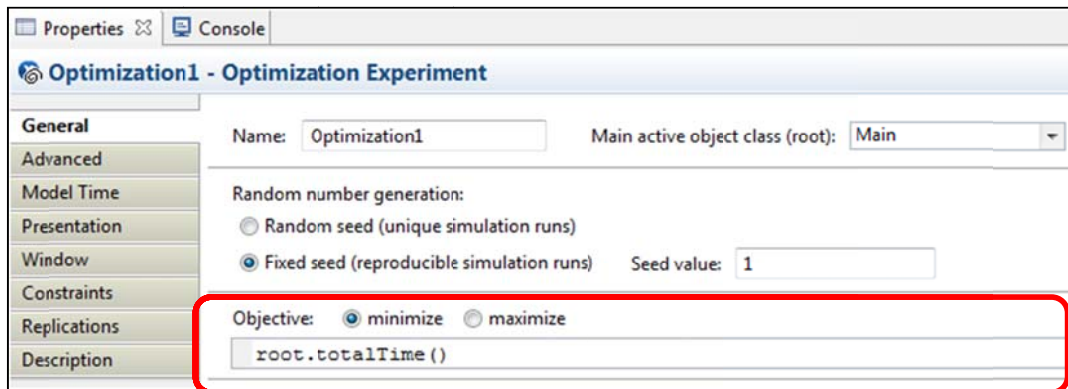


Figure 6-2: Setting the Optimization Objective

The second stage is to define the optimization parameters. The **OptQuest** Engine searches through possible values of these parameters to find the optimal parameters.

During the optimization process, the parameter's value is changed within an interval, which is defined by lower and upper bounds, according to its type. An optimization parameter type could be one of the following:

- Continuous parameter
- Discrete parameter
- Design parameter

Continuous parameter can take any value from the interval. The parameter precision determines the minimum value that continuous parameters can change. Discrete parameter is represented by a finite set of decisions with essential direction i.e. it can take values from the specified set only. It begins at a lower bound and increments by a specified step size up to an upper bound. On the other side, Design parameter is also represented by a finite set of decisions but there is no clear sense of direction. It begins at a lower bound and increments by a step size up to an upper bound. However, Values order is non-sequential. As each optimization parameter is defined, its type is chosen according to its nature and influence on the objective function. for example, any type of rate or ratio should be a continuous parameter, while, number of a specific resource should be discrete or design.

As illustrated in previous chapters, the optimization parameters for the model are the sequence of work in pipeline stations for each activity and the number of utilized resources in each station for each activity. The sequence for each activity was defined in step (3.7) in the previous chapter as an array of ten values. Each entity as it enters the network has a value assigned from each array to its corresponding entity's property. Among these properties is the entity's priority to seize the resources needed for each activity. The default values for the arrays of priorities of all activities are {1, 2, 3..... 9, 10} i.e. the first entity takes priority=1 and the second takes priority =2 and so on. In order to incorporate the arrays as optimization parameters, each element of each array shall be a separate optimization parameter that takes any value from 1 to 10. Accordingly, each array would be changed to {parameter1, parameter2, parameter3....., parameter9, parameter10} where parameter1 ≠ parameter2 ≠ parameter3 parameter9 ≠ parameter10. The resulting arrays after the optimization process would be the optimum sequence. The arrays of number of resources used for each activity such as number of trucks (TruckNo) and number of excavators (ExcavNo) will also be optimization parameters. Each element of the array shall take a value between one and the maximum number of units of the resource. The following steps to define the optimization parameters would be undertaken in the same order for each array:

2.1) The **Parameter** object is drawn from the *General* palette into the graphical editor in the main model. The object would be named `strng1` and it represents the first element in "stationProp1" array (see step (3.7) in chapter 5) which is used to assign entities' `StrngPriority`. `Strng1` indicate the priority assigned for first entity or station. the following properties are assigned to it:

Name: `strng1`
Type: `int` (The parameter value shall be integer number)

2.2) The previous step is repeated to create `strng2`, `strng3`, `strng4`..... `strng9` & `strng10`.

2.3) All ten parameters `strng1` to `strng10` are placed in "stationProps1" array. Its properties would be modified as follows:

Table 6-1: Properties of "stationProps1"

Name:	<code>stationProps1</code>	Array for <code>StrngPriority</code>
Type:	Other: <code>int []</code>	
Initial Value:	<code>new int [] {strng1, strng2, strng3, strng4, strng5, strng6, strng7, strng8, strng9, strng10}</code>	

2.4) Returning to optimization window, in General page of the experiment's properties go to the row of the Parameters table containing the parameter `strng1`. In the **Type** field, the

design type is chosen. The range for the parameter is specified where the parameter's lower bound is entered in the **Min** field and the parameter's upper bound in the **Max** field. The parameter step value is specified in the **Step** field. The values entered for the parameter is as follows:

Table 6-2: Values of Optimization Parameter “Strng1”

Parameter	Type	Value		
		Min	Max	Step
Strng1	design	1	10	1

The same values are entered for `strng2` to `strng10` to be as shown in figure (6-3).

Optimization1 - Optimization Experiment

Name: Optimization1 Main active object class (root): Main ☐ Ignore

Random number generation:
☐ Random seed (unique simulation runs)
☒ Fixed seed (reproducible simulation runs) Seed value: 1

Objective: ☒ minimize ☐ maximize
 root.totalTime()

Optimization step conditions
☒ Iteration count: 500
☒ Automatic stop

Parameters:

Parameter	Type	Value	Min	Max	Step	Suggested
strng1	design	1	1	10	1	
strng2	design	1	1	10	1	
strng3	design	1	1	10	1	
strng4	design	1	1	10	1	
strng5	design	1	1	10	1	
strng6	design	1	1	10	1	
strng7	design	1	1	10	1	
strng8	design	1	1	10	1	
strng9	design	1	1	10	1	
strng10	design	1	1	10	1	

Figure 6-3: Defining Optimization Parameters

2.5) Steps (2.1) & (2.2) are repeated to create the remaining optimization parameters in arrays “stationProp2”, “stationProp3”, “stationProp5”, “stationProp6”, “stationProp7”, “stationProp9”, “stationProp10”, “stationProp11”, “stationProp12”, “stationProp13” and “stationProp14”. They would contain the following optimization parameters:

Table 6-3: Properties of Arrays “stationProps2” to “stationProps14”

Name:	stationProps2	Array for TruckNo
Type:	Other: int []	
Initial Value:	new int [] {trucks1,trucks2,trucks3,trucks4,t rucks5,trucks6,trucks7,trucks8,tru cks9,trucks10}	
Name:	stationProps3	Array for BendPriority
Type:	Other: int []	
Initial Value:	new int [] {bend1,bend2,bend3,bend4,bend5,ben d6,bend7,bend8,bend9,bend10}	
Name:	stationProps5	Array for WeldPriority
Type:	Other: int []	
Initial Value:	new int [] {weld1,weld2,weld3,weld4,weld5,wel d6,weld7,weld8,weld9,weld10}	
Name:	stationProps6	Array for WelderNo
Type:	Other: int []	
Initial Value:	new int [] {welder1,welder2,welder3,welder4,w elder5,welder6,welder7,welder8,wel der9,welder10}	
Name:	stationProps7	Array for CoatPriority
Type:	Other: int []	
Initial Value:	new int [] {coat1,coat2,coat3,coat4,coat5,coa t6,coat7,coat8,coat9,coat10}	
Name:	stationProps9	Array for ExcvPriority
Type:	Other: int []	
Initial Value:	new int [] {excv1,excv2,excv3,excv4,excv5,exc v6,excv7,excv8,excv9,excv10}	
Name:	stationProps10	Array for ExcvNo
Type:	Other: int []	
Initial Value:	new int [] {excvtr1,excvtr2,excvtr3,excvtr4,e xcvtr5,excvtr6,excvtr7,excvtr8,exc vtr9,excvtr10}	
Name:	stationProps11	Array for LwrPriority
Type:	Other: int []	
Initial Value:	new int [] {lwr1,lwr2,lwr3,lwr4,lwr5,lwr6,lwr 7,lwr8,lwr9,lwr10}	
Name:	stationProps12	

