University of Alberta

Simulation-Based Scheduling of Module Assembly Yards with Logical and Physical Constraints

By ©

Luis Francisco Davila Borrego

A thesis submitted to the faculty of Graduate Studies in partial fulfillment of the requirements for the degree of **Master of Science**

in

Construction Engineering and Management

The Department of Civil and Environmental Engineering

Edmonton, Alberta
Fall 2004

Your file Votre référence
ISBN: 0-612-95730-6
Our file Notre référence
ISBN: 0-612-95730-6

Acknowledgement

The author wishes to express his sincere thanks to his two academic advisors, Dr. S. M. AbouRizk, and Dr. M. Al-Hussein for their guidance and encouragement in the preparation of this thesis.

Special thanks are due to my sponsors, "Becas Magdalena O. Vda. De Brockmann, A. C.," and to the NSERC / Alberta Construction Industry Research Chair for their generosity and confidence towards me, as well as to my family, specially my wife Clara for her decisive support and understanding during the years of this research.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

NSERC – Natural Sciences and Engineering Research Council of Canada

CPM – Critical Path Method

FLP – Facilities Layout Planning

SPS – Special Purpose Simulation

VBA – Visual Basic Applications

N/A – Not Applicable

ACT – Activity ID

ESA – Indicates the status of a module in terms of in progress or not in progress yet

EFA – Indicates the status of a module in terms of finished or not finished yet

P – Priority

F – Float

PSD – Planned Ship Date

ESD – Early Start Date

D – Duration

TD – Today (day when the simulation is performed)

CT – Cable Tray

EM – Equipment

M – Miscellaneous

PM – Pipe Rack

S – Structural

EHT – Electrical Heat Tracing

SimTime – Simulation Time

Chapter 1. Introduction

1.1 Motivation

Scheduling a module assembly yard is a difficult task, involving a number of factors, which govern the ultimate decision of module allocation. Those factors comprise physical and logical constraints imposed by the module yard as well as heuristic, experience-based scheduling rules used by superintendents. The module fabrication industry needs advanced tools and techniques for planning scheduling module assembly yards effectively. Allocating modules within a yard is a time-consuming task that must be improved. Module allocation must manage the constantly changing delivery dates and definite shipping dates by performing regular and weekly updates. In a module assembly yard, the type of scheduling problem is mainly determined by the allocation of each module within the yard (a module yard is divided into "bays" and each module occupies a fraction of a bay), the start and finish times of each module, and by ensuring that no constraints are violated. In addition, finish times need to be minimized since module shipping dates depend on them and likewise the yard usage must be maximized. Therefore, there is a need to develop a method that will assist the scheduler in distributing the modules in the assembly yard, improve the maintenance of the project schedule, perform regular (weekly) updates, and maximize yard utilization.

Two approaches can be used for solving scheduling problems: optimization or approximation. Optimization methods, which aim to find precise solutions using mathematical algorithms, are often unable to achieve feasible solutions to large problems due to the excessive computing requirements (Chong et al. 2003). Approximation algorithms do not always give an optimal solution; however, the solutions provided do improve results. Priority dispatch rules are perhaps the first approximation techniques used (Panwalkar and Iskander 1977). A dispatching rule is simply a rule of thumb giving priority to a particular order selected from among the many available orders at any stage. Simulation-based approaches are derived from dispatching rule-based approaches. Simulation-based scenarios employ resources to make decisions. When one of these resources becomes available, one or more dispatching rules may be used to make a decision (Banks 1998). Banks (1998) showed that scheduling problems generally must include restrictive assumptions in order to be solvable. When scheduling module assembly yards the following restrictive assumptions are applied:

1. Each module is an entity; no more than one module can be processed at the same time and in the same space.
2. There may be no preemption; once the fabrication of a module has started, it must be completed before another module can begin its fabrication in the same space.
3. There may be no cancellations; the assembly of modules must be brought to completion.
4. The fabrication of modules must be continuous.
5. A specific space may not be assigned to more than one module at a time.
6. A particular bay's space is available throughout the scheduling period.
7. The technological constraints behind the assembly process are known in advance and are immutable.
8. There is no randomness in the following items, however randomness may be incorporated to test if/then scenarios:
    a. The number of modules to assemble is known and fixed.
    b. The number of bays is known and fixed.
    c. The fabrication times are known and fixed.
    d. All other quantities needed to define a particular problem are known and fixed.

An operation's start and finish times for each job waiting to be processed must respond to the technological constraints approximating to a good solution but not always ensuring optimality. Each job is defined by its operations, processing times, and due dates. In a deterministic scheduling problem, numeric quantities such as processing times and due dates are assumed to be known in advance. However, most numerical quantities are not known in advance in real life; therefore, they are stochastic (subject to randomness). When facing deterministic-static problems with known data in advance, optimization-based approaches are more convenient. Nevertheless, simulation-based approaches are more useful when the data in not known in advance.

1.2 Research Objectives

The main objective of this research is to provide both a solution and improvements for the module yard scheduling practices. This goal will be accomplished through the development of a simulation model that contractors can use for finding appropriate solutions for module distribution and project schedule maintenance, and to maximize yard utilization.

The solutions will provide:

- A tabular format providing start date, finish date, ship date, location of module within yard, and location of module within bay.
- An auto-generate layout, which is a useful tool for the scheduler since the location and starting time of each module will be provided.

1.3 Research Methodology

To achieve the objectives a simulation-based technique has been developed. The problem is suitable for a simulation-based technique because:

- Physical and logical constraints as well as heuristic scheduling rules that superintendents use in real life can be implemented within the model,
- The process is based on the availability of resources, and
- Many different scenarios can be tested in order to obtain the one providing the best performance based on the accomplishment of delivery dates.

The new methodology provides a simple and easy-to-operate tool for module allocation and scheduling. This methodology has been incorporated into a computer system integrating the given information in a database format, through data processing using Visual Basic Application in Excel, and by means of the simulation model developed using *Simphony* (AbouRizk and Hajjar 1998). The system integration is illustrated in Figure 1.

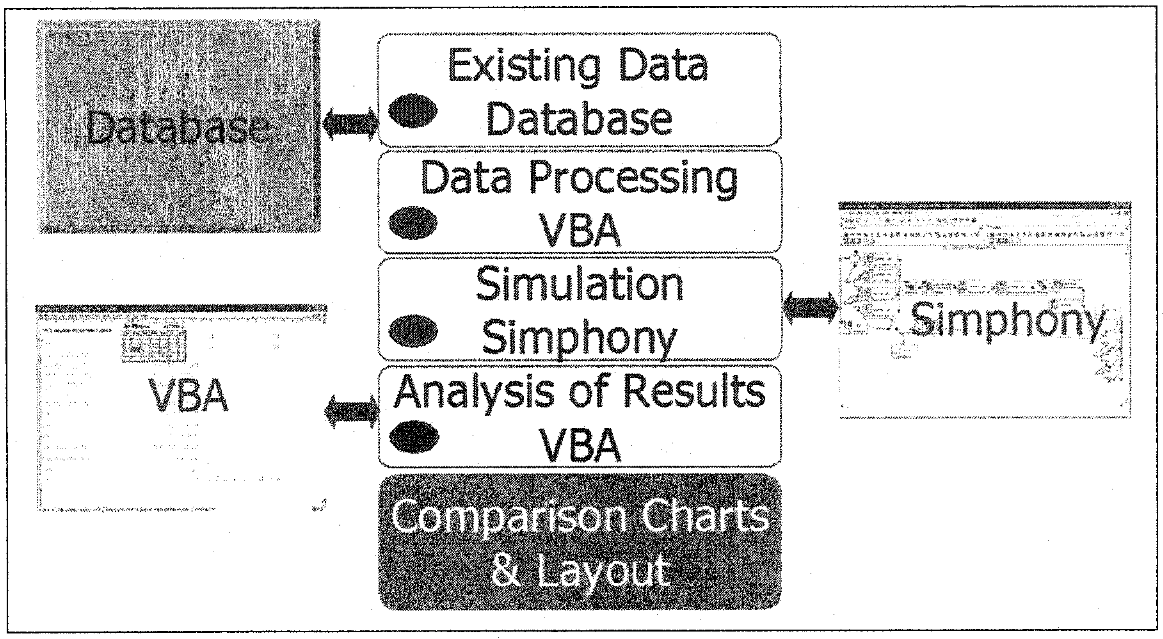


Figure 1 - System Integration

1.4 Organization of Thesis

In Chapter 2, a brief introduction to modularization (scheduling), the optimization of site layout, and simulation is presented. In Chapter 3, the design and development of the simulation-based technique for the module allocation problem is presented. In Chapter 4, a plan for implementation is presented followed by a case study approach in Chapter 5. Finally, Chapter 6 highlights the conclusions and recommends areas for future research.

Chapter 2. Overview of Modularization, Scheduling, Site Layout Optimization, and Simulation

2.1 Overview of Modularization

In conventional construction methods, building components, such as equipment, piping, valves, and platforms, are fabricated at each vendor's factory and then individually transported and installed in accordance with the installation plan. Customers are demanding cost reductions and shorter construction time, in order to meet this demand, modularization technology is required (Maru and Kawahata 2002). Modularization nowadays is viewed as an enhancement of projects including construction, industrial, and governmental. Maru and Kawahata (2002) have described modularization as a plant construction technique that simplifies installation work by using modules. A module is made up of pre-assembled components, such as equipment, piping, valves, and platforms. Those modules are then transported by rail, by ship, or by ground to their final location. The benefits of using modularization are numerous: opportunities for shorter schedules, lower cost, less risk, increased quality and greater construction flexibility for engineering, procurement, and construction (Burke, G. and Miller, R. 1998). Modularization is widely chosen due to the improvement this provides on schedule, quality, cost, and safety. However, customization cannot be completely eliminated from modularized design. There will always be site-specific issues necessitating modifications to reflect site-specific and client-specific requirements. The objective of modularization is to minimize the amount of time and effort devoted to customization on-site, and to reduce overall project cost (Schimmoller 1998). Preassembly and prefabrication is the wave of the future in industrial construction. Contractors and engineers around the world are realizing the benefits of cost savings, time savings, increases in work safety and equipment quality, and the increased production that may be achieved with modularization. Maru and Kawahata (2002) have identified four main areas as advantages of modularization:

1) Shorter construction duration: Having modules arriving to the construction site ready to be installed instead of performing the whole construction on-site has greatly improved construction duration. This improvement has been achieved due to the advantage of having modules under fabrication and assembly at the same time that on-site construction occurs.

2) Reduction of the number of workers at sites: Workers are divided into two job sites: the construction site and the module assembly site. Therefore, there is no longer the problem of having a large amount of workers on one site; each site is now less congested and the work conditions are more suitable for higher performances.

3) Improvement of safety and quality: Safety improvement is achieved due to the well-established safety controls that fabrication shops possess. Fabrication shops have a controlled environment; repetitive work allows workers to do the work faster and more safely. The module units are constructed in tight conditions. Therefore, a poor quality finish and the overall waste are minimized while savings are achieved. Having an efficiently designed and clean environment with good visibility permits workers to perform at their best level. Quality control is best executed in a fabrication shop ensuring that modules are finished correctly every time. Durability and reliability of modular construction is a reassuring factor for any owner.

4) Reduction of construction cost: Modular construction greatly reduces construction cost. The cost of labor off-site is lowered as is the total number of labor hours. These reductions are achieved by having fewer days working on a remote location, thereby lowering on-site administration costs. Performing hot work in the fabrication shop saves money since it is a type of work that can be costly in some locations. Finally, weather is a factor that influences the cost dramatically. Building the modules and having them shipped to the remote and rough weather location saves money as well as time.

Modular Construction in Northern Alberta

The Alberta oil sands are the world's largest producers of crude oil from sands and are the largest source producer in Canada. The oil sands are located near the city of Fort McMurray, Alberta and their operations are based on the Athabasca Oil Sands Deposit. The products depend strongly on modular construction facilities to meet its industrial needs. Maximizing the relocation of construction work hours away from Fort McMurray lowers costs, relieves the impact on base plant operations and on the community, and enhances construction safety. This relocation has been accomplished through an extensive modularization and preassembly program away from the worksite. A full module program results in significant direct and indirect cost savings. Labor outside of Fort McMurray is cheaper, compensation cost is reduced by increasing safety, savings in time reduce overall project costs, earlier market entrance, and savings in quality control, among others, assure direct and indirect cost savings. A formal Modular Design and Fabrication specification assure consistency of design on the project.

The Edmonton area has been the focus for module assembly and material marshalling due to its vast labor and transportation capabilities for northern Alberta. Module sub-assemblies fabricated outside the Edmonton area are shipped to Edmonton for installation in modules. The industrial sector of PCL is devoted to module construction in the Edmonton area. PCL's Pipe Fabrication and Module Assembly Yard are located in Nisku, Alberta. The Nisku PCL facility can produce up to 1000 tons of fabrication per month. Pipe and equipment racks, process skids, and building units are some of the modules assembled in PCL's module assembly yard (PCL 2003). PCL built over 300 modules in 2003 and expect its fair share of the upcoming large oil sands projects. The modules fabricated in 2003 are 100% complete, fully tested, insulated, fireproofed, and signed-off by Quality Assurance and Quality Control prior to shipment from Edmonton to Fort McMurray. All modules are road transportable. Their dimensional window is 24ft (7.32-m) wide, 29ft (8.84-m) high (loaded height), and multiples of 20ft (6.10-m) long. Weight limitations for highway transport are determined based on bridge capacities, transporter configurations, and seasonal highway load restrictions. These projects are an example of achievement and success within the modularization industry.

2.2 Scheduling and Site Layout Optimization

Scheduling Uncertainty Simulation and Optimization: Many subcontractors believe that real savings in time and money are found only in actual construction rather than through the application structured procedures for construction project management such as cost estimation, planning, scheduling, and/or control (Hegazy and Ersahin 2001). One of the most important tasks of a project manager is to optimize the construction schedule even when the total duration has already been determined. To achieve this goal, the project manager must consider a mathematical model in which the constraints and limitations may be more fully considered (Li 1996). All the projects have a certain degree of uncertainty in their executions. It is impossible to know with certainty and in advance which factors will play a roll in determining the duration of a project. Therefore, uncertainty is a huge factor influencing the performance of a project and its final success (Laufer 1996). Contingency plans are commonly done to take into account the reality of the uncertainty, the execution of these plans depends on several conditions. In spite of the diverse factors that influence a project, formal techniques for incorporating indeterministic conditions into scheduling have been recently developed, although they have not proved a popular choice. The interpretation of scheduling results as being a function of the project's probability and the need to use computers for certain of the available techniques have contributed to the overall lack of dissemination in scheduling research. For several reasons effective schedule optimization has not been achieved due to the complexity of projects, the difficulties associated with modeling all aspects combined, and the inability of traditional optimization tools to solve large-size construction schedule problems (Hegazy and Ersahin 2001a).

Site Layout Optimization: Yeh (1995) defined construction site layout as the design problem of arranging a set of predetermined facilities on a set of predetermined sites, while satisfying a set of layout constraints and optimizing layout objectives.

Construction site layout is essential to any project and has a significant impact on the economy, safety, and other aspects of a project (Mawdesley et al. 2002). Efficient layout planning of a construction site is fundamental to any successful project undertaking. The project manager or planner usually performs the task of preparing the site layout based on his or her own knowledge and expertise (Osman et al. 2003). Site layout planning is a complex problem that researchers have attempted to solve using a variety of optimization-based and heuristic-based techniques (Hegazy and Elbeltagi 1999). The task of site layout has a very dynamic relationship with the other preplanning tasks such as schedule development, selection of construction methods, procurement planning, workforce planning, material planning, equipment planning, and financial analysis (Cheng and O'Connor 1996). According to Hegazy and Elbeltagi (1999), the basic consideration in an effective site layout plan is the smooth and low-cost flow of materials, labor, and equipment within the site, in addition to satisfying the various work constraints and safety requirements. Hegazy and Elbeltagi (2001) suggested that layout planning could be viewed as a complex optimization problem resulting in many engineering applications ranging from the layout of manufacturing plants to the design of computer chips. They also pointed out that early models were based solely on mathematical optimization techniques and were successful in laying out only a single or a limited number of facilities due to the complexity of problem formulation.

Tan and Leung (2002) indicate that the layout planning of construction site facilities has a significant impact upon productivity, costs, and duration of construction. They also mentioned that although facilities layout planning (FLP) is such a critical process in construction planning, a systematic analysis of construction site layout is always difficult because of the presence of a vast number of trades and inter related planning constraints. The authors also noticed that practitioners of the construction industry lack a well-defined approach in construction site layout planning. For these reasons, the practitioners stated that FLP optimization using the scientific approach is nearly impossible to achieve. The FLP of construction sites has been carried out mainly through human judgment. Because of this human involvement, there are no conditions present that will lead consistently to the same result. To overcome the above problems, researchers have used mathematical and computing techniques in an attempt to arrive at an optimal solution (Tam and Leung 2002).

2.3 Simulation

Simulation can be simply defined as building a mathematically logical model of a system and using the model for experimentation using a computer. However, simulation in its broadest sense means imitating or representing reality (generating events before they occur) (Oglesby et al. 1989). The ideal objective of computer simulation is to optimize system performance. Creating a simulation involves the following steps (Web 1, 2004):

- Defining the system (well-defined boundaries)
- Modeling the system (system of equations, graphical modeling)
- Input and output analysis
- Validation/verification

Computer simulation is a valuable experimentation tool well suited to the study of resource-driven processes. It gives the analyst insight into resource interaction and may assist in identifying those significant factors in problematic domains. Simulation allows the modeler to experiment with and evaluate a variety of scenarios. Normally, such experimentation and study would be too costly to be carried out in the real world.

When dealing with the construction of facilities such as highways and buildings, construction engineers confront certain aspects of production that an industrial engineer faces daily. Industrial production can be done repetitively due to the characteristics of the products and of course to its production volume. Construction engineers are involved in developing and efficiently designing productive construction methods and processes. The uniqueness of the construction projects involved and the apparent lack of repetition throughout are perhaps reasons why the concept of studying work processes did not receive much attention until the late

1960s. At this moment it was recognized that although projects are typically unique, many construction processes such as earth moving, dewatering, and tunneling are repetitive and amenable to closer investigation. With the emergence of computer technology, the application of more sophisticated analytical methods has become increasingly accessible. Simulation of construction processes for establishing the anticipated levels of production and to solve certain problems related to the randomness of construction operations has become more widely accepted as a tool available for use in planning and estimating (Web 1, 2004). Simulation has the great advantage of predicting levels of production and of solving the randomness of construction operations.

Adapted by Teicholz in 1963, the "link node" model was the first method used.Au et al. (1969) suggested a construction bidding game in the late 1960s. This application is among the very first random number method related to gaming. It is still used at several universities for teaching purposes. Halpin developed the CYCLONE format at the University of Illinois (1973). CYCLONE is now the basis for numerous construction simulation systems. CYCLONE simplified the simulation modeling process and became accessible to people without a construction simulation background (Web 1, 2004). In 1973, Halpin and Woodhead developed at the University of Illinois the CONSTRUCTO project management game integrating the effects of weather and labor productivity into the management of projects in a network format (Halpin and Woodhead 1973). Another simulation tool (Cost Control Simulation - CCS) was developed by Borcherding (1977) at the University of Texas. CCS's objective was to develop a computer model for analyzing the financial aspects of a construction project. More recently, the concepts of the bidding game and the project management format have been integrated into an educational game, Superbid, at the University of Alberta (AbouRizk 1992). One of the most recent simulation tools is *Simphony* developed by AbouRizk and Hajjar (1998).

Simphony: The effective use of simulation within the industry is best done through the specialization and customization of models. Special Purpose Simulation (SPS) is a proven principle that can lead to the effective transfer of simulation knowledge to the construction industry. *Simphony* simplifies the SPS tool development process and standardizes the simulation, modeling, analysis, and integration features of such tools. It provides an environment that tailors to the needs of both novice and advanced simulation tool developers and users (Hajjar and AbouRizk 1999). *Simphony* is a Microsoft Windows-based computer system developed with the objective of providing a standard, consistent, and intelligent environment for both the development as well as the utilization of construction SPS tools. Developers can use *Simphony* to implement highly flexible simulation tools, which support graphical, hierarchical, modular and integrated modeling. Users have access to a single program, which allows them to build simulation models in an intuitive and user-friendly manner (Hajjar and AbouRizk 2002). Results can be viewed as part of the graphical user interface or exported for use by external systems such as estimating and scheduling programs. *Simphony* is characterized by the following functions:

1. Modular and hierarchical modeling for the representation of complex and large construction projects,
2. Both general purpose modeling constructs as well as specialized templates for specific construction methods,
3. Extension of SPS tools through the construction of models based on several templates,
4. Generation of custom output results in tabular and graphical formats,
5. Automated generation of externally accessible project planning data in a standard format,
6. Script-based modeling for accommodating advanced users wishing to bypass the graphical user interface, and
7. Storage and retrieval of commonly-used simulation model structures in the User Model Library.

Simphony Overview and Basic Features: (*Simphony*'s User Guide AbouRizk 2000)
Simphony represents an evolution in computer simulation and its integration into the construction industry. It is the result of over five years of research in the application of simulation-based planning techniques in the industry. Simphony consists of a foundation library, as well as specialized computer programs that allow for the development of new construction simulation tools in an efficient manner. *Simphony*'s promise is that, as a user, there is no need to posses any simulation background in order to take advantage of the benefits of simulation. When building models, there is access to a domain-specific set of building blocks, denoted "Modeling Elements". This means that the creation of a simulation model is done using a library of modeling elements with names to relate. There is a large library of modeling elements that are available with the base distribution of *Simphony*. If any of the existing modeling elements are not flexible enough to meet certain modeling needs, or if new modeling elements are needed to be developed for different construction operations, then a developer can extend the library.

Modular and Hierarchical Modeling: The main model building block in *Simphony* is the Modeling Element. The user builds a simulation model in *Simphony* by creating instances of modeling elements that resemble real components of a construction system, and linking them together in ways similar to those that exist in a real system. For representation of complex and large construction projects, *Simphony* provides a hierarchical modeling feature. A project can be represented by an abstracted model at the higher level that contains a limited number of modeling elements and relations. At a lower level, each of these elements can have its own child model, which represent the sub-system working inside that element. The number of these hierarchical levels is only limited by the computer system's resources.
General Purpose vs. Special Purpose Simulation (SPS): Simphony supports both general purpose modeling constructs (e.g. CYCLONE) which can be used to model different construction processes, as well as specialized templates for specific construction methods (e.g. Earth-moving and aggregate production) which are suitable for users with little simulation background.

Integration of SPS tools: Simphony allows the extension of specialized SPS tools through the construction of models based on several templates.

Custom Output Results: Simphony modeling elements can generate custom output results in the form of tables and graphs.

Automated Generation of Project Planning Data: Project planning data regarding costs and time can be automatically generated by *Simphony* during simulation and presented to the user in a standard format.

Script Based Modeling: Script based modeling allows advanced users wishing to bypass the graphical user interface to write a script to be processed by *Simphony* to handle advanced simulation behaviors.

User Elements: Simphony allows storage and retrieval of commonly used simulation model structures, known as "User Elements", in the User Elements' Library. These elements represent certain modeling elements with complex internal structures or special parameter settings that are commonly used.

The application of simulation: Senior (1995) proposed an algorithm built on the Cyclic Operation Network Technique (CYCLONE) (Halpin 1973; Halpin and Riggs 1992), a discrete-event simulation method oriented to construction applications, to compute task late-time and float information. Since late-time information has been used in the critical-path method (CPM), the availability of this information in a simulation technique could make the application of simulation more commonplace in construction practice. AbouRizk and Hajjar (1998) presented an approach to facilitate the adoption of simulation by the industry as they recognized the limited use of simulation by construction industry. This approach was based on special purpose simulation. They defined SPS as a computer-based environment built to enable a practitioner who is knowledgeable in a given domain, although not necessarily in simulation, to model a project within that domain in such a way that symbolic representations, navigation schemes within the framework, the creation of model specifications, and reporting functions are completed in a format native to the domain itself. The basic philosophy of special purpose simulation is that systems should be built for a specific target group. This philosophy obviously produces relatively restrictive tools, which can only be used within the intended application domain.

Chapter 3. Proposed Methodology

3.1 Introduction

Based on the review of modularization, scheduling, site layout optimization, and simulation, it is evident that an automation methodology is required to solve the module assembly yard scheduling-layout problem. A simulation model seems to be an appropriate approach. The approach involves the following physical and logical constraints as well as the heuristic rules that superintendents use in actual practice.

Physical and logical constraints:

- Module yard layout is fixed,
- Number of workers is fixed,
- Modules may only be shipped when the space in front of them is totally empty, and
- Maximum number of shipments per day is fixed.

Heuristic rules:

- After completion, modules may wait a maximum of "n" days for shipment,
- Module routing for allocation follows certain preferences including the type or the size,
- Once a module has been routed to a specific area, the work flow will be front to back (starting from bay # 1 to bay # n),
- Duration and dates are fixed, and
- Priority logic is employed – module with least amount of float will be given higher priority for assembly.

General purpose simulation constructs are used to model these constraints and heuristic rules.

3.2 Proposed Methodology Main Process

System process: Simphony (Hajjar and AbouRizk 2002), a SPS computer-based environment, was used to create the simulation model for this Module Assembly Schedule. This approach introduces a newly developed methodology, which utilizes a simulation technique for module scheduling and for optimizing the assembly yard utilization. The model integrates a database, simulation (*Simphony*), and Excel's built-in visual basic applications (VBA). The raw data provided by the company is stored in the database. This raw data contains those inputs used by the simulation: yard size, yard layout, yard capacity, number of bays, module types, module sizes, durations, early starts, and planned shipping dates. The priority for each module is calculated based on planned shipping dates, early start, and the actual date when the simulation takes place. The criteria ruling the allocation of the modules (physical and logical constraints, and heuristic rules) is incorporated with the simulation. The results include a tabular format containing start, finish, and shipping dates, a comparison chart between the previously planned schedule and the simulation schedule, and a module allocation layout chart. The system process is illustrated in Figure 2.

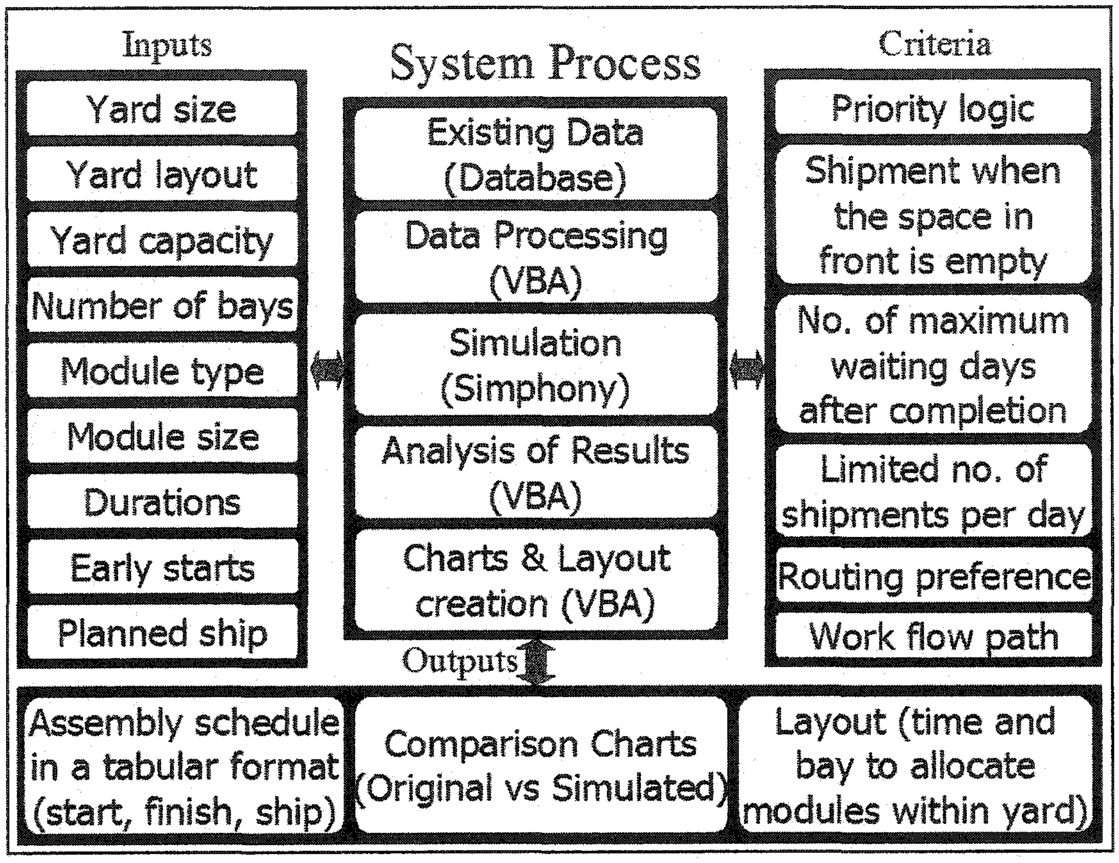

Inputs
System Process
Criteria
Yard size
Yard layout
Yard capacity
Number of bays
Module type
Module size
Durations
Early starts
Planned ship
Existing Data (Database)
Data Processing (VBA)
Simulation (Simphony)
Analysis of Results (VBA)
Charts & Layout creation (VBA)
Priority logic
Shipment when the space in front is empty
No. of maximum waiting days after completion
Limited no. of shipments per day
Routing preference
Work flow path
Outputs
Assembly schedule in a tabular format (start, finish, ship)
Comparison Charts (Original vs Simulated)
Layout (time and bay to allocate modules within yard)

Figure 2 - System Process

System Components: The database functions as the mediator for the proposed system. Simphony reads the inputs provided by the database (inputs are both provided directly by the company and calculated using Excel-VBA), the list of results are then shown in the Simphony model itself and back in the database, and the comparisons charts and layout are auto-generated using Excel-VBA. The system components are illustrated in Figure 3.

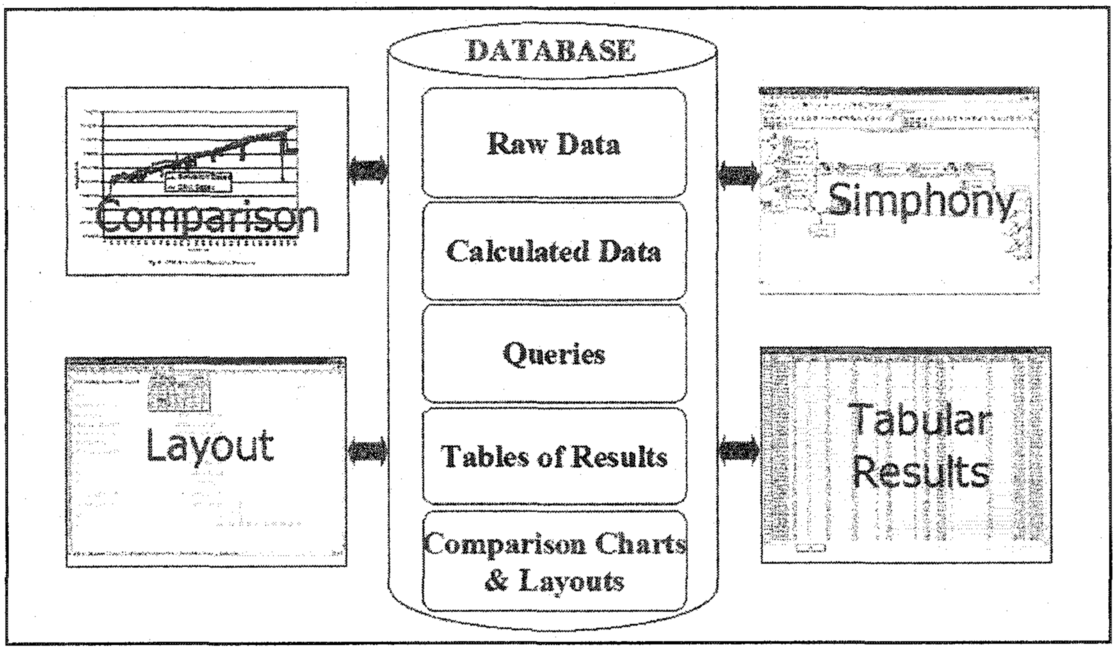


Figure 3 - System Components

System limitations: Banks (1998) showed that scheduling problems generally require restrictive assumptions in order to be solved. The following seven assumptions are adapted when scheduling module assembly yards:

1. Each module is an entity. Only one module may be processed at a time in a specific space.
2. There may be no cancellations. Once the fabrication of a module has started, it has to be completed before another module can start its fabrication on the same space.
3. The fabrication of modules must be continuous.
4. A specific space within a bay may not fabricate more than one module at a time.
5. Bay's space is available throughout the scheduling period.
6. The technological constraints are known in advance and are immutable.
7. The following four items are known and fixed:
    a. Number of modules to assemble
    b. Number of bays
    c. The fabrication times are known and fixed.

d. All other quantities needed for defining a particular problem.

An operation's start and finish times for each job waiting to be processed must satisfy certain technological constraints, and must accomplish optimality. Each job is defined by its operations, processing times, and due dates. In a deterministic scheduling problem, numeric quantities such as processing times and due dates are assumed to be known in advance. However, unknown (stochastic) parameters are inherently subject to randomness. Scheduling a module assembly yard is a difficult task, which involves a number of key factors governing the decision of module allocation including the type, the size, the start date, the duration, and the planned ship date of the modules. Currently this process is carried out manually and based solely on the experience of the foremen. The proposed methodology presented in this thesis was applied to a PCL module yard located in Nisku, Alberta.