Type:	Other: int []	Array for BckflPriority
Initial Value:	new int [] {bckfl1,bckfl2,bckfl3,bckfl4,bckfl5,bckfl6,bckfl7,bckfl8,bckfl9,bckfl10}	
Name:	stationProps13	Array for BldzrNo
Type:	Other: int []	
Initial Value:	new int [] {bldzr1,bldzr2,bldzr3,bldzr4,bldzr5,bldzr6,bldzr7,bldzr8,bldzr9,bldzr10}	Array for HdrtstPriority
Name:	stationProps14	
Type:	Other: int []	Array for HdrtstPriority
Initial Value:	new int [] {hdrtst1,hdrtst2,hdrtst3,hdrtst4,hdrtst5,hdrtst6,hdrtst7,hdrtst8,hdrtst9,hdrtst10}	

2.6) Steps (2.3) & (2.4) are repeated to define the parameters' type, range and step. They will be as follows:

Table 6-4: Values of Optimization Parameters

Parameter	Type	Value		
		Min	Max	Step
trucks1	design	1	2	1
trucks2	design	1	2	1
trucks3	design	1	2	1
trucks4	design	1	2	1
trucks5	design	1	2	1
trucks6	design	1	2	1
trucks7	design	1	2	1
trucks8	design	1	2	1
trucks9	design	1	2	1
trucks10	design	1	2	1
bend1	design	1	10	1
bend2	design	1	10	1
bend3	design	1	10	1
bend4	design	1	10	1
bend5	design	1	10	1
bend6	design	1	10	1
bend7	design	1	10	1
bend8	design	1	10	1
bend9	design	1	10	1

bend10	design	1	10	1
weld1	design	1	10	1
weld2	design	1	10	1
weld3	design	1	10	1
weld4	design	1	10	1
weld5	design	1	10	1
weld6	design	1	10	1
weld7	design	1	10	1
weld8	design	1	10	1
weld9	design	1	10	1
weld10	design	1	10	1
welder1	design	1	2	1
welder2	design	1	2	1
welder3	design	1	2	1
welder4	design	1	2	1
welder5	design	1	2	1
welder6	design	1	2	1
welder7	design	1	2	1
welder8	design	1	2	1
welder9	design	1	2	1
welder10	design	1	2	1
coat1	design	1	10	1
coat2	design	1	10	1
coat3	design	1	10	1
coat4	design	1	10	1
coat5	design	1	10	1
coat6	design	1	10	1
coat7	design	1	10	1
coat8	design	1	10	1
coat9	design	1	10	1
coat10	design	1	10	1
excv1	design	1	10	1
excv2	design	1	10	1
excv3	design	1	10	1
excv4	design	1	10	1
excv5	design	1	10	1
excv6	design	1	10	1
excv7	design	1	10	1
excv8	design	1	10	1

excv9	design	1	10	1
excv10	design	1	10	1
excvtr1	design	1	2	1
excvtr2	design	1	2	1
excvtr3	design	1	2	1
excvtr4	design	1	2	1
excvtr5	design	1	2	1
excvtr6	design	1	2	1
excvtr7	design	1	2	1
excvtr8	design	1	2	1
excvtr9	design	1	2	1
excvtr10	design	1	2	1
lwr1	design	1	10	1
lwr2	design	1	10	1
lwr3	design	1	10	1
lwr4	design	1	10	1
lwr5	design	1	10	1
lwr6	design	1	10	1
lwr7	design	1	10	1
lwr8	design	1	10	1
lwr9	design	1	10	1
lwr10	design	1	10	1
bckfl1	design	1	10	1
bckfl2	design	1	10	1
bckfl3	design	1	10	1
bckfl4	design	1	10	1
bckfl5	design	1	10	1
bckfl6	design	1	10	1
bckfl7	design	1	10	1
bckfl8	design	1	10	1
bckfl9	design	1	10	1
bckfl10	design	1	10	1
bldzr1	design	1	2	1
bldzr2	design	1	2	1
bldzr3	design	1	2	1
bldzr4	design	1	2	1
bldzr5	design	1	2	1
bldzr6	design	1	2	1
bldzr7	design	1	2	1

bldzr8	design	1	2	1
bldzr9	design	1	2	1
bldzr10	design	1	2	1
hdtst1	design	1	10	1
hdtst2	design	1	10	1
hdtst3	design	1	10	1
hdtst4	design	1	10	1
hdtst5	design	1	10	1
hdtst6	design	1	10	1
hdtst7	design	1	10	1
hdtst8	design	1	10	1
hdtst9	design	1	10	1
hdtst10	design	1	10	1

The next stage, after defining the optimization parameters, is defining the optimization constraints. A constraint is a condition defined on the optimization parameters. It defines a range for an optimization parameter. Each time the optimization engine generates a new set of values for the optimization parameters, it creates a feasible solution that satisfies this constraint; thus the space of searching is reduced, and the optimization is performed faster. A constraint is a well-formed arithmetic expression describing a relationship between the optimization parameters. It always defines a limitation by specifying a lower or an upper bound e.g. parameter1 >= 10. The constraints needed for this model shall insure that no two parameters of the same array have the same value. In that way, every station has a specific priority for each activity that no other station has. There will be two constraints for the parameters for each array. The first constraint insures that the total value for all ten parameters equals 55 which are total of values from one to ten. The second constraint insures that the value of multiplying all ten parameters equals 3,628,800 which is the factorial of 10 (10!). The following steps to define the optimization constraints would be undertaken in the same order for each array:

3.1) In *Constraints* page of the experiment's properties go to the first row of the constraints table. In the **Expression** field, the first constraint is typed in the form "strng1+strng2+strng3+strng4+strng5+strng6+strng7+strng8+strng9+strng10".

3.2) In the **Type** field, the "=" sign is selected from the dropdown menu.

3.3) In the **Bound** field, the value "55" is entered. The constraint is then enabled by selecting the checkbox in **Enabled** field. The table shall be as follows:

Table 6-5: Definition of 1st Constraint for Stringing Optimization Parameters

Enabled	Expression	Type	Bound
<input checked="" type="checkbox"/>	strng1+strng2+strng3+strng4+strng5+strng6+strng7+strng8+ strng9+strng10	=	55

3.4) In a new row of the constraints table, the second constraint would be defined in the same way described in steps (3.1) to (3.3). it would be as follows:

Table 6-6: Definition of 2nd Constraint for Stringing Optimization Parameters

Enabled	Expression	Type	Bound
<input checked="" type="checkbox"/>	strng1*strng2*strng3*strng4*strng5*strng6*strng7*strng8* strng9*strng10	=	3628800

The constraints table would be as shown in the figure below

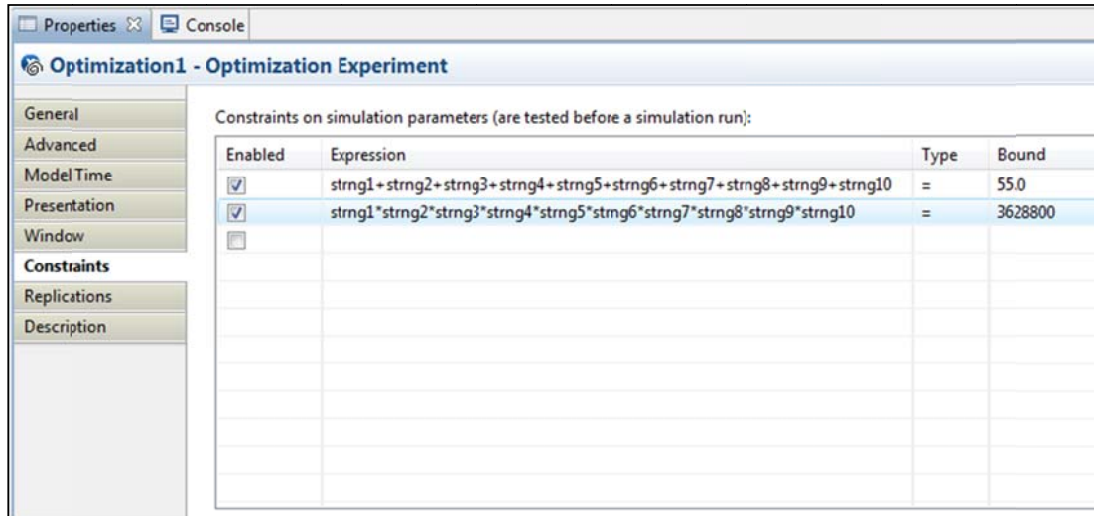


Figure 6-4: Constraints Table in Properties View

3.5) Defining the rest of the constraints by repeating steps (3.1) to (3.4) would result in the following table:

Table 6-7: Constraints for Optimization Parameters

Enabled	Expression	Type	Bound
<input checked="" type="checkbox"/>	bend1+bend2+bend3+bend4+bend5+bend6+bend7+bend8+bend9+bend10	=	55
<input checked="" type="checkbox"/>	bend1*bend2*bend3*bend4*bend5*bend6*bend7*bend8*bend9*bend10	=	3628800
<input checked="" type="checkbox"/>	weld1+weld2+weld3+weld4+weld5+weld6+weld7+weld8+weld9+weld10	=	55
<input checked="" type="checkbox"/>	weld1*weld2*weld3*weld4*weld5*weld6*weld7*weld8*weld9*weld10	=	3628800
<input checked="" type="checkbox"/>	coat1+coat2+coat3+coat4+coat5+coat6+coat7+coat8+coat	=	55

	9+coat10		
<input checked="" type="checkbox"/>	coat1*coat2*coat3*coat4*coat5*coat6*coat7*coat8* coat9*coat10	=	3628800
<input checked="" type="checkbox"/>	excv1+excv2+excv3+excv4+excv5+excv6+excv7+excv8+ excv9+excv10	=	55
<input checked="" type="checkbox"/>	excv1*excv2*excv3*excv4*excv5*excv6*excv7*excv8* excv9*excv10	=	3628800
<input checked="" type="checkbox"/>	lwr1+lwr2+lwr3+lwr4+lwr5+lwr6+lwr7+lwr8+lwr9+lwr10	=	55
<input checked="" type="checkbox"/>	lwr1*lwr2*lwr3*lwr4*lwr5*lwr6*lwr7*lwr8* lwr9*lwr10	=	3628800
<input checked="" type="checkbox"/>	bckfl1+bckfl2+bckfl3+bckfl4+bckfl5+bckfl6+bckfl7+bckfl8 + bckfl9+bckfl10	=	55
<input checked="" type="checkbox"/>	bckfl1*bckfl2*bckfl3*bckfl4*bckfl5*bckfl6*bckfl7*bckfl8* bckfl9*bckfl10	=	3628800
<input checked="" type="checkbox"/>	hdtst1+hdtst2+hdtst3+hdtst4+hdtst5+hdtst6+hdtst7 + hdtst8+hdtst9+hdtst10	=	55
<input checked="" type="checkbox"/>	hdtst1*hdtst2*hdtst3*hdtst4*hdtst5*hdtst6*hdtst7* hdtst8* hdtst9*hdtst10	=	3628800

The fourth and last stage is forming the graphical presentation for the experiment. However, as the outline of the graphical presentation does not affect the optimization experiment. It is just a way to show the ongoing optimization process and the final results in any convenient way of the modeler's choice. Therefore, the graphical presentation used in the current model would be briefly shown without illustrating the specific steps to put it together.

The graphical presentation consists of three parts. The button used to run the simulation, a table to show the current and best iteration values for the optimization parameters and the objective function and a graph to show the different values of objective function and its pattern of declining with iterations. The figure below shows the three parts for one array of optimization parameters.

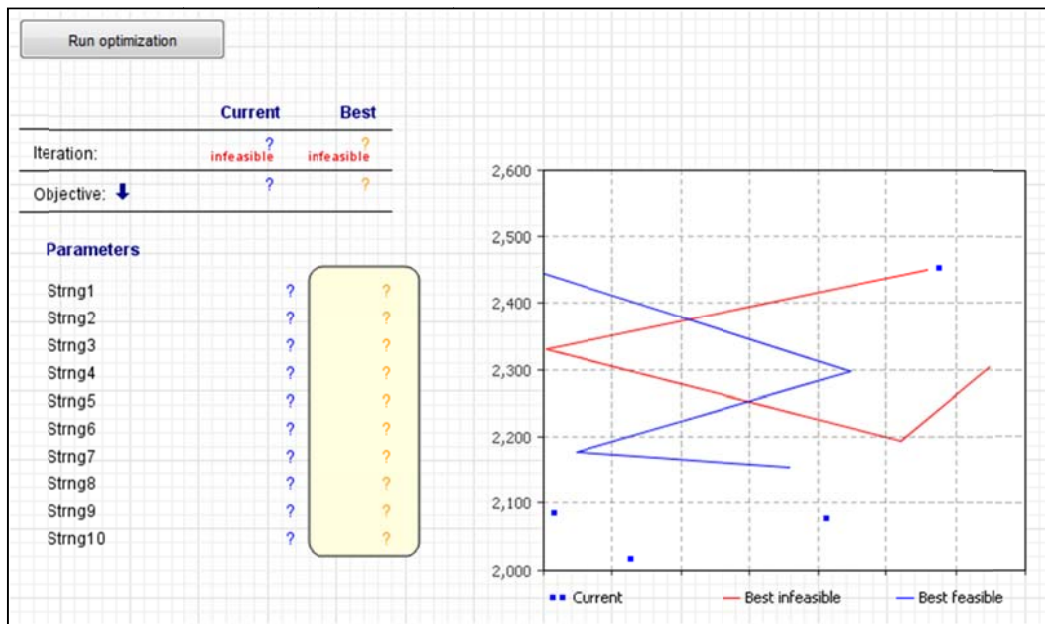


Figure 6-5: Graphical Presentation for Optimization Model

6.3. Case Study Description

The South Valley Gas Pipeline is 930-km pipeline extending from Dahshour in the north to Aswan in the south with a 30"-32" diameter and total investment cost of about 5.7 billion Egyptian pounds. The gas pipeline crosses the Nile at three sites using Horizontal Directional Drilling technology. It is considered the largest gas pipeline in Egypt as its capacity is around 12 billion cubic meters per annum.

The project was a result of the efforts of a number of Egyptian petroleum sector companies such as The Egyptian Gas Holding Company, Ganoub Holding Company, GASCO, PETROJET and ENPPI. It consisted of six phases in addition to Dahshour compression station to keep up with the expected growing demand of gas. The phases were:

- Phase 1: Dahshour – El-Koraimat Gas Pipeline (90 km length, 36" diameter)
- Phase 2: El-Koraimat – Beni Suef Gas Pipeline (30 km length, 32" diameter)
- Phase 3: from Beni Suef to Abu – Qorqas city in Menia (150 km length, 32" diameter)
- Phase 4: from Abu – Qorqas to Assiut (147 km length, 32" diameter)
- Phase 5: from Assiut to Gerga (121 km length, 32" diameter)
- Phase 6: from Gerga to Aswan (390 km length, 30" diameter)



Figure 6-6: Map of South Valley Gas Pipeline

The phases of the South Valley Gas Pipeline were completed in November 2009 as it reached Aswan and started to feed east and west the city. The average construction rate of the pipeline was 360 m/ day. The simulation model represents a 100-km segment of the pipeline. It is divided into 10 stations with a length of 10 km each. The 100-km segment in the case study took about 260 workdays or 2,600 hours. The construction team that worked in this segment is as follows:

- 10 pipe trucks
- 1 pipe bending machine
- 30 welding crews
- 1 weld inspection team
- 5 Joint coating crews
- 6 Excavators
- 6 Side booms
- 4 Bulldozers
- 1 hydro-testing crew

6.4. Application of Model

The resources defined in the model in chapter (5) matches the resources used in the case study. However, In order to overcome the model limitations, some of the utilized resources' numbers were represented differently. The ten trucks were represented by two trucks in the model were each truck represents five trucks. The thirty welding crews were represented as three welders in the model. Each one represents a group of ten crews. In addition, the five joint coating crews are represented by one coating team in the model. The six excavators are condensed in two groups represented by two excavators. Also, the four bulldozers are represented by two in the model. On the other hand, the model simulates well the other characteristics of the real project. The layout of the pipeline project as well as the location of the base camp is represented in the model. In addition, the activities durations are represented accurately in the model. However, the triangular distribution of the durations is assumed to show the simulation model capabilities.