Yard characteristics: As shown in Figure 4, the PCL Module Facility is divided into 4 areas (i.e. A, B, C, and D). Each area contains between 4 and 14 bays (see Table 1). In total there are 36 full bays and 3 half bays. Each full bay is 260-feet (79.25-m) long where Half Bays are 130-feet (39.62-m) long.

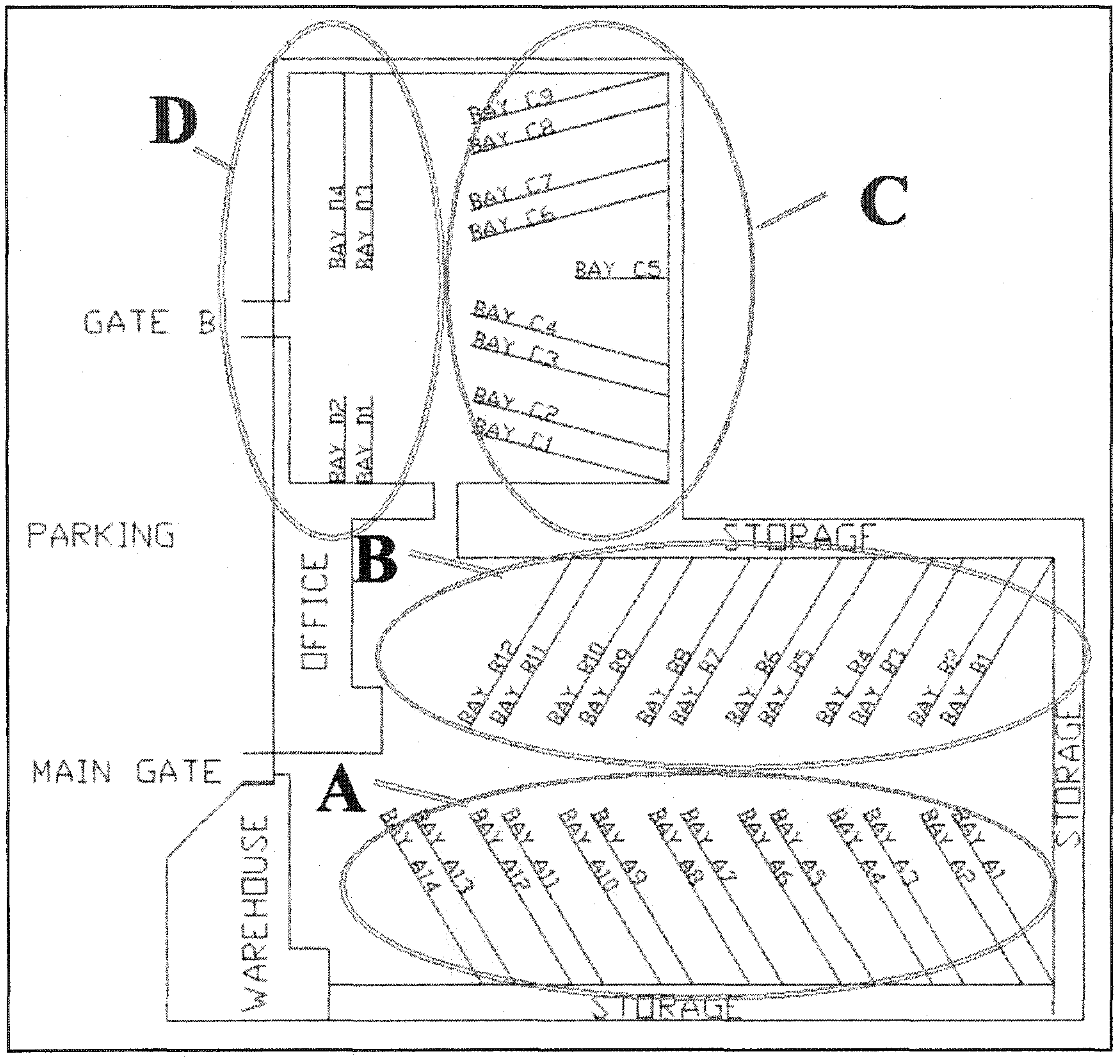


Figure 4 - PCL Module Yard Layout

Table 1 – Number of Bays per Area

	No. of Bays	Bay Number
Bay Area "A"	14 Full Bays	Bay A1 to A14
Bay Area "B"	12 Full Bays	Bay A1 to A14
Bay Area "C"	8 Full Bays	Bay C1 to C4 Bay C6 to C9
	1 Half Bay	Bay C5
Bay Area "D"	2 Full Bays	Bay D1 to D2
	2 Half Bays	Bay D3 to D4

The number of modules in a bay is a function of the available sizes and types of modules present; Table 2 lists the quantity of modules that each bay can contain. In addition, Table 2 lists the module yard capacity ranging from 75 to 186 modules, which is also a function of the module types and sizes.

Table 2 - Module size Types per Bay and Yard Capacity by Module Size

Module Size Class	Module Size (Feet)	Quantity		Capacity of Yard
		Full Bay	Half Bay	
A	0' – 20'	10	4	372
B	21' – 40'	5	2	186
C	41' – 60'	3	1	111
D	61' – 80'	3	1	111
E	81' – 100'	2	1	75
F	101' – 120'	2	1	75
G	121' – 140'	1	N/A	72
H	141' – 160'	1	N/A	36

Module Classification Procedure

To simplify the process of module assignment, modules are categorized into five classes based on their type (see Table 3).

Table 3 - Module Type Classification

Type Class	Module Type
"CT"	Cable Tray
"EM"	Equipment
"M"	Miscellaneous
"PM"	Pipe Rack
"S"	Structural Only

The specifics of the case were analyzed as invariable constraints for the project schedule. Those constraints need to be determined by a group of experts such as the scheduler and the project manager. The simulation allows us to test different scenarios in which the constraints assumed at the beginning of the project are changed and proper evaluation of each of those scenarios is undertaken to determine a better combination of constraints to be used once it is possible for the company to make those changes. The following assumptions apply to the first scenario:

- Module yard layout is fixed,
- Resources (man-hours) not taken into account,
- Modules may only be shipped when the space in front of them is completely empty,
- Maximum number of shipments per day is six,
- After completion, modules may wait a maximum of five days for shipment,
- Module routing for allocation follows certain preferences; in this case they are routed according to their type,
- Once a module has been routed to a specific area, the work flow will be front to back (starting from bay # 1 to bay # n),
- Duration and dates are fixed, and
- Priority logic is employed; that is the module with the least amount of float will be given higher priority for assembly

Simulation Model Development.

1) Setting the database to integrate with the simulation module

The database contains the information necessary for the simulation model. It consists of fields (columns), which will be expressed as "attributes" in *Simphony*, and records (rows) representing modules, which will be expressed as "entities" in *Simphony*. The first fields contain the data needed for the simulation: ACT (Module ID), durations (each subtasks has a different duration), TypeClass, UnitsRequested (attribute derived from the size type and the number of modules that each bays could contain), number of workers (each subtask has a different number of workers), EarlyStartDate, ESA (attribute that specifies whether the module has begun or not), PlannedShipDate, EFA (attribute to specify whether the module has finished or not), WaitingDays (attribute that specifies the maximum allowable number of waiting days that a module can wait for shipment), and priority (attribute that define the priority of each module at the time of fabrication). The information contained in these attributes is known almost completely in advance. However, the attribute "UnitsRequested" was derived from the size type and the number of modules that each bay could contain. Space is represented as resources. To standardize all the bays, a fixed number of resources (space) is calculated. This fixed number is calculated based on the quantity of modules that a bay can contain depending on the module type sizes. The number of modules that a bay can contain ranges from ten of the smallest modules, size type "A" (0' – 20'), to only one of the largest modules, size type "H" (141' – 160'). Based on the number of modules that a bay can contain (quantity) depending on their size types, the minimum number of virtual resources needed to simplify the simulation process has been identified. This number has been found to be 30. A module size type "H" requires 30 units (resources) for fabrication, which means that it requires the whole bay, and a module size type "A" requires only 3 units (resources) for fabrication, meaning that it only requires 1/10 of a bay. Those virtual resources do not represent exact units such as feet or meters, etc. Table 4 shows the number of virtual resource units ("UnitsRequested") that a size type needs based on the number of resources that a bay can contain:

Table 4 - Units of Resources per Size Type

Size Type	Quantity	Units Requested	No. Resources per Bay (30)
A (0' – 20') Modules	10	3	10 * 3 = 30
B (21' – 40') Modules	5	6	5 * 6 = 30
C (41' – 60') Modules	3	10	3 * 10 = 30
D (61' – 80') Modules	3	10	3 * 10 = 30
E (81' – 100') Modules	2	15	2 * 15 = 30
F (101' – 120') Modules	2	15	2 * 15 = 30
G (121' – 140') Modules	1	30	1 * 30 = 30
H (141' – 160') Modules	1	30	1 * 30 = 30

Finally, the attribute, "priority," is calculated based on whether or not the module has already started. If the module has not started yet then the priority is equal to "P." "P" is equal to 500 - (Float "F" + Absolute Value of the minimum of the floats, abs (min (F)). "F" is equal to (PlannedShipDate "PSD" – EarlyStartDate "ESD" – Duration "D"). If the module has already started, then a calculation must be performed to ensure that its priority is higher than the priorities of the modules that have not yet started. The calculation by adds the difference between the day when the simulation is performed, "TD," and the EarlyStartDate, "ESD," to the maximum of the "P" among the modules that have not started their fabrication yet, max (P). This equation will assign the highest priorities to those modules that started at the earliest time. "500" has been chosen as an arbitrary number to ensure that "P" remains positive. The calculation of the priorities is expressed in the following equations:

<u>Calculation of priority (P) for modules that have not started fabrication yet:</u>

$P = 500 - (F + \text{abs}(\min(F)))$

Where:

$F = PSD - ESD - D$

F = Float

PSD = PlannedShipDate

ESD = EarlyStartDate

D = Duration

<u>Calculation of priority (P) for modules that have already started fabrication:</u>

$P = (ESD - TD) + \max(P)$

Where:

ESD = EarlyStartDate

TD = Today (day when the simulation is performed)

Max (P) = Maximum "P" among the modules that have not yet started fabrication

All these calculations are done in Excel and exported automatically to the database. There are other seven fields needed to perform schedule updates. These fields are:

Bay (attribute that specifies in which bay where is the module being built), BaySize (number of total space units that the bay where the module is being fabricated has), Task (subtask in which the module is currently in progress), NoOfDaysAlreadyOnTask (number of days since the subtask started), NoOfDaysSinceStart (number of days since the fabrication process started), No_of_Modules_Behind (number of modules behind the current module in the same bay), and No_of_Units_Occupied_Behind (number of space units occupied by the module or modules behind the current module). Since the simulation is intended to be capable of performing regularly updates in these seven fields the user must enter the information regarding the modules that are under fabrication at the time of the schedule update. The rest of the fields only have the attribute name but no data at the beginning of the simulation. Once the simulation has run, those fields will use the new processed data. These fields include Space_in_frontr (attribute determining the space available for hosting more modules), Start (the expected start date for the process and the starting time of each subtask), Finish (the expected finishing date for the process and the finishing time of each subtask), and Shipping (the expected shipping date of the module).

2) Identifying Finished Modules

The first step within the simulation is to identify the modules that have started and also finished fabrication by the date when the simulation takes place. The modules that finished their fabrication process before the simulation started simply record the starting time and finishing time since there is no need to process them again.

3) Identifying Started Modules (Modules that are already under fabrication)

The modules that have already started their fabrication process need to be placed on their respective bays where they are been fabricated. With the information about Bay, Task, NoOfDaysAlreadyOnTask, NoOfDaysSinceStart, No_of_Modules_Behind, and No_of_Units_Occupied_Behind the correct placement of each module is done. These modules will now be placed on their respective bays, ensuring that the simulation will start placing modules only where bays have empty assembly space. The modules that have started their assembly process previously to the simulation will join the simulated fabrication process at the subtask in which they are currently under fabrication.

4) Identifying Type Class

The modules that have not yet started fabrication are routed to a particular area, which will depend on the Type Class. In this case, it was assumed that the TypeClass "PM" are preferably routed to Bay area "A", TypeClass ("EM") will be routed preferably to bay area "B," while TypeClass "CT" is preferably routed to bay area "C," TypeClasses "M" and "S" are both preferably routed to bay area "D." In addition, routing could also be a function of "UnitsRequested" (size).

5) Looking for Space to Assemble

Once a module has been routed to a specific area it is necessary to check whether there is a bay that could contain the module during its assembly process as well as the availability of labor to perform the first subtask of the assembly process. More than one module may arrive and request resources for assembly at the same time; in such a case, the assembly priority for these modules is based on the value of each module's attributed "Priority." The modules request resources based on the highest priority. Modules request an available bay for assembly and "Manhours" for the first subtask of workers. The space is assigned to that module, which possesses the highest priority and that will satisfy all the assembly constraints. This location will be the closest empty space available for assembly within a bay area starting from the number 1 to the last number; (a module routed to Bay Area "A" will be assigned to the closest empty available space among A1 to A14 starting by A1, see Figure 5). The number of resources (space) that each module requests is determined by its attribute "UnitsRequested." Four auxiliary resources types are needed in order to keep track of vital information used during the assembly process such as the finishing times of the modules behind in the same bay, the number of modules already in the bay, and the amount of space available for more modules in that bay. These new resources types are "EndBackModule," "FinishCurrentModule," "ModuleCounter," and

"SpaceInFront." "EndBackModule" keeps track of the module's finishing time in the back of a bay. "FinishCurrentModule" keeps track of the module with the latest finishing time in a bay. "ModuleCounter" records the number of modules in one bay. "SpaceInFront" tracks whether there is available space in front of the current module to fabricate more modules. "EndBackModule" and "FinishCurrentModule" have to be initialized at the beginning of the simulation with a very high value to ensure that when the bay is empty it frees up resources for assembly. This high number, normally set at 1000, ensures that no matter which module arrives to a bay, its finishing time will be shorter than the finishing time of the virtual module in the back of the bay. This association only occurs when the bay is empty, that is, when the bay has at least one module, then the values given are those expected finishing times. Once a module is assigned to a bay, those values will be updated with the simulated finishing time. At this point, each module requesting space for assembly will also check the finishing time of the module to ensure that it does not exceed the finishing time recorded for the module in the back of the bay and for the module with the highest finishing time in that bay. When a bay is completely empty again, the values for "EndBackModule" and "FinishCurrentModule" will take the high initial value that they had at the beginning of the simulation. "ModuleCounter" is initialized with a value of zero since the bays are empty at the beginning of the simulation. Each time a module is placed in a bay, the "ModuleCounter" increases by one unit and every time a module leaves the bay the "ModuleCounter" is decreased by one unit. "SpaceInFront" takes the initial value of one (1), representing the empty space available for modules in a bay. The initial value is zero (0) when all the resources in the bay are occupied and there is no more available space to host more modules (see Figure 5).