The process of evaluating the model and examining its results consists of three stages of model runs. These stages aim to calibrate the model conditions and monitor its outputs. The result of these stages would be a well-defined model that simulates the real construction process, forecasts any problems or obstacles and obtains a better schedule by optimizing the resource utilization to get less project duration than the actual schedule.

In the first stage, the simulation model was run without any alteration from the original conditions and schedule of the case study. This is done through the model illustrated in chapter (5) without introducing the optimization parameters defined in this chapter. The results of these runs are to be compared to the actual duration of construction in the case study. In the case study, the construction of 100-km segment took about 2,600 hours. After running the simulation 50 times, the average resulting duration is 2,408 hours with standard deviation of 70 hours. The maximum value was 2,579 hours and the minimum was 2,300 hours. The following graph shows the values obtained in the fifty runs. The results show that the model gets a relatively close value to the real duration of the case study on which the model was developed. However, the variation of results from the actual duration was due to the triangular distribution of activities' durations in the model against the deterministic values calculated in the real schedule.

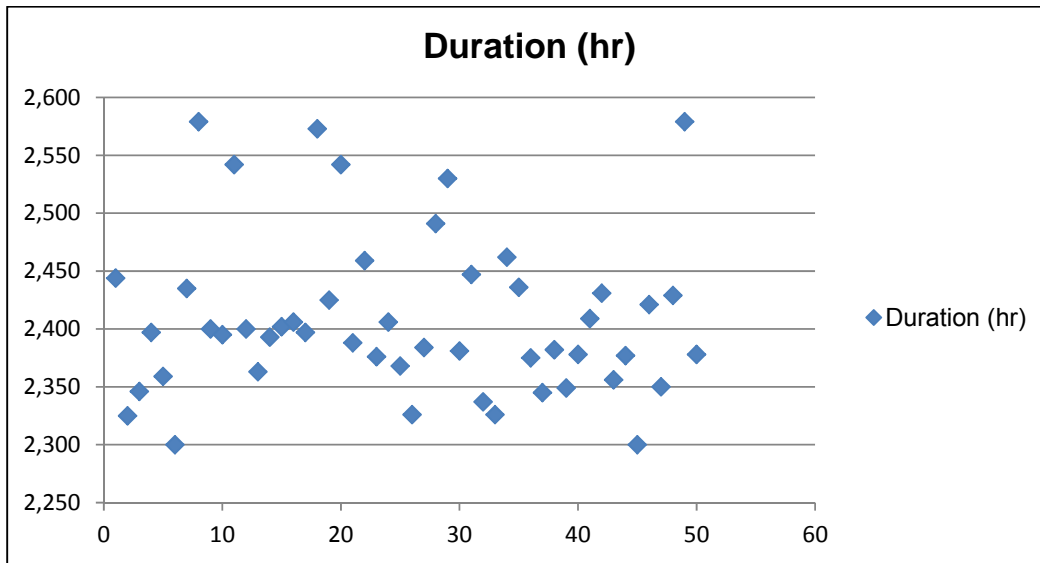


Figure 6-7: Results of 50 Simulation Runs

In the second stage of simulation model runs, a simplified version of the optimization module developed in previous chapter is used. As opposed to finding a separate sequence of work for each activity, the objective of this module would be to find an optimal sequence that is fixed for all activities.

The objective of this stage is to test the optimization module using a smaller number of optimization parameters in order to locate any bugs and fix them. In addition, the resulting sequence of activities and the associated minimization of total duration of construction is observed to verify the model and assure its effectiveness.

The simulation model as well as the optimization module that were defined in the previous chapters has to be modified in order to match the proposed model. There are two main steps to achieve that; the first is to assign a single array of priorities from the arrays previously defined to be the array of priorities for all activities. The second step is to define the optimization parameters as the elements of that array.

The first step is accomplished by modifying the Java code in **NetworkEnter** object that was defined in step (3.6) in chapter (5). This modification shall assign the same value for each entity's priority in every activity. The array used would be "stationProps1" array. The modification in the code entered in **On enter** field shall be as follows:

Name:	networkEnter	(The default name)
Entity class:	Entity	(The name of the subclass created in second stage)
Network:	network	(The network defined in step (3.1))
Entry node:	entitySource	(Part of the graphical network. Refer to step (1.3))

On enter: (The following code is executed as every entity enters the object)

```
int i=j;
strngProp1=stationProps1[i];
if( entity instanceof Station )
((Station)entity).StrngPriority = strngProp1;
strngProp2=stationProps2[i];
if( entity instanceof Station ) ((Station)entity).TruckNo
= strngProp2;
//
bendProp1=stationProps1[i];
if( entity instanceof Station )
((Station)entity).BendPriority = bendProp1;
bendProp2=stationProps4[i];
if( entity instanceof Station ) ((Station)entity).BendNo
= bendProp2;
//
weldProp1=stationProps1[i];
if( entity instanceof Station )
((Station)entity).WeldPriority = weldProp1;
weldProp2=stationProps6[i];
if( entity instanceof Station )
((Station)entity).WelderNo = weldProp2;
//
coatProp1=stationProps1[i];
if( entity instanceof Station )
((Station)entity).CoatPriority = coatProp1;
//
excVProp1=stationProps8[i];
if( entity instanceof Station )
((Station)entity).ExcVdiff = excVProp1;
excVProp2=stationProps1[i];
if( entity instanceof Station )
((Station)entity).ExcVPriority = excVProp2;
excVProp3=stationProps10[i];
if( entity instanceof Station ) ((Station)entity).ExcVNo
= excVProp3;
//
lwrProp1=stationProps1[i];
if( entity instanceof Station )
((Station)entity).LwrPriority = lwrProp1;
//
bckflProp1=stationProps1[i];
if( entity instanceof Station )
((Station)entity).BckflPriority = bckflProp1;
bckflProp2=stationProps13[i];
if( entity instanceof Station ) ((Station)entity).BldzrNo
= bckflProp2;
//
hdtstProp1=stationProps1[i];
if( entity instanceof Station )
((Station)entity).HdtstPriority = hdtstProp1;
j++;
```

The second step is to change the type of the rest of the parameters shown in the optimization parameters table to “fixed”. This way the only parameters that would be adjusted in the

optimization process shall be strng1 to strng10. As those two modifications to the model are done, the optimization model is ready for running phase.

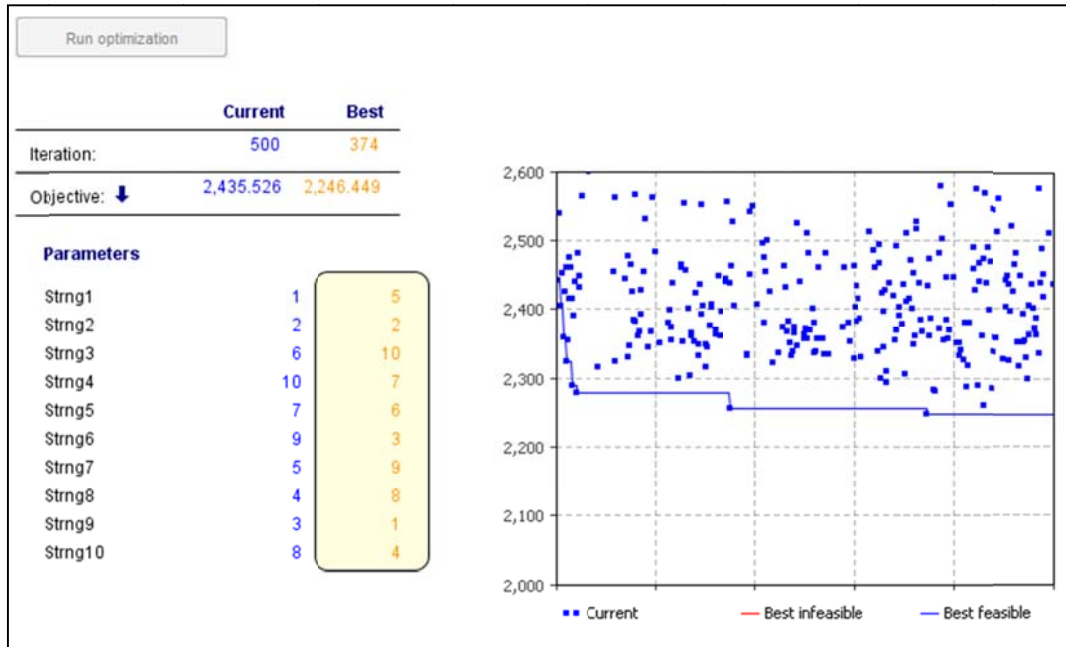


Figure 6-8: Second Stage Optimization Model

After running the optimization model for ten times, the result shown in the figure above was reached in all runs. Instead of the original sequence, an alternative sequence was devised through the model. The resulting sequence is as follows:

Table 6-8: Second Stage's Resulting Sequence of Work for Pipeline Stations

Station Order	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth	Ninth	Tenth
Station number	5	2	10	7	6	3	9	8	1	4

The result of optimization shows a reduction of total duration from 2,408 hours in the original schedule to **2,247** hours i.e. 161 working hours were saved with a reduction ratio of **6.6%**. In addition, no bugs were encountered running the optimization model.

As the second stage of runs was completed successfully, the optimization model defined in the previous chapter could be used to complete the objective of the research by defining the optimum schedule in the third stage of runs. As the model will not need any modifications, the results of this stage would be presented straight away. However, due to hardware limitations of the computer used to run the optimization model, the model could not be run as a whole. Therefore, a number of optimization sub-models were developed to reduce the number of variables in each sub-model to match the computational abilities of the computer.

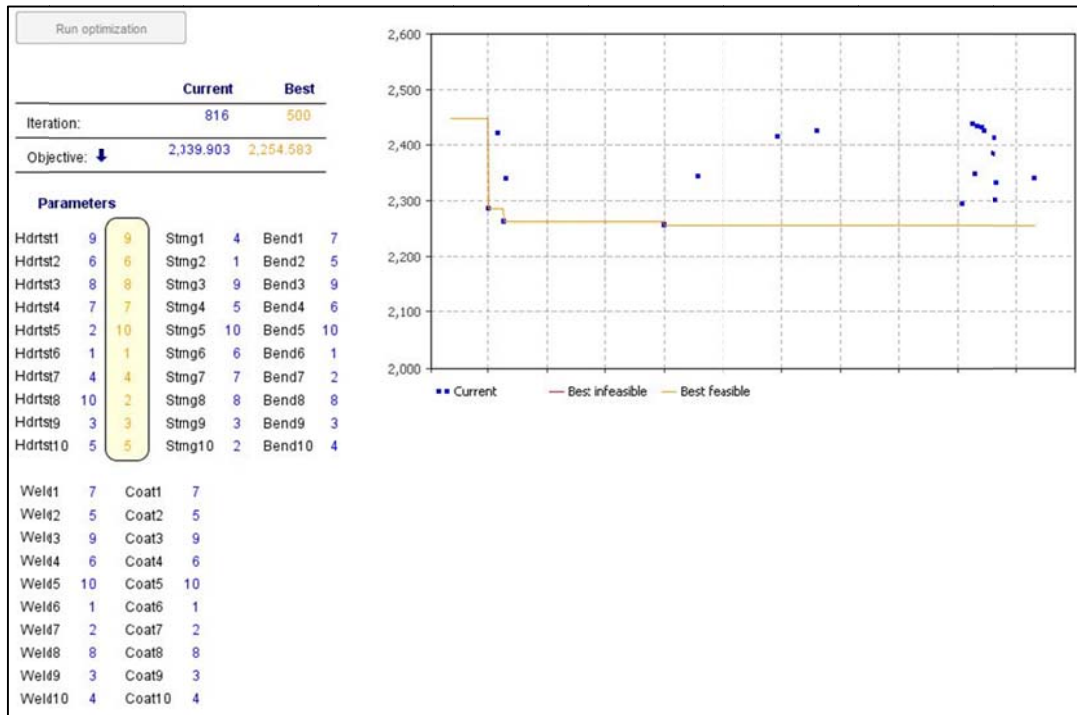


Figure 6-9: Example of Used Sub-models

After running each of the sub-models for 5 times at least, the resulting sequence was as follows where each station has its order in the sequence for each activity:

Table 6-9: Final Sequence of Work for Pipeline Stations

	Station Order	First	Second	Third	Fourth	Fifth	Sixth	Seventh	Eighth	Ninth	Tenth
	Activity										
Station number	Stringing	5	3	8	7	6	4	1	9	10	2
	Bending	5	3	8	1	4	2	10	9	7	6
	Welding	5	3	8	1	4	2	10	9	7	6
	Coating	5	3	8	1	4	2	10	9	7	6
	Excavation	5	1	3	4	2	10	9	8	6	7
	Lowering	5	1	3	4	2	10	9	8	6	7
	Backfilling	5	1	3	4	2	10	9	8	6	7
	Hydro-testing	5	1	3	4	2	10	9	8	6	7

The result of optimization shows a total duration of 2,238 hours. This sequence has reduced 170 working hours in comparison to the original sequence with a reduction ratio of 7.1%. No bugs were encountered running the optimization model.

6.5. Validation of Model

The final step is to validate the results of both the simulation and optimization models and show how relative they are to actual schedules developed using traditional methods. The method used for this purpose was presenting the bar charts for the original schedule for the case study project as well as the optimized schedule, shown in appendix C, to five professionals in the field of project management and planning for oil and gas construction projection with years of experience that ranges from two to fifteen years. After illustrating AnyLogic software to them and summarizing the work done in the simulation and optimization models, they were required to answer a questionnaire with five questions by giving each a question a degree from (1) to (5) where (1) stands for **strongly disagree** and (5) stands for **strongly agree**. The five questions that were presented were as follows:

1. Is the actual schedule reasonable?
2. Is the optimized schedule reasonable?
3. Is using the simulation model easy?
4. Is changing any parameters in the model easy?
5. Would you use it in scheduling a real project?