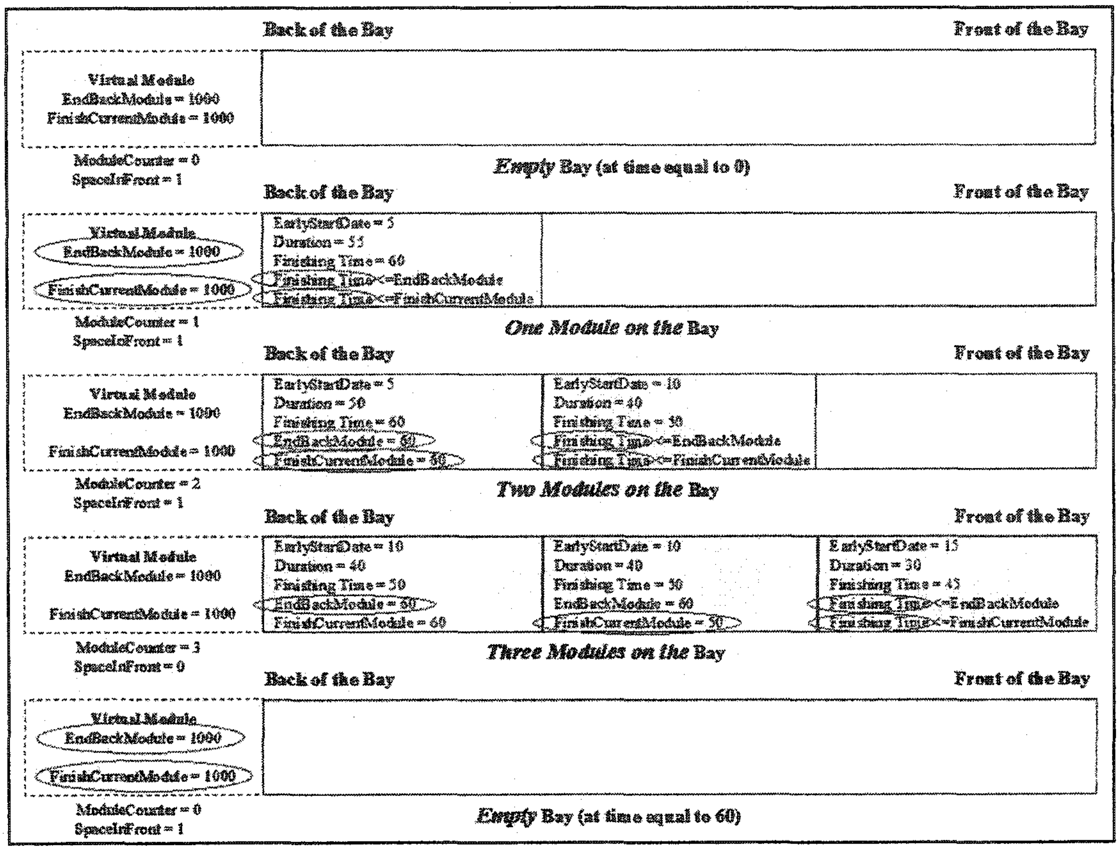


Figure 5 - Auxiliary Attributes Used When Processing Modules on a Bay

6) Assembly Process

At this point four more auxiliary attributes are used. The "Assembly" attribute keeps track of the number of space units for assembly expressed as resources that have been utilized for each module. "Space" records the number of space units expressed as resources left in front of the module in the bay in which each module is assembled at the time of assigning the resources for assembly. This amount of resources must be available before shipping a module once it finished assembling (ensuring that all the required space in front of the module for shipment is empty). The "Total" attribute represents the space units as resources to be released once a module has been shipped. The total amount is the sum of "Assembly" plus "Space," which is not necessarily equal to the initial total amount of resources in the bay. This discrepancy is due to the possibility of having other modules in process behind the last module assigned in the same bay. The "Bay" attribute takes the name of the bay in which the module has been processed. When the module is supplied with the resources requested and space for assembly, the module continues with the simulation process and the "Start" attribute is assigned the value of "Simulation Time" that represents the time at which that module will actually begin its assembly process. To simulate the assembly, the main task has been divided into six subtasks (structure, piping, cable tray, electrical heat tracing (EHT), insulation, and fireproof) integrating all the assembly process. The durations of those subtasks were given as a fixed value; however, the subtasks could also use random values generated through a probabilistic distribution based on statistical data. Once the module simulation has begun the auxiliary attribute will assume the initial value of zero (0) takes the value of one (1). The starting time of the first subtask is also recorded. Once the subtask is completed, the finish time of that subtask is recorded.

7) Subtask Process

Once the first subtask has been completed the finishing time of that subtask is recorded and the workers needed to perform the task are released and ready to be assigned to other modules. The module may proceed onto the second subtask. The number of workers needed to perform the second task is checked. If sufficient workers are available the second subtask may begin. At this point the subtask start time is recorded. When the subtask is completed, the finishing time of that subtask is recorded and the workers needed to perform it are released and ready to be assigned to other modules. This process is the same for all subsequent subtasks; however, at the end of the last subtask the finish time of the module assembly process is also recorded.

8) Checking Space Available for Shipping

Once the assembly process has been completed the module is ready to be shipped. In order to satisfy the condition that a module will only be shipped if there are no other modules being fabricated in front of it within the same bay, the model has to look for available empty space in front of the module in that bay. This condition is satisfied through a comparison of the number of resources available in that bay with the value of the "Space" attribute. If both values are equal, then the module will go on with the simulation process, such an action means that there are no other modules under fabrication or waiting for shipment in front of this module.

9) Requesting Shipment

Since it was assumed that only a limited number of shipments could occur on the same day, another resource named "Ship per Day" is required, which would hold the number of available resources. A module is ready to request for shipment once it has finished its assembly process and has no other modules in front of it. The request of one resource out of those available for shipping takes place. If the resource is available; then the module will finish its process; if there are no resources available, then the module will wait until there is a resource available for it.

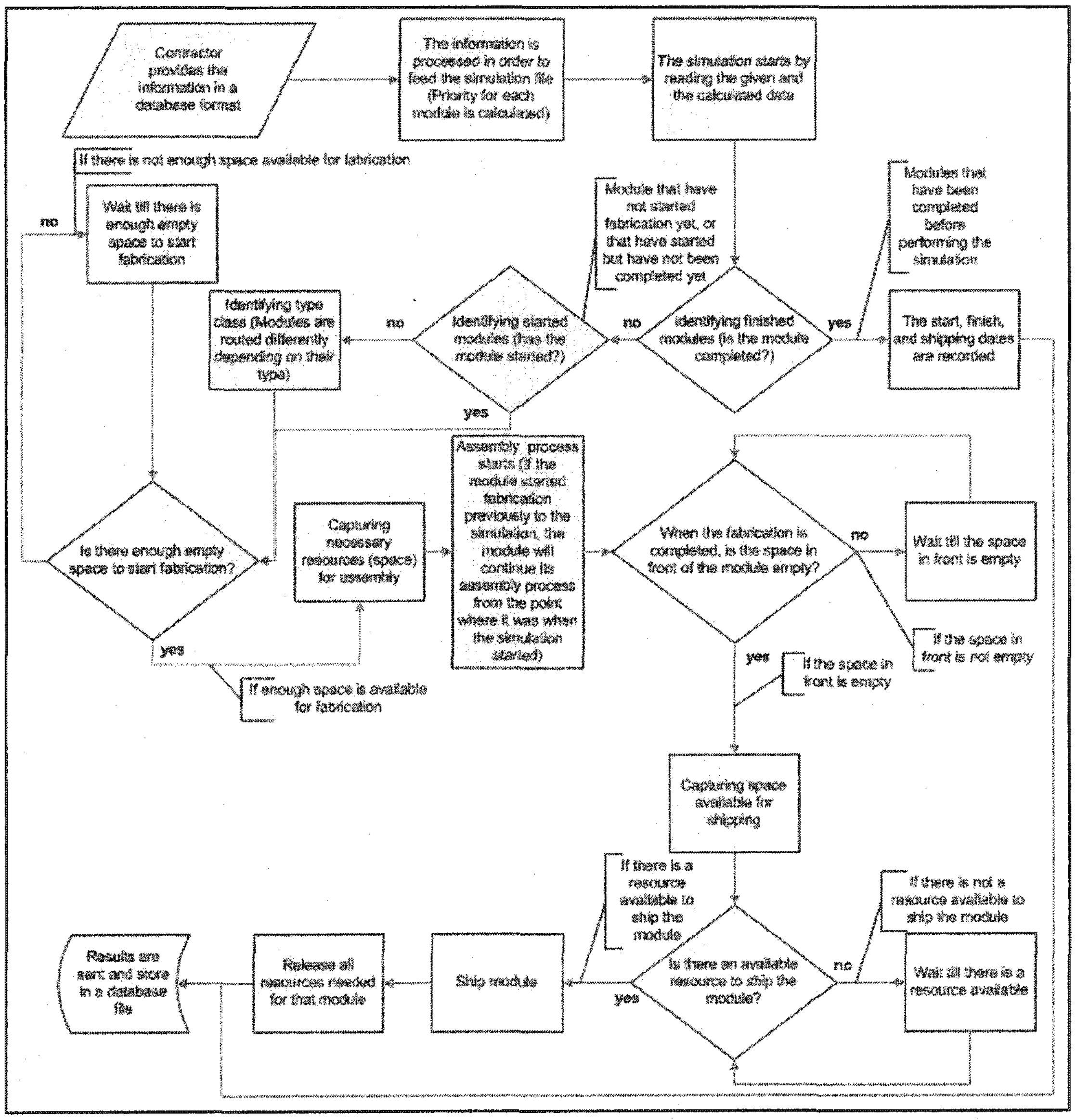
10) Shipped Modules

When a module is shipped, the "Shipping" attribute records its shipping time. The number of available resources of the bay where the module was processed is set to be equal to the value of the attribute "Total" ("Assembly" + "Space"). At this point a task with a duration equal to one (1) is processed; resources can be released thereafter. This process is undertaken because it is assumed that the shipping of a module will last one day and no modules will start assembly on that bay until the next day.

11) Results

As the simulation takes place, attributes are being input into a table. At the end of the simulation the information is sent back to the database. The database applies a query to change the dates into a date format. These results are plotted against the previously planned CPM schedule with which they are compared. An auto-generated layout chart is also created.

Based on reading the data provided, data processing, and simulation process to storing the results in a database, Figure 6 summarizes the process used by the model from.



Contractor provides the information in a database format
The information is processed in order to feed the simulation file (Priority for each module is calculated)
The simulation starts by reading the given and the calculated data
If there is not enough space available for fabrication
no
Wait till there is enough empty space to start fabrication
Module that have not started fabrication yet, or that have started but have not been completed yet
Modules that have been completed before performing the simulation
Identifying type class (Modules are routed differently depending on their type)
no
Identifying started modules (has the module started?)
no
Identifying finished modules (is the module completed?)
yes
The start, finish, and shipping dates are recorded
yes
Assembly process starts (If the module started fabrication previously to the simulation, the module will continue its assembly process from the point where it was when the simulation started)
Capturing necessary resources (space) for assembly
Is there enough empty space to start fabrication?
When the fabrication is completed, is the space in front of the module empty?
no
Wait till the space in front is empty
If the space in front is not empty
yes
If the space in front is empty
yes
If enough space is available for fabrication
Capturing space available for shipping
If there is a resource available to ship the module
If there is not a resource available to ship the module
Results are sent and store in a database file
Release all resources needed for that module
Ship module
yes
Is there an available resource to ship the module?
no
Wait till there is a resource available

Figure 6 - Model Description

3.3 Conclusion

The proposed methodology, a simulation-based technique, is a reliable approach to solve the module assembly yard scheduling-layout problem since it provides the industry with an automated methodology.

Chapter 4. Implementation

4.1 Introduction

This newly developed simulation technique for module scheduling and optimization of the assembly yard utilization described in this thesis will be implemented using Simphony (AbouRizk and Hajjar 1998). The implementation retrieves the information from the database, computes the necessary inputs for the simulation, builds the simulation model, and reports results.

4.2 Special Requirements of Module Assembly Scheduling Process

The assembly of pipe spool modules involves many uncertain factors, which complicate its scheduling process. These factors also pose a challenge for the scheduler in producing an efficient schedule, which optimizes the use of the space (module yard) as well as the human resources involved in the assembly process while meeting clients' delivery dates. Given the relatively fast production cycle of module assembly, the scheduling process must be carried out frequently and requires advanced automated tools to perform this modularization task. Modularization is carried out in module yards and once the modules are completed they are shipped to the industry plants. Without a proper scheduling system, it is very difficult to maximize yard usage and to improve delivery dates. Since module assembly in a yard depends on physical and logical constraints, a method in which these constraints are built-in during the scheduling process would be beneficial to the industry in so far as it saved time while panning the schedule. It is also beneficial in so far as it avoided mistakes while placing modules in the yard.

The scheduling technique developed in this thesis was considered to be a prime candidate for a simulation model. The following seven main challenges were identified:

1. *Integrating the Database with Simphony:* The number of records for this simulation is vast and the simulation records must be updated frequently. As

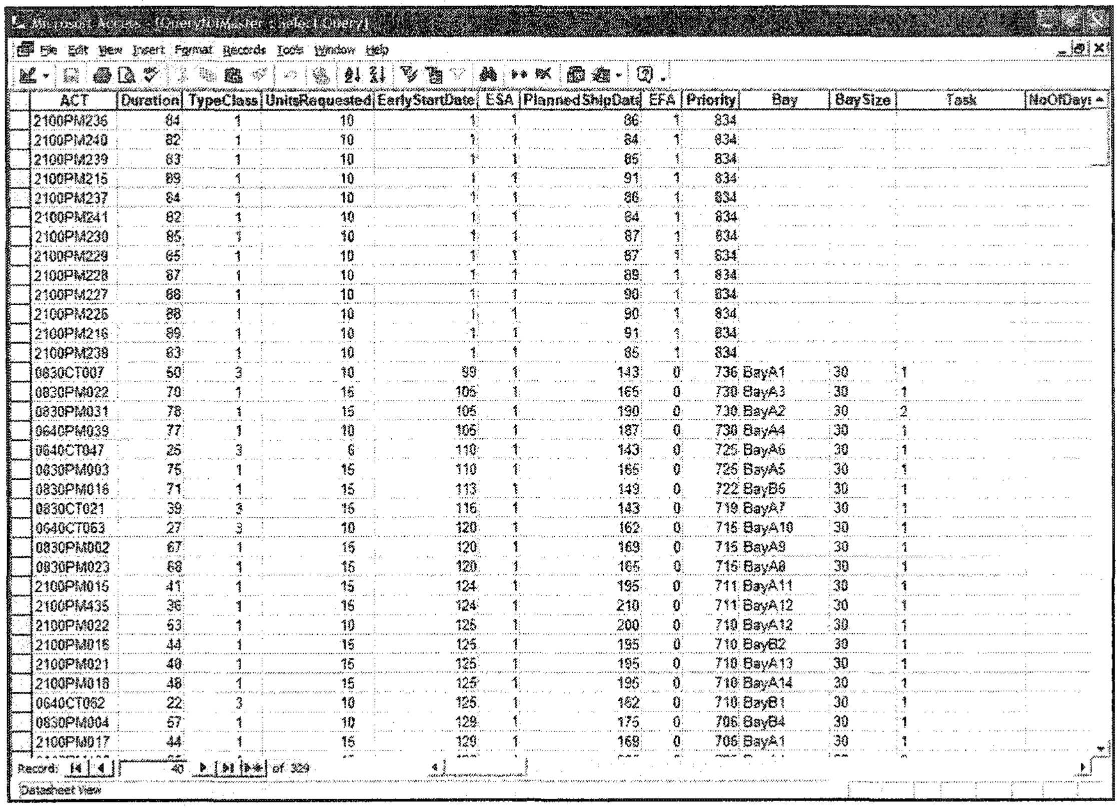
there was a need to have the simulation linked to the database file in order to simplify the procedure, two new elements were added to the common template in <u>*Simphony*</u> (AbouRizk and Hajjar 1998). These two new elements are called the "Database Link Element" and the "Results Element".

2. *Determination of the Priority Logic:* Modules are processed based on the priority. The calculation of module priority has been previously explained in Chapter 3.
3. *Representation of the bays:* Bays are represented by resources and the number of resources forming a bay is based on the size and the total number of modules that may fit in a bay.
4. *Placing modules that are already under fabrication when the simulation starts in the exactly same bay where they are being assembled in real life, and being able to join the simulated fabrication process at the subtask in which they are currently under fabrication.*
5. *Keeping track of the finishing times of the modules under fabrication in the same bay.* Before a module starts its assembly process it has to be guaranteed that it will not delay the modules that are currently under fabrication in that same bay (if any). To accomplish that, two attributes were added as resources. These two attributes keep track of the finishing time of the module in the back of the bay and the finishing time of the module with the latest finishing time on that bay. The expected finishing time (starting time + duration) of the module to be built is compared to those attributes.
6. *Keeping track of the modules that are under fabrication in the same bay.* In order to know in which part of the bay a module is being fabricated and if more space will still be available to fabricate more modules in that bay, two attributes were also added as resources where they are expressed as resources previously taken from that bay and resources still available in that bay.
7. *Ensuring that there are no modules in front of any module at the time of shipment.* When a module starts its assembly process it request resources for assembly, the number of resources left on that bay represent the space in front of it. This space is required to be empty at the time of shipment for that

specific module. Therefore, those resources are recorded as an attribute also expressed as a resource called "space". When a module is ready to be shipped its attribute "space" is compared to the number of resources available on that bay.

4.3 Database

Reading data from the database: In order to have the information available in *Simphony*, database file was necessary. *Simphony* is a Visual Basic-based program; it has the flexibility, therefore, to create new elements, which are capable of using both the Visual Basic Code and Library. Two new elements called the "Database Link Element" and the "Results Element" were created. These elements are able to read information contained in tables and queries within a database file and import/export the information to and from *Simphony* with the same order and format. Once those elements are pasted into the *Simphony* Design Window, the user can employ the parameters window to specify the path and the name of the database file as well as the tables or queries name from/to which to import/export the data. The data must be ordered from the earliest date to the latest date based on EarlyStartDate rules. A query has been built to prioritize the information used for importing the data. The query lists the modules in an ascending order based on their EarlyStartDate. Figure 7 illustrates the query (data format), which contains the attributes for the input as well as those attributes in which the output will be recorded after the simulation; the records representing one module each ordered by the EarlyStartDate (column 5) are also illustrated.