The results are presented in below figure and they were summarized to show the following conclusions:

1. The actual and optimized schedules were accepted and found reasonable
2. The model needs to be more user-friendly as it is hard to modify any parameter or any part of the model to fit another project
3. The simulation model is highly effective in showing the work progress at any instant of the project life time
4. The optimization model may present an effective tool in the future in the field of planning and scheduling

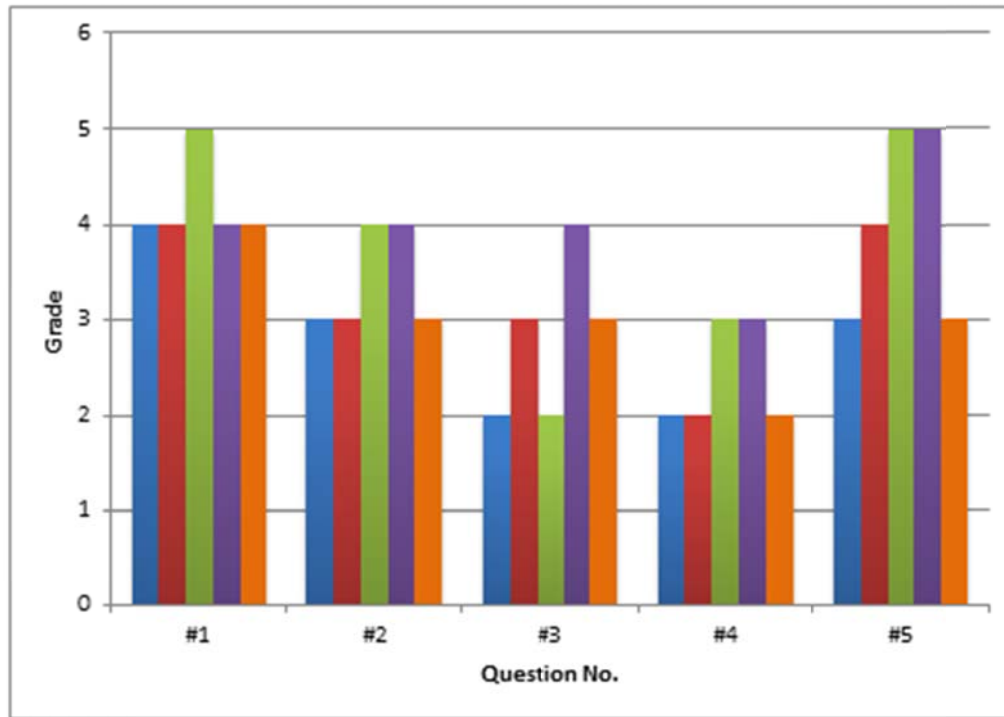


Figure 6-10: Results of Questionnaire

6.6. Conclusion

As the simulation model was finished in the previous chapter, the next stage was to construct the optimization model. It would be used to find an optimal solution for the scheduling problem in hand. The stages of construction of this model were shown in this chapter. The first stage was defining the objective function which is the construction time. The second stage was to define the optimization parameters which would be used to reach the optimal solution. The third stage was defining the constraints that control the values assigned for the optimization parameters. The final stage would result in a graphical representation of the iterations and final result of the optimization process. In the last part of the chapter, a real case study was presented to evaluate the results' accuracy against the real figures and its effectiveness in reducing the total duration of the project. The validation of the model using a questionnaire that was presented to five professionals in project management field has shown that although the model still needs to be easier to use, it significantly improves the scheduling process and would be a very useful tool in the future.

CHAPTER 7

CONCLUSIONS

7.1. Summary of Research

This study presents a simulation model, designed to aid construction personnel in planning and scheduling pipeline construction projects. The research tried to explore all issues that arise when scheduling a project with many repetitive activities considering all repetitive project characteristics. The development of the model was a means to solve any problems that might arise when conducting repetitive project scheduling and not considering the important aspects of repetitive project.

The first stage of this research was defining the research problem. A comprehensive review of the domain literature has been conducted. This stage of the research found that the current repetitive project scheduling techniques do not provide project planners and management with the optimal sequence of work among the units of the project; Other problems and limitations that were addressed can be summarized as follows:

1. Difficulty of Visualizing the entire project.
2. Continuity of work for crews is not ensured.
3. Dealing with resources constraints is very difficult.
4. Difficulty of incorporating the probabilistic nature of durations
5. Logical interconnections are extremely difficult to comprehend.
6. Inability to give justifications for management for their decisions
7. Difficulty of predicting needed corrective actions.
8. Dealing with space related constraints

Then, a thorough study of pipeline construction process was held to fully comprehend the different activities, the needed resources for each activity and the various characteristics of the project's units; a process which would be well used in defining the different classes and variables of the model in the next stage. Last, examples of utilizing deterministic models, developed in the literature, for the present scheduling problem in order to present their limitation in dealing with repetitive projects.

The second stage of development of the present simulation model, after defining the problems and limitations, was the design stage. First, the typical form of the actual space of a pipeline construction project was represented as a network that includes all the nodes and the paths linking them. The length of the paths on the model represents the actual distances covered by the equipment during the construction works. Next, different classes were defined for the project's units or stations as well as the utilized resources. These classes helped in defining any needed characteristics to the different stations such as soil properties; characteristics which would affect the schedule and the needed resources. After that, the

logical network connecting the project's activities, the resource pools, the process variables and the graphic network is created. In this network, the probabilistic durations, the required resources and the order of work in stations for each activity is defined. Also, any outputs such as the total duration are defined in this stage. Finally, a simulation experiment is created to provide the visual overview of the construction process and the values of outputs.

The third stage is creating the optimization experiment. This experiment would be used to find the optimal schedule and number of resources utilized in each station for each activity. In this experiment, arrays of stations' priority to seize the available resources and the number of resources are defined as the "Optimization Parameters". AnyLogic software has an optimization engine that would automatically find the optimum values for these arrays to achieve the objective of minimizing the total duration of the project while preserving the defined constraints. The output of this experiment shall be the required schedule i.e. the main objective of this research.

7.2. Research Contributions

The primary purpose of this research study is to develop a new model to overcome some of the most important shortcomings and limitations of the current scheduling methods in scheduling projects with many repetitive activities. The development of this model incorporates the following contributions:

1. The primary contribution of this research is the development of a scheduling simulation model for scheduling repetitive linear construction projects in general, and pipeline construction projects in particular. Both typical and atypical activities could be modeled easily in same model.
2. Identifying the activities required in pipeline construction projects and their interrelationships by means of studying the literature, reviewing industry references and performing an in-depth analysis of a solid case study.
3. The development of a method that aids the management in running multiple scenarios for any needed corrective action plan.
4. The model allows the planners to evaluate the impact of using probabilistic activity durations on project completion time using any type of statistical distributions.
5. The incorporation of the detailed geometric layout of the project such as the base camp location, the length of linking roads as well as any physical obstructions.
6. The model allows utilization of multiple crews at different locations simultaneously

7. The ability to define any attributes to different stations and resources such as different quantities of bends in the pipeline, different soil types and different productivity rates for equipment
8. The model allows the incorporation of non-repetitive activities along with repetitive ones
9. The development of an optimization model for generating least duration schedules for all types of repetitive construction projects.
10. The development of visual animation of execution of the project that could aid the management, planners and construction superintendents in visualizing the project's activities. Later versions of AnyLogic software provide a 3D animation model.

7.3. Recommendations for Future Research

The present simulation-based model for repetitive project scheduling is a promising trial that can be utilized in future research efforts. The model is flexible and can be applied to schedule and control any type of linear repetitive projects. However, in order to expand the potential applications of this model, the following recommendations for future research can be made:

1. The model has a number of shortcomings that affect the output. The first is that AnyLogic software has a fixed rule to utilize all available resources in the resource pool in a fixed order even if a nearer resource unit is available. Future research can find a way to avoid this rule. The second is that the triangular distribution for probabilistic durations was assumed. The literature provides many methods that could be used to determine the suitable distribution for activity durations. The last is that the optimization model requires a high-end hardware that has high computational capabilities in order to get credible results.
2. A GIS sub-module could be combined with the model to get many advantages. Such module could be used to determine the suitable locations for base camps in relation to populated areas and infrastructure. It can highlight any potential conflicts with any existing utilities, such as power lines and roads and incorporate them in the schedule.
3. The model could incorporate the weather effect of activities durations, productivities and unscheduled stops
4. A sub-module could be added to the model to optimize the number of crews and equipment needed for the project to get the best result for crew work continuity

5. As project planners cannot be expected to be knowledgeable about simulation techniques and the programming languages used in simulation, user-friendly software could be developed to allow input entry from the user, builds a model and presents the simulation results.

Appendix A

Java Code for Costume Class “Station”

```

/**
 * stati
 */
public class Station extends
com.xj.anylogic.libraries.enterprise.Entity implements
java.io.Serializable {

    double PipeNo;

    double StrngPriority;

    double TruckNo;

    double BendPriority;

    double BendNo;

    double WeldPriority;

    double WelderNo;

    double WeldPipeNo;

    double CoatPriority;

    double Excvdiff;

    double ExcvPriority;

    double ExcvNo;

    double LwrPriority;

    double BckflPriority;

    double BldzrNo;

    double HdrtstPriority;

    /**
     * Default constructor
     */
    public Station(){
    }

    /**
     * Constructor initializing the fields

```



```

        */
        public Station(double PipeNo, double StrngPriority,
double TruckNo, double BendPriority, double BendNo,
double WeldPriority, double WelderNo, double
WeldPipeNo, double CoatPriority, double Excvdif,
double ExcPriority, double ExcNo, double LwrPriority,
double BckflPriority, double BldzrNo, double
HdrtstPriority){
            this.PipeNo = PipeNo;
            this.StrngPriority = StrngPriority;
            this.TruckNo = TruckNo;
            this.BendPriority = BendPriority;
            this.TruckNo = BendNo;
            this.WeldPriority = WeldPriority;
            this.WelderNo = WelderNo;
            this.WeldPipeNo = WeldPipeNo;
            this.CoatPriority = CoatPriority;
            this.ExcVdiff = ExcVdiff;
            this.ExcPriority = ExcPriority;
            this.ExcNo = ExcNo;
            this.LwrPriority = LwrPriority;
            this.BckflPriority = BckflPriority;
            this.BldzrNo = BldzrNo;
            this.HdrtstPriority = HdrtstPriority;
        }

        @Override
        public String toString() {
            return
                "PipeNo = " + PipeNo + " " +
                "StrngPriority = " + StrngPriority + " " +
                "TruckNo = " + TruckNo + " " +
                "BendPriority = " + BendPriority + " " +
                "BendNo = " + BendNo + " " +
                "WeldPriority = " + WeldPriority + " " +
                "WelderNo = " + WelderNo + " " +
                "WeldPipeNo = " + WeldPipeNo + " " +
                "CoatPriority = " + CoatPriority + " " +
                "ExcVdiff = " + ExcVdiff + " " +
                "ExcPriority = " + ExcPriority + " " +
                "ExcNo = " + ExcNo + " " +
                "LwrPriority = " + LwrPriority + " " +
                "BckflPriority = " + BckflPriority + " " +
                "BldzrNo = " + BldzrNo + " " +
                "HdrtstPriority = " + HdrtstPriority + "
";
        }

```

```
        /**
         * This number is here for model snapshot storing
purpose<br>
         * It needs to be changed when this class gets
changed
         */
        private static final long serialVersionUID = 1L;
    }
```

Appendix B

Resource Pools Properties

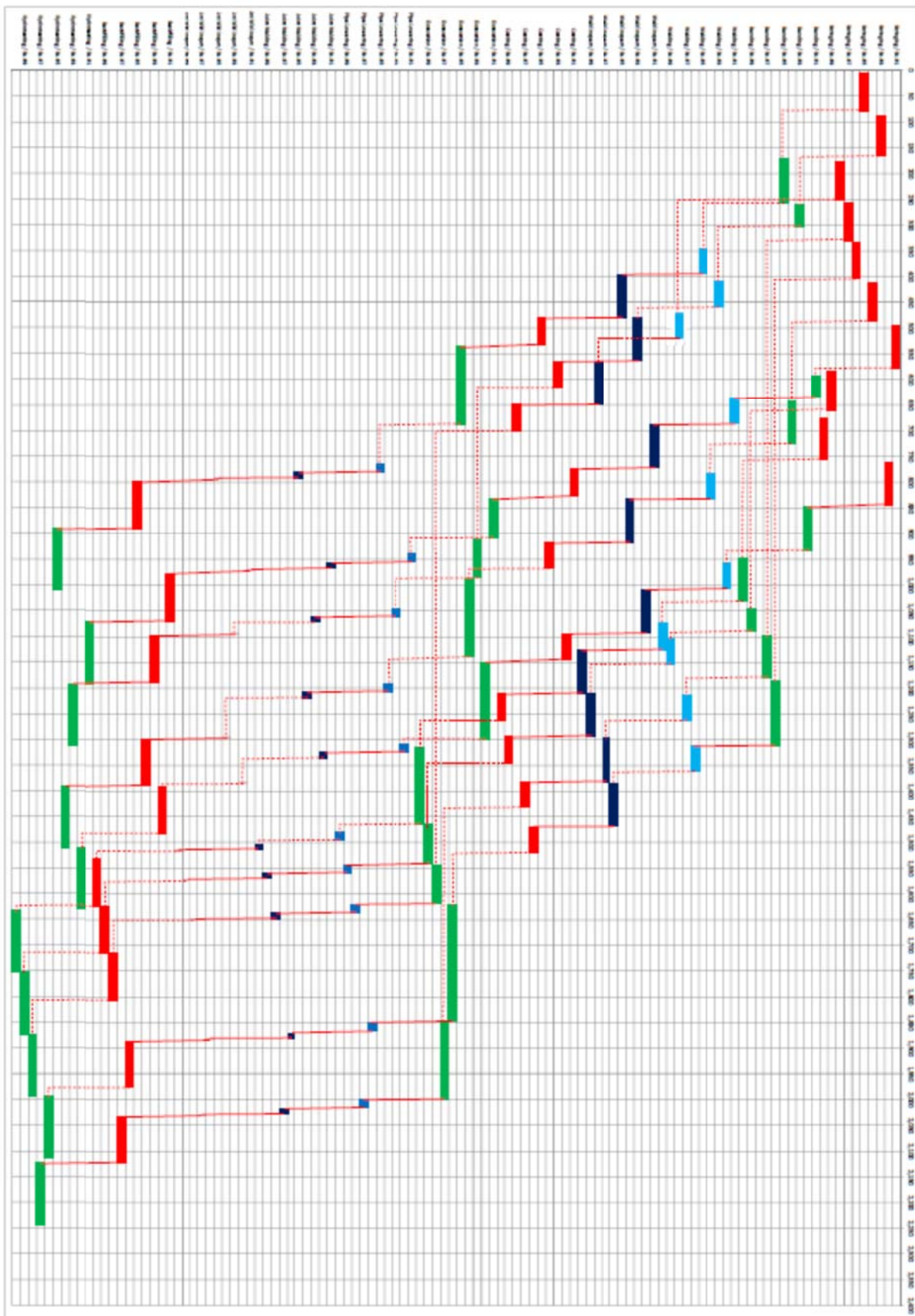
Pipes	Name:	Pipe
	Resource type:	Portable
	Capacity defined:	Directly
	Capacity:	100
	Idle unit animation shape:	PipeShape (Refer to table (5-1))
	Busy unit animation shape:	PipeShape
	Home defined by:	Single node
	Home path:	PipeYard
Trucks	Name:	Truck
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	2
	Speed:	150
	Idle unit animation shape:	TruckShape (Refer to table (5-1))
	Busy unit animation shape:	TruckShape
	Home defined by:	Single node
	Home path:	ResidenceCamp
Side booms	Name:	Sideboom
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	4
	Speed:	25
	Idle unit animation shape:	SideboomShape (Refer to table (5-1))
	Busy unit animation shape:	SideboomShape
	Home defined by:	Single node
	Home path:	ResidenceCamp
Pipe benders	Name:	Pipebender
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	1
	Speed:	25
	Idle unit animation shape:	PipebenderShape (Refer to table (5-1))
	Busy unit animation shape:	PipebenderShape

	Home defined by:	Single node
	Home path:	ResidenceCamp
Welders	Name:	Welder
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	3
	Speed:	150
	Idle unit animation shape:	WelderShape (Refer to table (5-1))
	Busy unit animation shape:	WelderShape
	Home defined by:	Single node
	Home path:	ResidenceCamp
Excavators	Name:	Excavator
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	2
	Speed:	25
	Idle unit animation shape:	ExcavShape (Refer to table (5-1))
	Busy unit animation shape:	ExcavShape
	Home defined by:	Single node
	Home path:	ResidenceCamp
Inspection teams	Name:	Inspector
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	1
	Speed:	150
	Idle unit animation shape:	InspectorShape (Refer to table (5-1))
	Busy unit animation shape:	InspectorShape
	Home defined by:	Single node
	Home path:	ResidenceCamp
Coating teams	Name:	CoatingTeam
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	1
	Speed:	150