ACT	Duration	TypeClass	UnitsRequested	EarlyStartDate	ESA	PlannedShipDate	EFA	Priority	Bay	BaySize	Task	NoOfDays
2100PM236	84	1	10	1	1	86	1	834				
2100PM240	82	1	10	1	1	84	1	834				
2100PM239	83	1	10	1	1	85	1	834				
2100PM215	89	1	10	1	1	91	1	834				
2100PM237	84	1	10	1	1	86	1	834				
2100PM241	82	1	10	1	1	84	1	834				
2100PM230	85	1	10	1	1	87	1	834				
2100PM229	65	1	10	1	1	87	1	834				
2100PM228	87	1	10	1	1	89	1	834				
2100PM227	88	1	10	1	1	90	1	834				
2100PM226	88	1	10	1	1	90	1	834				
2100PM216	89	1	10	1	1	91	1	834				
2100PM238	83	1	10	1	1	85	1	834				
0830CT007	50	3	10	99	1	143	0	736	BayA1	30	1	
0830PM022	70	1	15	105	1	165	0	730	BayA3	30	1	
0830PM031	78	1	15	105	1	190	0	730	BayA2	30	2	
0640PM039	77	1	10	105	1	187	0	730	BayA4	30	1	
0640CT047	25	3	6	110	1	143	0	725	BayA6	30	1	
0830PM003	75	1	15	110	1	165	0	725	BayA5	30	1	
0830PM016	71	1	15	113	1	149	0	722	BayB5	30	1	
0830CT021	39	3	15	116	1	143	0	719	BayA7	30	1	
0640CT063	27	3	10	120	1	162	0	715	BayA10	30	1	
0830PM002	67	1	15	120	1	169	0	715	BayA9	30	1	
0830PM023	68	1	15	120	1	165	0	715	BayA8	30	1	
2100PM015	41	1	15	124	1	195	0	711	BayA11	30	1	
2100PM435	36	1	15	124	1	210	0	711	BayA12	30	1	
2100PM022	53	1	10	125	1	200	0	710	BayA12	30	1	
2100PM016	44	1	15	125	1	195	0	710	BayB2	30	1	
2100PM021	48	1	15	125	1	195	0	710	BayA13	30	1	
2100PM018	48	1	15	125	1	195	0	710	BayA14	30	1	
0640CT062	22	3	10	125	1	162	0	710	BayB1	30	1	
0830PM004	57	1	10	129	1	175	0	706	BayB4	30	1	
2100PM017	44	1	15	129	1	169	0	706	BayA1	30	1	

Record: 40 of 329

Datasheet View

Figure 7 - Query Ordered by EarlyStartDate

New Elements Creation: The "Database Link Element" requires the parameters of the Data Source and of the query/table to be specified. Also, it will display the number of records, the number of attributes, and the entire query/table as an output. The element is formed using four functions: OnCreate, OnSimulationInitialize, OnSimulationInitializeRun, and OnSimulationProcessEvent. The first function of the element is the OnCreate function in which all the necessary attributes of the element are declared and the element itself is defined. In the OnCreate function, the database name is specified as well as its path. The table or the query from which to retrieve the information is also specified. The matrix to store the information from the table or query is also declared.

The OnSimulationInitialize subroutine is where the simulation first opens the database to set the table. The number of records, the number of attributes, and the names of the attributes are read. The table is completely emptied at this point. In the OnSimulationInitialize subroutine the attributes' names are read beginning with the first attribute. This process is undertaken with each module (record). Once the subroutine is finished, a table of "n" rows and "m" columns is created; however, the attributes' values are still empty.

The OnSimulationInitializeRun subroutine launches the first module; which fills the first record of the table with the values of the attributes contained in that module. The OnSimulationInitializeRun begins the simulation process at its EarlyStartTime firing the first module.

During the OnSimulationProcessEvent subroutine; the simulation will fill the rest of the table with the attribute values of each module. Similar to the OnSimulationInitializeRun, the modules initialize the simulation process based on its recorded EarlyStartTime.

The "Results Element" also requires that the parameters of the data source and the query/table are specified. Also, it will display the number of records, the number of attributes, and the entire query or table as output. Similar to the "Database Link Element" the OnCreate function declares the necessary attributes of the element and essentially defines the element itself. The OnSimulationInitialize subroutine first opens the database in order to establish the results table. The number of records, number of attributes, and the names of the attributes are read. The table is completely emptied at this point. The two additional subroutines are OnSimulationTransferIn and OnSimulationPostRun. The OnSimulationTransferIn subroutine records the results in the "Results Element" table.

OnSimulationPostRun exports the results to table the "Results" table of the previously identified database file.

4.4 Modifications to *Simphony*'s Common Template

Three elements from the existing common template were modified to suit the special requirements of this simulation. These three elements are: "Declare Resources", "Waiting File", and "Release Resouces".

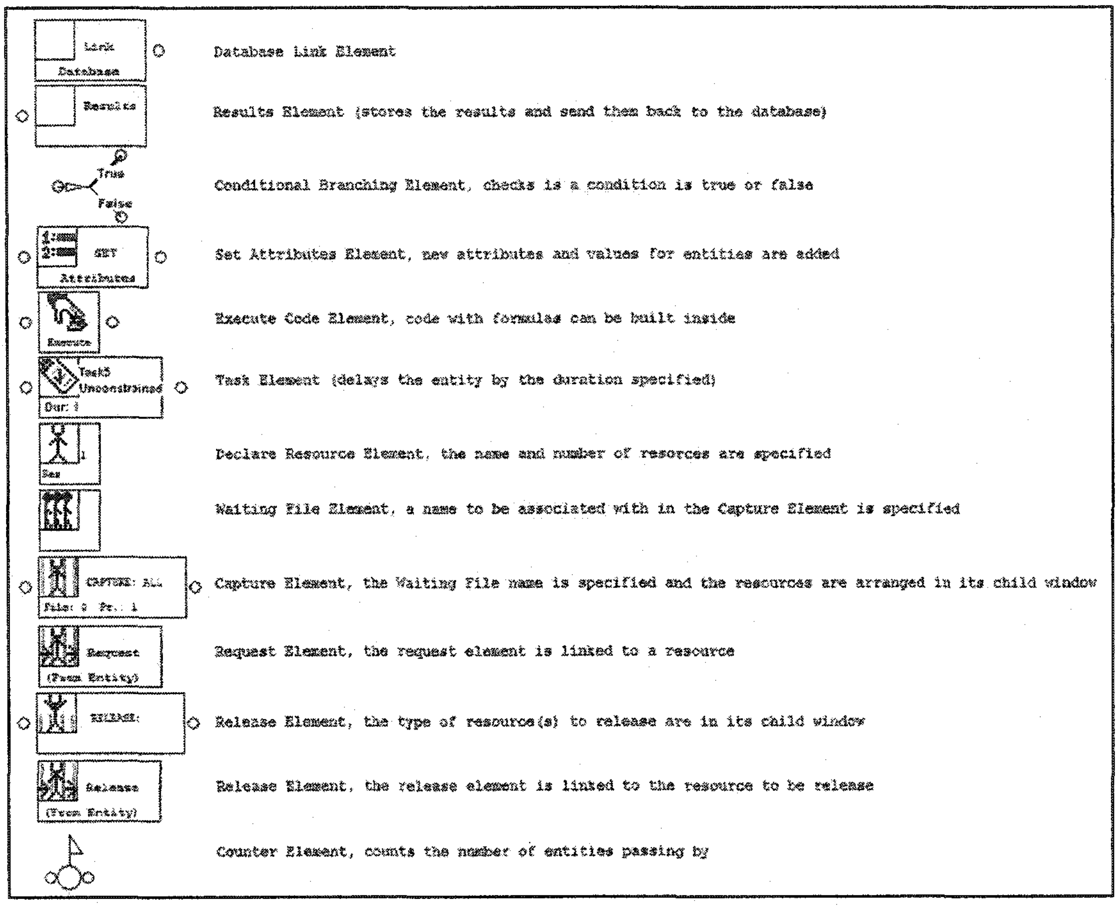
Declare Resources Element: Two subroutines from the Declare Resources element were modified. The OnCreate function of the Declare Resources element was modified to add the EndBackModule, FinishCurrentModule, ModuleCounter, and SpaceInFront attributes. The EndBackModule attribute tracks the finishing time of the module in the back of a bay. "FinishCurrentModule" keeps track of the module with the latest finishing time for each bay. "ModuleCounter" records the number of modules in one bay. "SpaceInFront" tracks the available space preceding the current module in order to fabricate more modules. The OnSimulationInitializeRun subroutine was also modified by initializing the new attributes at 1000,1000,0, and 1. "EndBackModule" and "FinishCurrentModule" attributes, must be associated with a high value (larger than all of the modules durations) to ensure that when the bay is empty it grants resources for assembly. For this reason, the attributes have been given a value of 1000, ensuring that regardless of the order in which modules arrive to a bay, that module's finishing time will be shorter than the virtual module in the back of the bay. The "ModuleCounter" attribute is initialized with a value of zero since the bays are empty at the beginning of the simulation. "SpaceInFront" takes the initial value of one (1), which represents the empty space available for modules in a bay.

Waiting File Element: The Waiting File element encompasses the process of granting resources. The module assembly schedule is restricted by those physical and logical constraints mentioned previously as well as heuristic rules. These constraints and heuristic rules have been added to the Waiting File element. This addition allows the waiting file to grant resources only when all criteria have been met. The changes have been made within the Case "ANY"; when a module arrives to the Capture element and request resources for assembly, the module will request "ANY" of the resources (bays) available. The five request types must first be differentiated. In order to differentiate between the modules requesting for the space that they are currently occupying (for modules that started the fabrication process previously to the simulation), those requesting space for assembly, those requesting labor, those requesting space available for shipment, and those requesting a resource for shipment, an auxiliary attribute ranging from zero (0) to eight (8) is checked every time a request is placed. If the auxiliary attribute is zero (0), then the module is requesting space for assembly (modules that have not started their fabrication process), if the value is two (2), then the module is requesting the space that they are currently occupying (modules that started the fabrication process previous to the simulation), if the value is three (3), then the module is requesting workers for the structure subtask, if the auxiliary attribute is four (4), then the module is requesting workers for the piping subtask, if the auxiliary attribute is five (5), then the module is requesting workers for the cable tray subtask, if the auxiliary attribute is six (6), then the module is requesting workers for the EHT subtask, if the auxiliary attribute is seven (7), then the module is requesting workers for the insulation subtask, if the auxiliary attribute is eight (8) then the module is requesting workers for the fireproof subtask; if the value is one (1), then the module is requesting for another type of resource. In this latter situation, the process is exactly the same as a normal request processed by the *Simphony* Common Template. The resource request must then be fixed, which will enable resources to be requested based on an entity attribute (formula) rather than specifying a keyed number within the Capture element. If the module in the back (EndBackModule) and the module with the latest finishing time (EndCurrentModule) on a bay plus the maximum allowable number of waiting days for shipment (WaitingDays) finish later than the module requesting resources (the finishing time of the module requesting resources is calculated by adding all the subtasks' durations to the current simulation time, which represents the start time, then the module is granted the resources and the assembly, space, total, and bay attributes are updated. The EndBackmodule attribute changes its value from 1000 to the expected finish time if the module requesting resources is placed at the back of the bay. The EndCurrentModule attribute also updates its value to the expected finish time of that module requesting resources. ModuleCounter and SpaceInFront are the remaining attributes. ModuleCounter is increased by one unit every time a module is granted resources while SpaceInFront checks that the space available is greater than zero (0).

Release Resources Element: The OnSimulationProcessEvent of the "Release Resources" element was modified to update the EndBackModule, FinishCurrentModule, and ModuleCounter attribute every time a module has finished its assembly process and the space resources are released. The auxiliary attribute used on the Waiting File to differentiate the requesting of space for assembly is given a value of two (2) in order to accomplish this process at the end of the assembly. The attributes revert to their original value representing an empty bay (EndBackModule = 1000, FinishCurrentModule = 1000, and ModuleCounter = 0).

4.5 Simulation Model

The simulation model has been built mainly using *Simphony*'s Common Template with two new elements mentioned before and with the modifications made to three of the existing elements. The counter-element from *Simphony*'s CYCLONE II was also used in this model. See Figure 9 for a legend of the elements used in this model.



Database Link Element

Results Element (stores the results and send them back to the database)

Conditional Branching Element, checks is a condition is true or false

Set Attributes Element, new attributes and values for entities are added

Execute Code Element, code with formulas can be built inside

Task Element (delays the entity by the duration specified)

Declare Resource Element, the name and number of resorces are specified

Waiting File Element, a name to be associated with in the Capture Element is specified

Capture Element, the Waiting File name is specified and the resources are arranged in its child window

Request Element, the request element is linked to a resource

Release Element, the type of resource(s) to release are in its child window

Release Element, the release element is linked to the resource to be release

Counter Element, counts the number of entities passing by

Figure 8 - Legend (*Simphony*'s elements used for this model)

The model was built following that simulation model development process explained in Chapter 3 and summarized in Figure 6. Figure 9 shows the main window containing *Simphony*'s simulation model built for module assembly scheduling. The twenty-five points following Figure 9 are the explanation of the simulation model. Also, Figure 10 shows the flow chart explaining the model.

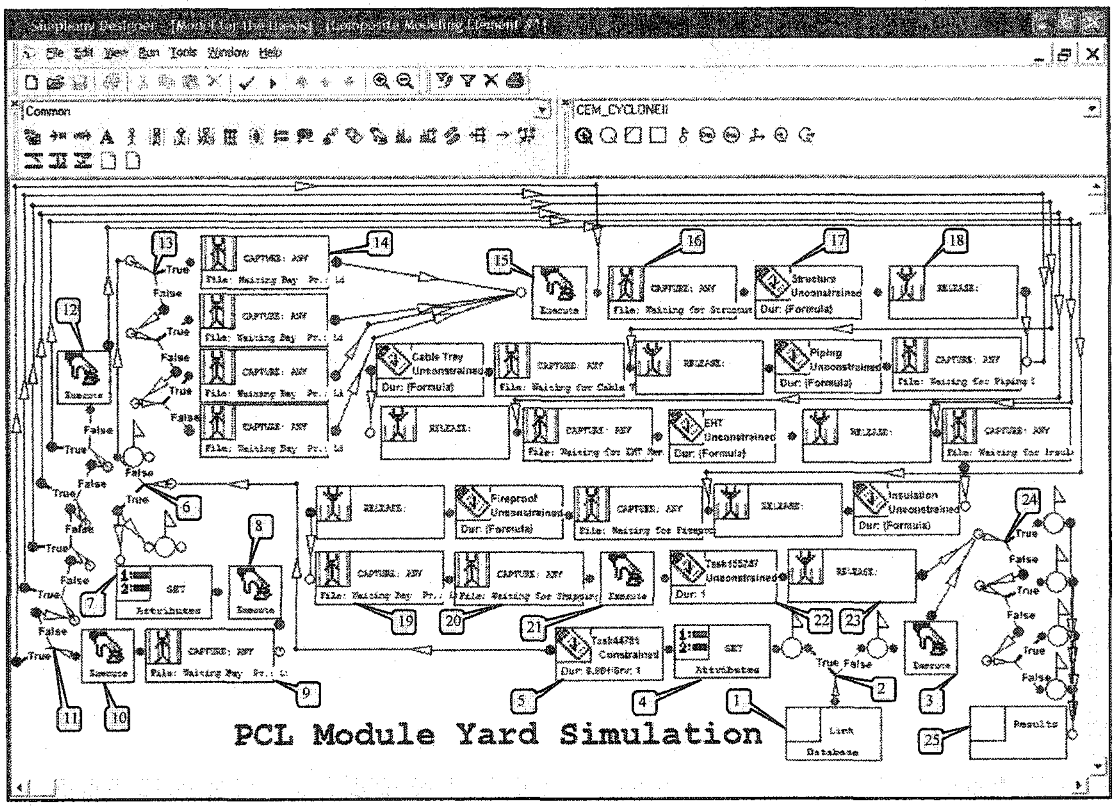


Figure 9 - *Simphony*'s Simulation Model

[1] Simulation start (reads data from database)

[2] Differentiating finished modules and counting the number of finished and unfinished modules.

"EFA" (Attribute to differentiate if the module has finished its assembly process or not)

If "EFA" = 0, the module has not completed its assembly process yet.

If "EFA" = 1, the module has already completed its assembly process.

[3] "EFA" =1. The starting time, finishing time, and shipping time of the finished modules are recorded.

"Start"= "EarlyStartDate"

"Finish" = "EarlyStartDate" + "Duration"

"Shipping" = "PlannedShipDate"

[4] The Auxiliary attribute to differentiate types of request and release is set to zero (0).

"Auxiliary" = 0

[5] Task is undertaken with a delay of 0.0001 days to ensure that no modules request resources at the exactly same time. Modules having the same "EarlyStartDate" will request resources at the same time; this small delay will form a queue when granting resources. Since the dates are based on integer numbers, the delay of fractions of a second will not alter the dates.

[6] Differentiating started modules from modules not yet started and counting them.

"ESA" (Attribute to differentiate if the module has started its assembly process or not)

If "ESA" = 0, the module has not started its assembly process yet.

If "ESA" = 1, the module has already started its assembly process.

[7] Based on the information provided regarding the modules already in progress, the attributes assembly, space, and total are recorded.

Assembly = "UnitsRequested"

"Space" = "BaySize" – "Assembly" – "No_of_Units_Occupied_Behind"

"Total" = "Assembly" + "Space"

[8] The auxiliary attribute takes the value of two (2), symbolizing a module that started previously to the simulation. It is also determined that the bay in which that module is been assembled still has available space for other assembly.