	Idle unit animation shape:	CoatShape (Refer to table (5-1))
	Busy unit animation shape:	CoatShape
	Home defined by:	Single node
	Home path:	ResidenceCamp
Bulldozers	Name:	Bulldozer
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	2
	Speed:	25
	Idle unit animation shape:	BulldozerShape (Refer to table (5-1))
	Busy unit animation shape:	BulldozerShape
	Home defined by:	Single node
	Home path:	ResidenceCamp
Hydro-testing teams	Name:	HydTestTeam
	Resource type:	Moving
	Capacity defined:	Directly
	Capacity:	1
	Speed:	150
	Idle unit animation shape:	HydTestShape (Refer to table (5-1))
	Busy unit animation shape:	HydTestShape
	Home defined by:	Single node
	Home path:	ResidenceCamp

Appendix C

Bar Charts for Actual and Optimized Schedules



REFERNCES

- Al Sarraj, Z. M. (1990). Formal Development of Line-of-Balance Technique. *Journal of Construction Engineering and Management*. 116(4), 689 – 704.
- Ammar, M. A. (2013). LOB and CPM Integrated Method for Scheduling Repetitive Projects. *Journal Of Construction Engineering & Management*, 139(1), 44-50.
- Arditi, D. and Albulak, Z. (1986). Line-of-Balance Scheduling in Pavement Construction. *Journal of Construction Engineering and Management*. 112(3), 411-424.
- Ashley, D.B. (1980). Simulation of Repetitive Unit Construction. *Journal of the Construction Division*. 106(C02), 185-194.
- Birrell, G. (1980). Construction Planning - Beyond the Critical Path. *Journal of the Construction Division*. 106(C03), 389-407.
- Carr, M.I. and Meyer, W.L. (1974). Planning Construction of Repetitive Building Units. *Journal of the Construction Division*. 100(3), 403-412.
- Chrzanowski, E. and Johnston, D. (1986), Application of Linear Scheduling. *Journal of Construction Engineering and Management*. 112(4), 476-491.
- Dressler, J. (1980). Construction Management in West Germany. *Journal of the Construction Division*. 106(C04), 447-487.
- Eldin, N. and Senouci, A. (1994). Scheduling and Control of Linear Projects. *Canadian Journal of Civil Engineering, CSCE*. 21, 219-230.
- El-Rayes, K. (1997). *Optimized scheduling for repetitive construction projects* (Order No. NQ40315). Available from ProQuest Dissertations & Theses Global. (304460117). Retrieved from <http://search.proquest.com/docview/304460117?accountid=8423>
- El-Rayes, K. and Moselhi, O. (1998). Resource-Driven Scheduling of Repetitive Activities. *Construction Management and Economics*. 16, 433-446.
- Hajdasz, M (2014). Flexible management of repetitive construction processes by an intelligent support system. *Expert Systems with Applications*. 41 (2014), 962–973.
- Handa, V. and Barcia, R. (1986). Linear Scheduling Using Optimal Control Theory. *Journal of Construction Engineering and Management*. 112(3), 387-393.

Hassanein, A. (2003). *Planning and scheduling highway construction using GIS and dynamic programming* (Order No. NQ77904). Available from ProQuest Dissertations & Theses Global. (305303811).

Retrieved from <http://search.proquest.com/docview/305303811?accountid=8423>

Insa Wrede, S. (2014, August). Europe's approach to Russia's gas power. *Deutsche Welle Website*. Retrieved from <http://www.dw.de/europes-approach-to-russias-gas-power/a-17887505>

Johnson, S.M. (1954). Optimal Two and Three-Stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly*. 1, 61–67.

Johnston, D. (1981). Linear Scheduling Method for Highway Construction. *Journal of the Construction Division*. 107(C02), 247-261.

Kavanagh, D.P. (1985). SIREN: A Repetitive Construction Simulation Model. *Journal of Construction Engineering and Management*. 111(3), 308-323.

Kim, K. and de la Garza, J. M. (2003). Phantom Float. *Journal of Construction Engineering and Management*. 129(5), 507 - 517.

Moradi, S., Nasirzadeh, F and Golkhoo, F (2015). A hybrid SD–DES simulation approach to model construction projects. *Construction Innovation*. 15 (1), 66 – 83.

Moselhi, O. and El-Rayes, K. (1993 (a)). Scheduling of Repetitive Projects with Cost Optimization. *Journal of Construction Engineering and Management*. 118(4), 681-697.

Moselhi, O. and El-Rayes, K.(1993 (b)). Least Cost Scheduling for Repetitive Projects. *Canadian Journal of Civil Engineering*, CSCE. 20, 834-843.

Moselhi, O. and Hassanein, A. (2003). Optimized Scheduling of Linear Projects. *Journal of Construction Engineering and Management*. 129(6), 664 - 673.

Neale, R. H., and Raju, B. (1988). Line of Balance Planning by Spread Sheet. *Building Technology and Management*. (January), 22-27.

O'Brien, J.J. (1975). VPM Scheduling for High-Rise Buildings. *Journal of the Construction Division*. 101(4), 895-905.

O'Brien, J.J., Kreitzberg, F.C. and Mikes, W.F. (1985). Network Scheduling Variations for Repetitive Work. *Journal of Construction Engineering and Management*. 111(2), 105-116.

Palmer, D.S. (1965). Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time - A Quick Method of Obtaining a Near Optimum. *Operational Research Quarterly*. 16, 101–107.

Peña-Mora, F., Han, S., Lee, S., and Park, M. (2008). Strategic-Operational Construction Management: Hybrid System Dynamics and Discrete Event Approach. *Journal of Construction Engineering and Management*. 134(9), 701–710.

Pinedo, M.L. (2011). *Scheduling: Theory, algorithms, and systems, Fourth Edition*, New York: Springer.

Reda, R.M. (1990). RPM: Repetitive Project Modeling. *Journal of Construction Engineering and Management*. 116(2), 316-330.

Russell, A.D. and Caselton, W.F. (1988). Extensions to linear Scheduling Optimization. *Journal of Construction Engineering and Management*. 114(1), 36-52.

Selinger, S. (1980). Construction Planning for Linear Projects. *Journal of the Construction Division*. 106(C02), 195-205.

Smith, C.E. (2013, Febuary). Worldwide Pipeline Construction: Crude, products plans push 2013 construction sharply higher. *Oil & Gas Journal*. Retrieved from <http://www.ogj.com/articles/print/volume-111/issue-02/special-report--worldwide-pipeline-construction/worldwide-pipeline-construction-crude-products.html>

Stradal, O. and Cacha, J. (1982). Time Space Scheduling Method. *Journal of the Construction Division*. 108(C03), 445-457.

Tavakoli, A. and Riachi, R. (1990). CPM Use in ENR Top 400 Contractors. *Journal of Management in Engineering*. 6(3), 282–295.

Thabet, W. Y., and Beliveau, Y. L. (1994). HVLS: Horizontal and Vertical Logic Scheduling for Multistory Projects. *Journal of Construction Engineering and Management*. 120(4), 875-892.

Wagner, H.M. (1959). An Integer Programming Model for Machine Scheduling. *Naval Research Logistics Quarterly*. 6, 131–140.

Whiteman, W. E., and Irwing, H. G. (1988). Disturbance Scheduling Techniques for Managing Renovation Work. *Journal of Construction Engineering and Management*. 114(2), 191-213.

Worldwide Onshore Pipeline Construction Market Appears Strong through 2015 (2010, December). *Pipeline News*. Retrieved from <http://pipeline-news.com/feature/worldwide-onshore-pipeline-construction-market-appears-strong-through-2015>