"Auxiliary" = 2

If ("BaySize" – "UnitsRequested" – "No_of_Units_Occupied_Behind") = 0

"Space_in_front" = No

Else

"Space_in_front" =Yes

[9] The module requests the space currently occupying in the bay in which it is been assembled in real life.

Bat to request resources from = "Bay"

Number of resources to request from "Bay" = "UnitsRequested"

[10] The starting time of the module is recorded. Also, the auxiliary attribute takes the value of (1) symbolizing that the module has been assigned to a bay.

"Auxiliary" = 1

"Start" = SimTime

[11] Modules that started their assembly process previously to the simulation are currently under assembly in a specific subtask. These series of conditional branching differentiates the subtask in which the module is at the simulation starting time. Depending on the subtasks the auxiliary attribute could take the value of (3) for structure, (4) for piping, (5), for cable tray, (6) for EHT, and (7) for insulation.

If "Task" = 1 (structure)

"Auxiliary" = 3

If "Task" = 2 (piping)

"Auxiliary" = 4

If "Task" = 3 (cable tray)

"Auxiliary" = 5

If "Task" = 4 (EHT)

"Auxiliary" = 6

If "Task" = 5 (insulation)

"Auxiliary" = 7

[12] When the last of those conditional branching is false, the auxiliary attribute takes the value of (8) for fireproof.

"Auxiliary" = 8

[13] Routing modules to different bay areas

If "TypeClass" = 1 the module is preferably routed to bay area "A"

If "TypeClass" = 2 the module is preferably routed to bay area "B"

If "TypeClass" = 3 the module is preferably routed to bay area "C"

If "TypeClass" = 4 the module is preferably routed to bay area "D"

If "TypeClass" = 5 the module is preferably routed to bay area "D"

[14] Looking for empty space on a bay to start fabrication

Number of resources to request from any of the available bays = "UnitsRequested"

[15] Once the module has been assigned to an empty space, the Assembly, Space, Total, and Bay attribute are updated. The auxiliary attribute is associated with the value of one (3), symbolizing that the module is ready to requested workers for the first subtask.

"Assembly" = "UnitsRequested"

"Space" = "Current" (Current number of available resources in the bay)

"Total" = "Assembly" + "Current"

"Bay" = Name of the bay from where the resources have been assigned

"Auxiliary" = 3

[16] Request labor in order to begin the module fabrication process

Number of workers to request = "Manhours_Struct"

[17] The first subtask (structure) is processed and the starting time of the subtask is recorded.

Task duration = "Duration_Override_Struct"

"Actual_Start_Struct" = SimTime

[18] Release of the labor used in the first subtask (structure), the finishing time of the subtask is recorded, and the auxiliary attribute takes the value of four (4), symbolizing that the module is ready to requested workers for the second subtask.

Number of workers to release = "Manhours_Struct"

"Actual_Finish_Struct" = SimTime

"Auxiliary" = 4

[19] Request for available shipment space (empty space in front of the completed module)

Number of resources to request symbolizing the empty space needed for shipping = "Space"

[20] Request shipment

Number of resources to request = 1 (one shipment is requested)

[21] The shipping time is recorded and the auxiliary attribute takes the value of two (2), symbolizing that the module is ready to be shipped.

"Shipping" = SimTime

"Auxiliary" = 2

[22] Shipment occurs during one whole day; therefore, resources should be released after the completion of the shipment process

Task duration = 1

[23] Release resources (space and shipment)

Number of space resources to release = "Total"

Number of shipment resources to release = 1

[24] Differentiating modules according to their type and counting them.

If "TypeClass" = 1 pass counter No. 1

If "TypeClass" = 2 pass counter No. 2

If "TypeClass" = 3 pass counter No. 3

If "TypeClass" = 4 pass counter No. 4

If "TypeClass" = 5 pass counter No. 4

[25] Simulation ends (send results back to the database)

Note: Between steps [18] and [19], the five (5) other subtasks are also performed. They request labor, process the task, and release the labor used in a similar manner to steps [16], [17], and [18].

"Manhours_Struct" changes for "Manhours_Piping", "Manhours_Cable_Tray", "Manhours_EHT", "Manhours_Insulation", and "Manhours_Fireproff" depending on the task. In a similar way "Duration_Override_Struct" and "Actual_Start_Struct" change depending on the task. Also, the "Auxiliary" attribute takes the value of five (5) to symbolize that the module is ready to requested workers for the third subtask, the value of six (6) to symbolize that the module is ready to requested workers for the fourth subtask, the value of seven (7) to symbolize that the module is ready to requested workers for the fifth subtask, and the value of eight (8) to symbolize that the module is ready to requested workers for the sixth subtask.

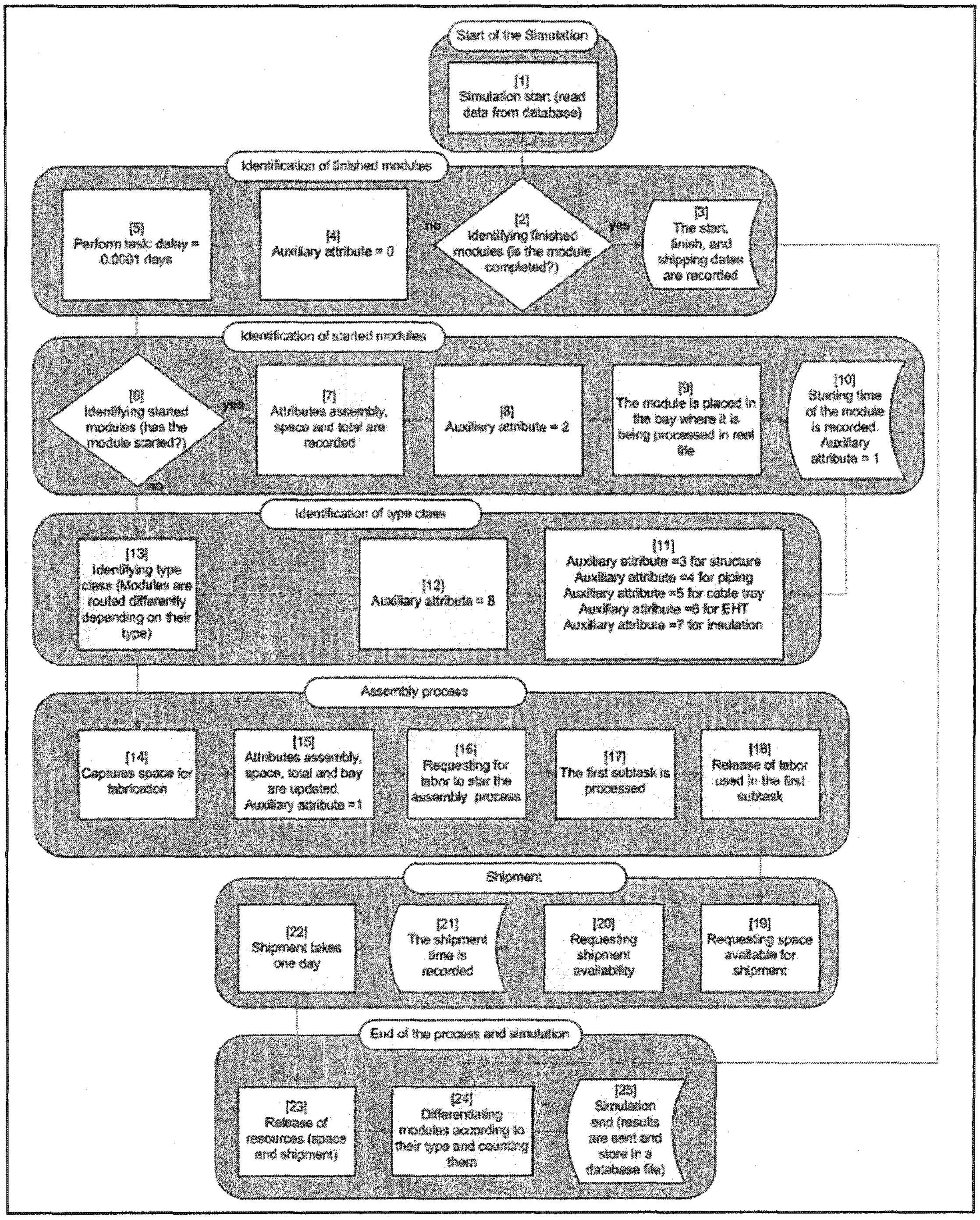

Start of the Simulation
[1] Simulation start (read data from database)
Identification of finished modules
[5] Perform task: delay = 0.0001 days
[4] Auxiliary attribute = 0
no
[2] Identifying finished modules (is the module completed?)
yes
[3] The start, finish, and shipping dates are recorded
Identification of started modules
[6] Identifying started modules (has the module started?)
yes
no
[7] Attributes assembly, space and total are recorded
[8] Auxiliary attribute = 2
[9] The module is placed in the bay where it is being processed in real life
[10] Starting time of the module is recorded. Auxiliary attribute = 1
Identification of type class
[13] Identifying type class (Modules are routed differently depending on their type)
[12] Auxiliary attribute = 8
[11] Auxiliary attribute =3 for structure
Auxiliary attribute =4 for piping
Auxiliary attribute =5 for cable tray
Auxiliary attribute =6 for EHT
Auxiliary attribute =7 for insulation
Assembly process
[14] Captures space for fabrication
[15] Attributes assembly, space, total and bay are updated. Auxiliary attribute =1
[16] Requesting for labor to star the assembly process
[17] The first subtask is processed
[18] Release of labor used in the first subtask
Shipment
[22] Shipment takes one day
[21] The shipment time is recorded
[20] Requesting shipment availability
[19] Requesting space available for shipment
End of the process and simulation
[23] Release of resources (space and shipment)
[24] Differentiating modules according to their type and counting them
[25] Simulation end (results are sent and store in a database file)

Figure 10 - Simulation Model Flow Chart

The main window of the simulation also contains the resource declaration. The bays in every area, the availability of shipments per day, and the skill workers available for each subtask are declared. The parameters of each of these resources can be edited, after which the total number of resources is set. The waiting files are also declared at this point. There is only one waiting file for all the bays instead of one per bay area because bays arrive and request for space from the preferred area previously chosen except when that area is full. In that case, the module will be assigned to a bay in a different area. Shipment and subtasks each have a waiting file since those tasks are independent from the rest of the resource requests. Figure 11 shows the resource declaration and the waiting files.

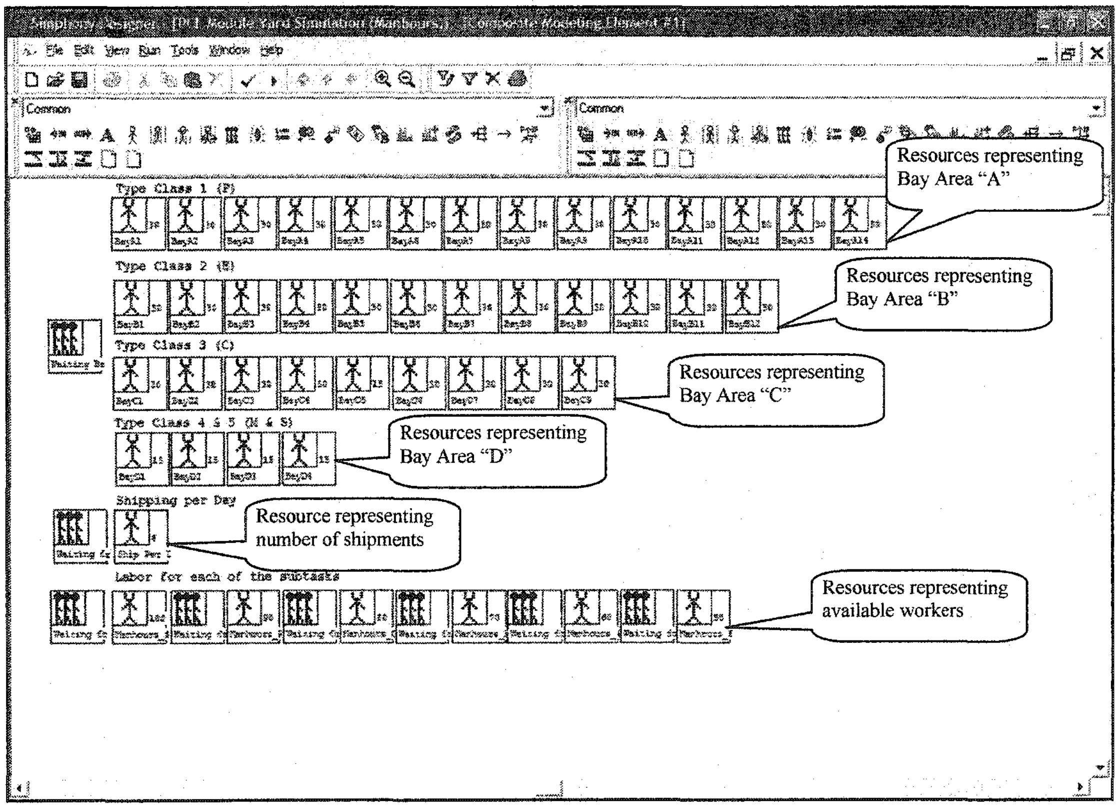

Type Class 1 (F)
Type Class 2 (E)
Type Class 3 (C)
Type Class 4 & 5 (H & S)
Shipping per Day
Labor for each of the subtasks
Resources representing Bay Area "A"
Resources representing Bay Area "B"
Resources representing Bay Area "C"
Resources representing Bay Area "D"
Resource representing number of shipments
Resources representing available workers

Figure 11 – *Simphony*'s Resources and Waiting Files Window

Child windows are built beneath the Capture and the Release elements. Figure 12 shows the Capture element window in which the resources are arranged in an imitation of the module yard layout. These resources must be created based on the planned sequence for filling bays. The Capture element's child window modules are preferably routed to bay area "A". In this case, the resources were created starting with bay area "A" (from A1 to A14), then bay area "B" (from B1 to B12), then bay area "C" (from C1 to C9), and finally bay area "D" (from D1 to D4).

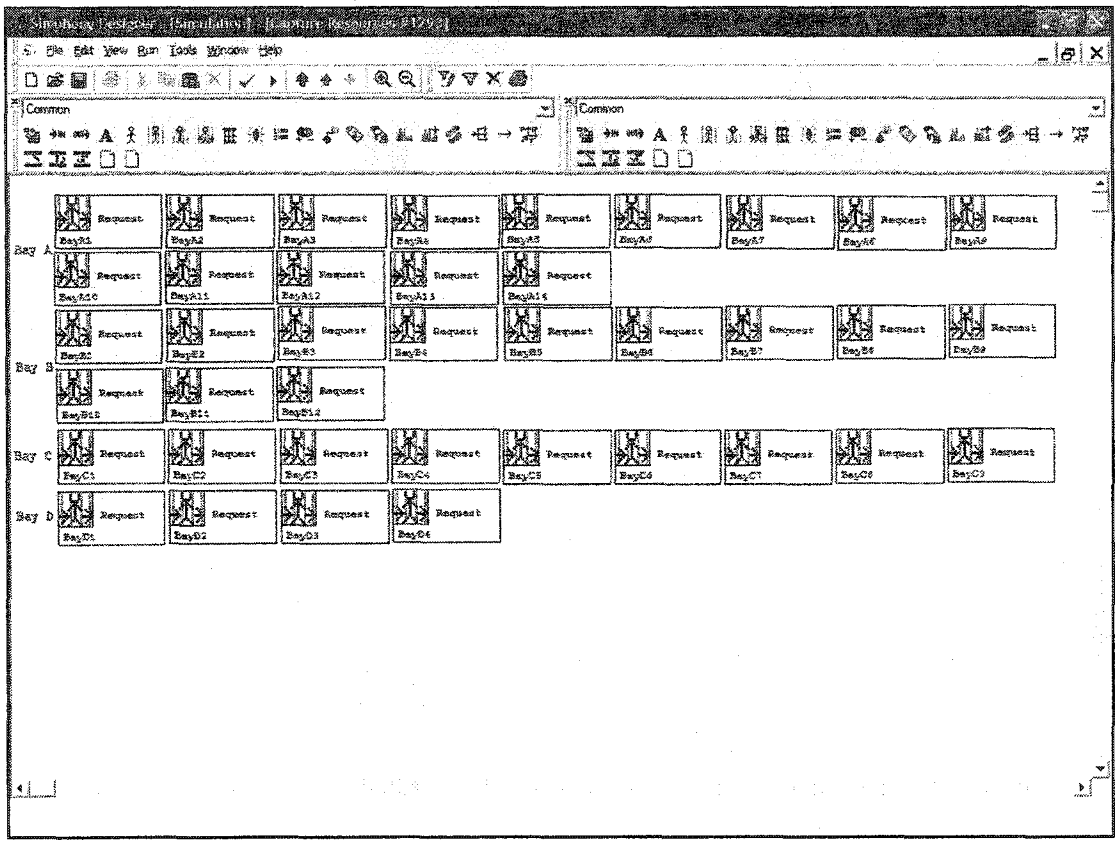


Figure 12 - Child Window of the Capture Element when Modules are Routed to Bay Area "A"

Figure 13 shows the Release element window in which those resources scheduled for release are built. One resource for every resource type to be released must be built. Two releases take place simultaneously when a module is shipped: 1) the ship per day resource is released and ready to ship more modules and 2) the space occupied by that module in a bay is now released and ready to host other modules. The number of bay space resources to be released is specified in each entity by its "Total" and "Bay" attribute.

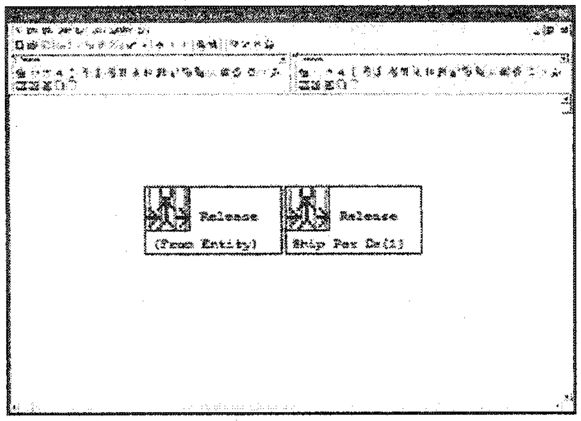


Figure 13 - Child Window of the Release Element (bay space and ship per day)

Chapter 5. Case Study Analysis

5.1 Introduction

The module assembly yard schedule described in this thesis will be implemented in PCL's module assembly yard in Nisku, Alberta (see Figure 14 for an aerial view of the yard). PCL's module assembly project consists of the assembly of 268 piperack modules, 40 equipment modules, and 21 cable tray modules for the one of the projects in the oil sands located in Fort McMurray, Alberta. Pipe spools are produced in PCL's Nisku pipe fabrication facility and transported to the module assembly yard for inclusion in a wide variety of structural modules. The scope of the work comprises of the erection of structural steel and the installation of pipe spools, insulation, cable tray installation, heat tracing, equipment, and fireproofing. Once installed in the steel module frames, the spools are seamed, heat traced, and insulated in order to complete the assemblies. Electrical cable tray installation, and fireproofing are also completed prior loading and delivering the module for installation on-site.

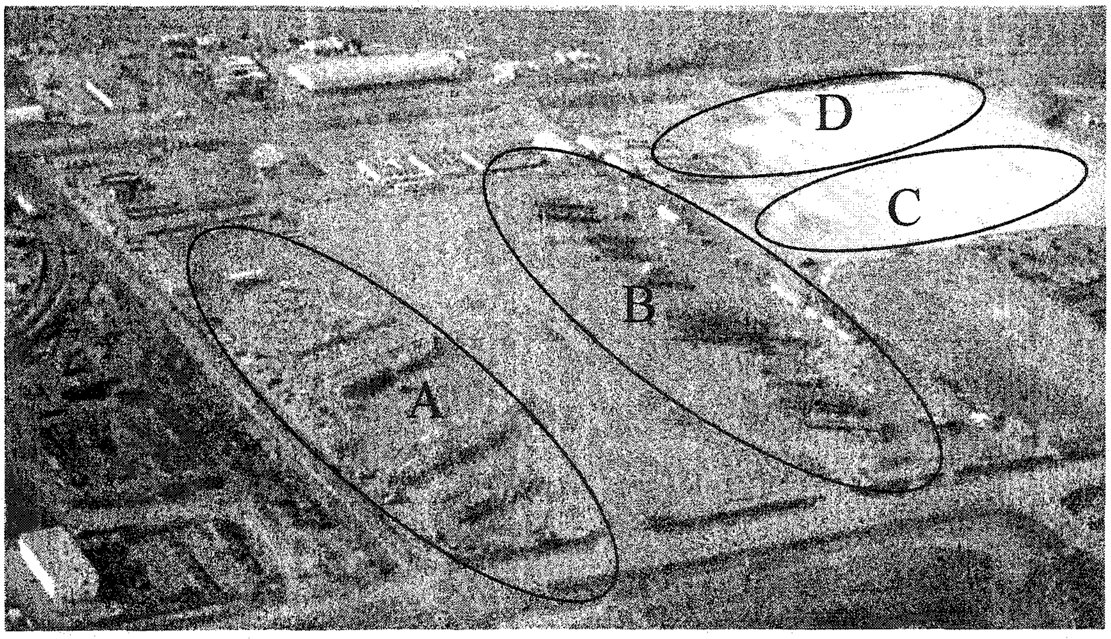


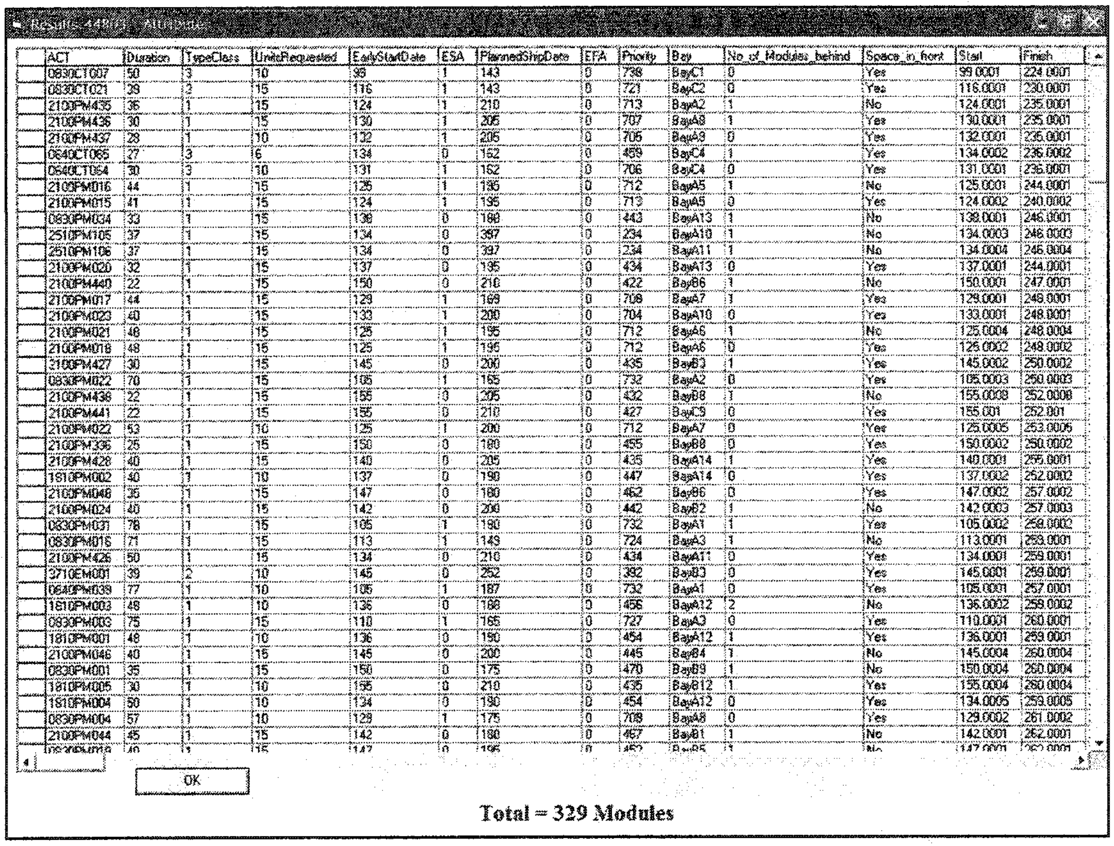
Figure 14 - PCL's Module Assembly Yard in Nisku, Alberta

The specifics of the case were analyzed and treated as invariable constraints to the project schedule. Those constraints need to be determined by a group of experts including the scheduler and the project manager. The simulation enables the testing of different scenarios in which the constraints assumed at the beginning are modified and a proper evaluation of each scenario is undertaken. This exercise will ensure a better combination of constraints. This section presents the results of nine scenarios. In each scenario, one constraint was changed, while all other constraints were left fixed. The following assumptions apply to the first scenario:

- Module yard layout is fixed (see Figure 5, Chapter 3),
- Resources (man-hours) not taken into account,
- Modules may only be shipped when the space in front of them is completely empty,
- Maximum number of shipments per day is six,
- After completion, modules may wait a maximum of five days for shipment,
- Module allocation routing follows certain preferences; in this case they are routed according to their type,
- Once a module has been routed to a specific area, the work will flow front to back (starting with bay #1 to bay #n),
- Duration and dates are fixed (duration varied from 21 days to 92 days), and
- Priority logic is employed; that is the module with the least amount of float will be given higher priority for assembly

5.2 Validation of Basic Model

Results are organized in a tabular format providing Activity ID (ACT), durations, class type, units requested (size), early start date, the module's start status at the beginning of the simulation (ESA), the planned shipping date, the module's completion status (EFA), priority, location of module within the yard (bay in which the module was processed), location of the module within the bay (number of modules behind in the bay), space in front (in the bay), simulated start dates (process and subtasks), simulated finish date (process and subtasks), and simulated ship date. Each simulation run provides results for 329 modules (see Figure 15).

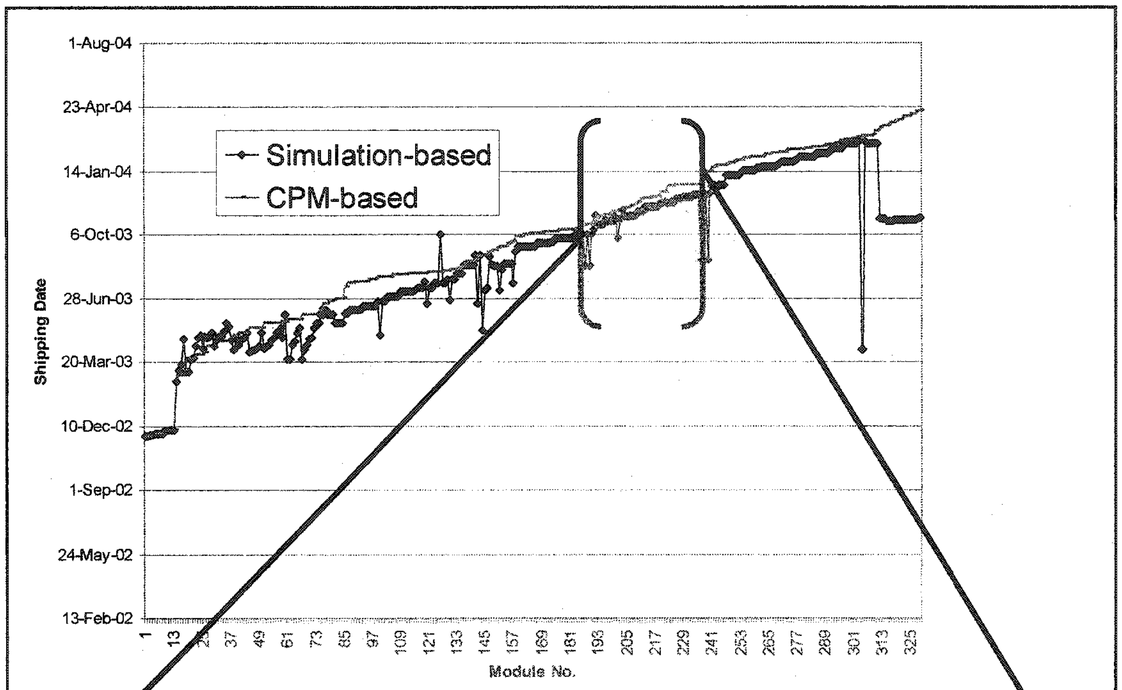


ACT	Duration	TypeClass	UnitsRequested	EarlyStartDate	ESA	PlannedShipDate	EFA	Priority	Bay	No_of_Modules_behind	Space_in_front	Start	Finish
0830CT007	50	3	10	99	1	143	0	738	BayC1	0	Yes	99.0001	224.0001
0830CT021	29	2	15	116	1	143	0	721	BayC2	0	Yes	116.0001	230.0001
2100PM435	36	1	15	124	1	210	0	713	BayA2	1	No	124.0001	235.0001
2100PM436	30	1	15	130	1	205	0	707	BayA8	1	Yes	130.0001	235.0001
2100PM437	28	1	10	132	1	205	0	705	BayA9	0	Yes	132.0001	235.0001
0840CT065	27	3	6	134	0	162	0	459	BayC4	1	Yes	134.0002	236.0002
0840CT064	30	3	10	131	1	162	0	706	BayC4	0	Yes	131.0001	236.0001
2100PM016	44	1	15	125	1	195	0	712	BayA5	1	No	125.0001	244.0001
2100PM015	41	1	15	124	1	195	0	713	BayA5	0	Yes	124.0002	240.0002
0830PM034	33	1	15	138	0	188	0	443	BayA13	1	No	138.0001	246.0001
2510PM105	37	1	15	134	0	397	0	234	BayA10	1	No	134.0003	246.0003
2510PM106	37	1	15	134	0	397	0	234	BayA11	1	No	134.0004	246.0004
2100PM020	32	1	15	137	0	195	0	434	BayA13	0	Yes	137.0001	244.0001
2100PM440	22	1	15	150	0	218	0	422	BayB6	1	No	150.0001	247.0001
2100PM017	44	1	15	129	1	169	0	708	BayA7	1	Yes	129.0001	248.0001
2100PM023	40	1	15	133	1	200	0	704	BayA10	0	Yes	133.0001	248.0001
2100PM021	48	1	15	125	1	195	0	712	BayA6	1	No	125.0004	248.0004
2100PM018	48	1	15	125	1	195	0	712	BayA6	0	Yes	125.0002	248.0002
2100PM427	30	1	15	145	0	200	0	435	BayB3	1	Yes	145.0002	250.0002
0830PM022	70	1	15	105	1	165	0	732	BayA2	0	Yes	105.0003	250.0003
2100PM438	22	1	15	155	0	205	0	432	BayB8	1	No	155.0008	252.0008
2100PM441	22	1	15	155	0	210	0	427	BayC9	0	Yes	155.001	252.001
2100PM022	53	1	10	125	1	200	0	712	BayA7	0	Yes	125.0005	253.0005
2100PM336	25	1	15	150	0	180	0	455	BayB8	0	Yes	150.0002	250.0002
2100PM428	40	1	15	140	0	205	0	435	BayA14	1	Yes	140.0001	255.0001
1810PM002	40	1	10	137	0	190	0	447	BayA14	0	Yes	137.0002	252.0002
2100PM048	35	1	15	147	0	180	0	462	BayB6	0	Yes	147.0002	257.0002
2100PM024	40	1	15	142	0	200	0	442	BayB2	1	No	142.0003	257.0003
0830PM031	78	1	15	105	1	190	0	732	BayA1	1	Yes	105.0002	258.0002
0830PM016	71	1	15	113	1	149	0	724	BayA3	1	No	113.0001	253.0001
2100PM426	50	1	15	134	0	210	0	434	BayA11	0	Yes	134.0001	259.0001
3710EM001	39	2	10	145	0	252	0	392	BayB3	0	Yes	145.0001	259.0001
0640PM039	77	1	10	105	1	187	0	732	BayA1	0	Yes	105.0001	257.0001
1810PM003	48	1	10	136	0	188	0	456	BayA12	2	No	136.0002	259.0002
0830PM003	75	1	15	110	1	165	0	727	BayA3	0	Yes	110.0001	260.0001
1810PM001	48	1	10	136	0	190	0	454	BayA12	1	Yes	136.0001	259.0001
2100PM046	40	1	15	145	0	200	0	445	BayB4	1	No	145.0004	260.0004
0830PM001	35	1	15	150	0	175	0	470	BayB9	1	No	150.0004	260.0004
1810PM005	30	1	10	155	0	210	0	435	BayB12	1	Yes	155.0004	260.0004
1810PM004	50	1	10	134	0	190	0	454	BayA12	0	Yes	134.0005	259.0005
0830PM004	57	1	10	129	1	175	0	708	BayA8	0	Yes	129.0002	261.0002
2100PM044	45	1	15	142	0	180	0	467	BayB1	1	No	142.0001	262.0001

Total = 329 Modules

Figure 15 - Results in a Tabular Format

This information is plotted in comparison with the project schedule (see Figure 16), which is based on module shipping dates. Figure 16 also lists the shipping date obtained by applying the simulation and the shipping date previously planned by the contractor using the CPM within the highlighted period.



Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Simulation Based	CPM Based	Simulation Based	CPM Based	Simulation Based	CPM Based	Simulation Based	CPM Based
17-Sep-03	6-Oct-03	1-Oct-03	15-Oct-03	1-Oct-03	13-Nov-03	27-Nov-03	12-Dec-03
17-Sep-03	7-Oct-03	1-Oct-03	15-Oct-03	30-Oct-03	14-Nov-03	26-Nov-03	22-Dec-03
17-Sep-03	7-Oct-03	7-Oct-03	17-Oct-03	13-Nov-03	14-Nov-03	26-Nov-03	22-Dec-03
17-Sep-03	8-Oct-03	7-Oct-03	17-Oct-03	4-Nov-03	17-Nov-03	28-Nov-03	22-Dec-03
18-Sep-03	8-Oct-03	7-Oct-03	20-Oct-03	4-Nov-03	18-Nov-03	26-Nov-03	23-Dec-03
18-Sep-03	9-Oct-03	7-Oct-03	20-Oct-03	4-Nov-03	20-Nov-03	3-Dec-03	23-Dec-03
24-Sep-03	9-Oct-03	19-Aug-03	21-Oct-03	4-Nov-03	21-Nov-03	3-Dec-03	23-Dec-03
24-Sep-03	9-Oct-03	7-Oct-03	21-Oct-03	4-Nov-03	21-Nov-03	3-Dec-03	23-Dec-03
24-Sep-03	10-Oct-03	19-Aug-03	22-Oct-03	7-Nov-03	25-Nov-03	4-Dec-03	23-Dec-03
24-Sep-03	10-Oct-03	7-Oct-03	22-Oct-03	12-Nov-03	28-Nov-03	4-Dec-03	23-Dec-03
24-Sep-03	10-Oct-03	15-Oct-03	30-Oct-03	12-Nov-03	1-Dec-03	5-Dec-03	23-Dec-03
24-Sep-03	10-Oct-03	6-Nov-03	31-Oct-03	12-Nov-03	2-Dec-03	5-Dec-03	23-Dec-03
25-Sep-03	10-Oct-03	22-Oct-03	4-Nov-03	19-Nov-03	2-Dec-03	9-Dec-03	23-Dec-03
26-Sep-03	10-Oct-03	22-Oct-03	5-Nov-03	19-Nov-03	3-Dec-03	9-Dec-03	23-Dec-03
30-Sep-03	10-Oct-03	22-Oct-03	6-Nov-03	19-Nov-03	5-Dec-03	9-Dec-03	23-Dec-03
30-Sep-03	14-Oct-03	30-Oct-03	6-Nov-03	19-Nov-03	5-Dec-03	28-Aug-03	7-Jan-04
30-Sep-03	14-Oct-03	28-Oct-03	7-Nov-03	19-Nov-03	5-Dec-03	27-Aug-03	9-Jan-04
30-Sep-03	14-Oct-03	28-Oct-03	10-Nov-03	19-Nov-03	5-Dec-03	9-Dec-03	13-Jan-04
30-Sep-03	14-Oct-03	28-Oct-03	12-Nov-03	26-Nov-03	12-Dec-03	28-Aug-03	14-Jan-04
30-Sep-03	14-Oct-03	5-Nov-03	12-Nov-03	26-Nov-03	12-Dec-03	10-Dec-03	14-Jan-04

Figure 16 - CPM Schedule vs. Simulation Schedule (shipping dates)

Since the plot is ranked using shipping dates, the simulation schedule (simulation based) seems to indicate overall that the shipping dates were accomplished sooner than planned (CPM based).

This graph shows improvement in the schedule. This improvement is especially evident during in the latter part of the project where module allocation was more flexible due to the completion of modules filling the bays at the beginning of the project.

The following Primavera Project Planning Gant chart (see Figure 17) illustrates in the first bar the expected schedule without using the simulation model, and in the second bar, the expected schedule using the simulation model.

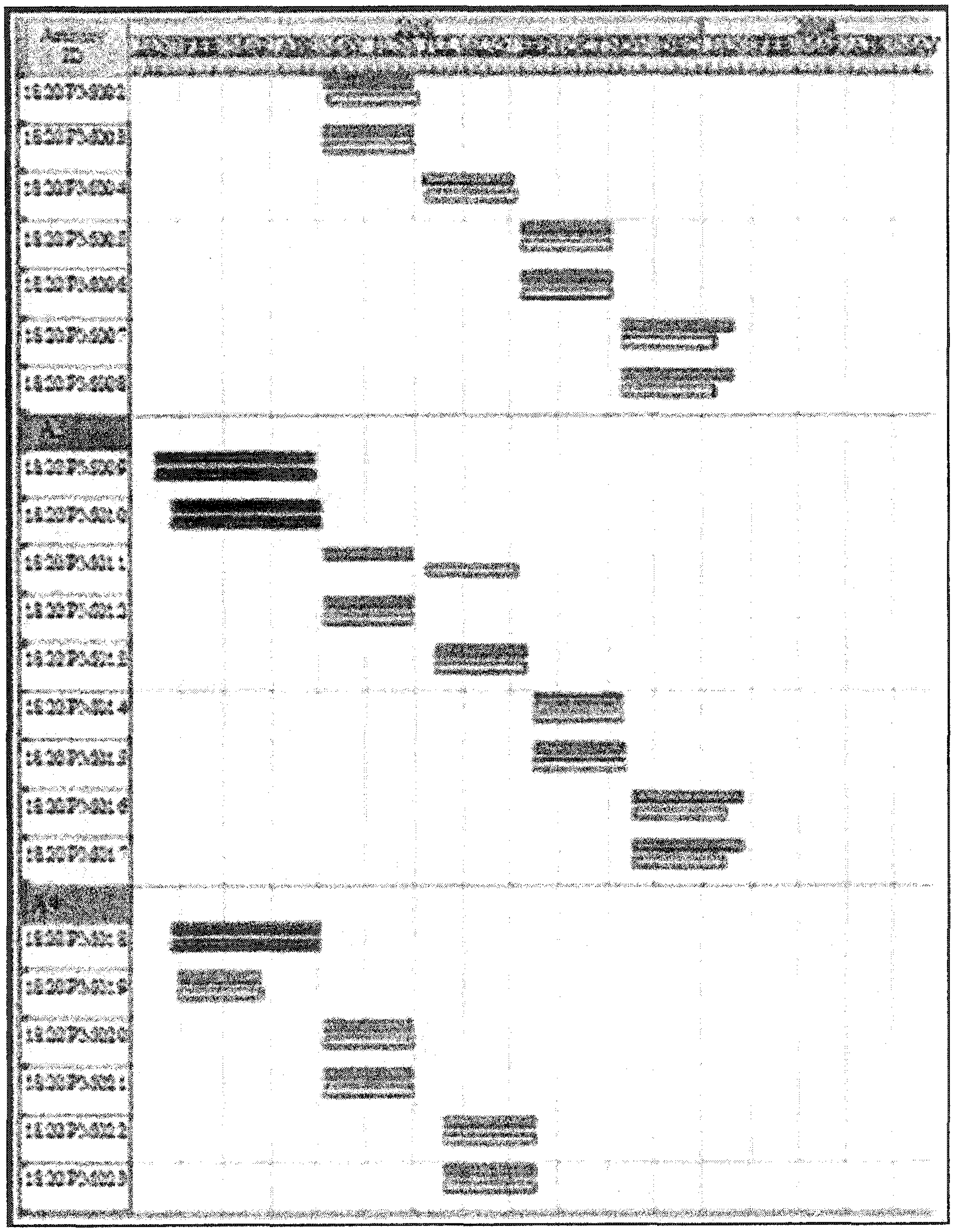
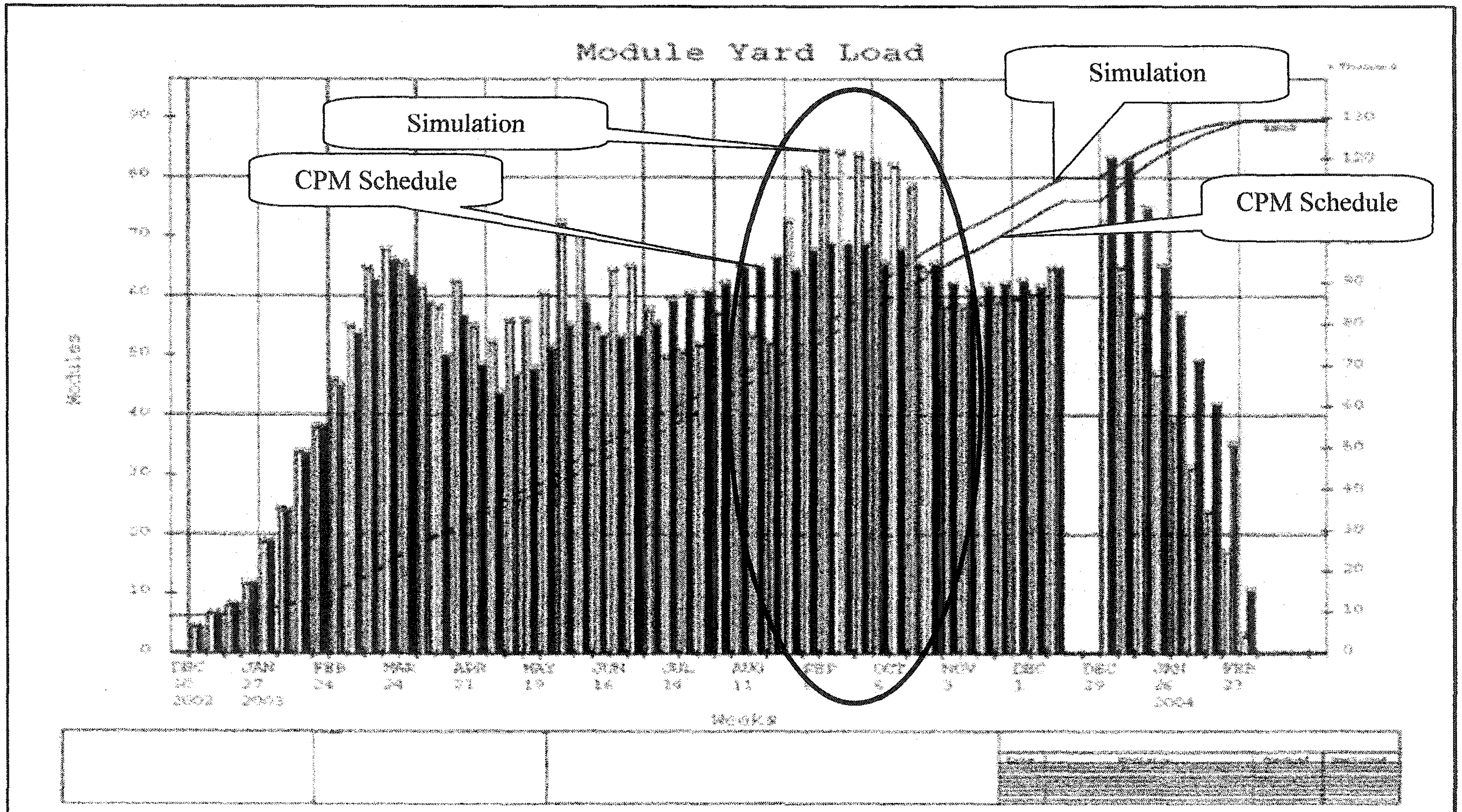


Figure 17 - Primavera Project Schedule vs. Simulation Schedule

The following Primavera Project Planning work load Primavera graph (see Figure 18), which integrates the expected amount of work per week, shows how the simulation schedule (light grey bar) proposes an increase in fabrication at the middle of the project, which leads to reduce production at the end of the project, ensuring an earlier project completion date. The schedule proposed without using simulation is represented by the dark grey bar. The selected period (September 2003 to November 2003) shows that the schedule proposed previous to the simulation does not utilize the yard at its maximum capacity. The yard was under utilized for this period having less than 70 modules in production during a week. When the simulation approximates to an optimal schedule the yard is utilized to a greater capacity having more than 80 modules in production during a week.



Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 18 - Workload

Finally, an auto-generated layout is produced (see Figure 19). This layout is a useful graph for the scheduler, since it shows the location of each module at a specific time. The scheduler can plan in advance by checking the distribution of modules in the yard on any date required:

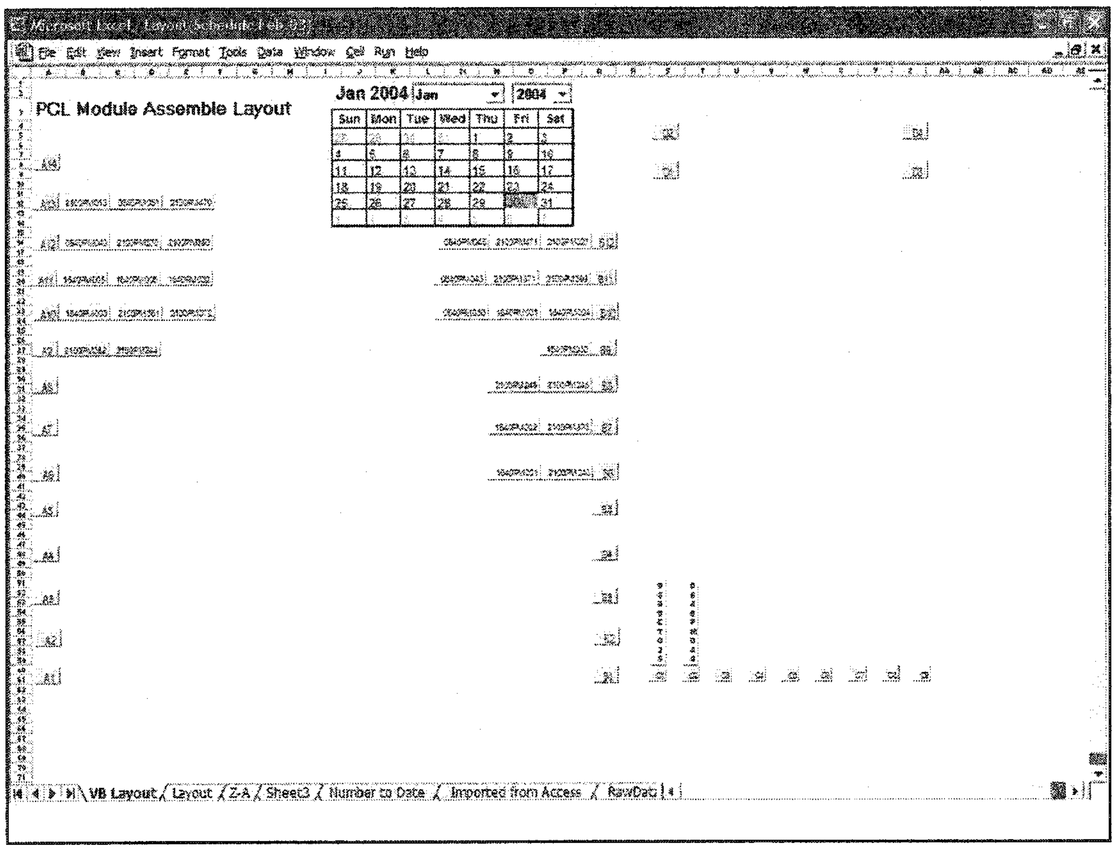


Figure 19 - Auto-generated Layout

5.3 Experimentation with the Model (improving scheduling heuristic rules)

As already discussed, the constraints assumed to perform this simulation may be changed by presenting differing scenarios that may or may not further improve the actual module assembly schedule. Nine different scenarios were obtained by testing the change made in the following constraints:

- *Changing module yard layout (3 scenarios):* The purpose of these three scenarios is to provide the manager with the flexibility to apply risk analysis and to be prepared for schedule crashing.

- *Modules are shipped regardless that the space in front of the current module is not empty:* The purpose of this scenario is to ensure that, should the schedule be improved the manager will analyze the additional cost involved in removing a module.
- *Varying the number of shipments per day (two scenarios):* The purpose of these scenarios is to determine the less possible number of shipments per day to minimize the equipment needed for shipments.
- *Varying the number or waiting days for finished modules to be shipped:* The purpose of this scenario is to determine the maximum number of waiting days for a module before shipment.
- *Routing modules according to a different characteristic such as size instead of type*: The purpose of this scenario is to provide the manager with the flexibility to compare if varying modules based on size instead of type influence the schedule.
- *Test different durations with randomness instead of just fixed durations:* This scenario provides the manager with a non-deterministic schedule that allows for the application of risk analysis to a module's completion time.

For this module assembly yard scheduling-layout problem, schedule quality has been defined through a comparison between the planned project schedule using Primavera Project Planner and each of the schedules obtained after each simulation run. The objective of this optimization is to minimize the delivery dates and to maximize yard usage.

Scenario 1: Module yard layout (bay area "A" only): The number of modules in a bay is a function of the sizes and the types of the modules. Each bay can contain between 1 and 10 modules, depending on their size. In addition, the module yard capacity, which is also a function of the module types and sizes, ranges from 75 to 186 modules, depending on their combination. Changing the module yard layout has a great impact on the improvement or deterioration of the schedule. If only bay area "A" is considered as the only layout available (see Figure 20) and all other constraints are left fixed, the simulation schedule in comparison to the project schedule, deteriorated (see Figure 21). The whole yard consists of 36 full bays and 3 half bays. Bay area "A" consists of 14 full bays only. As a result, 329 modules are fabricated in a space smaller than 40% of the whole yard capacity.

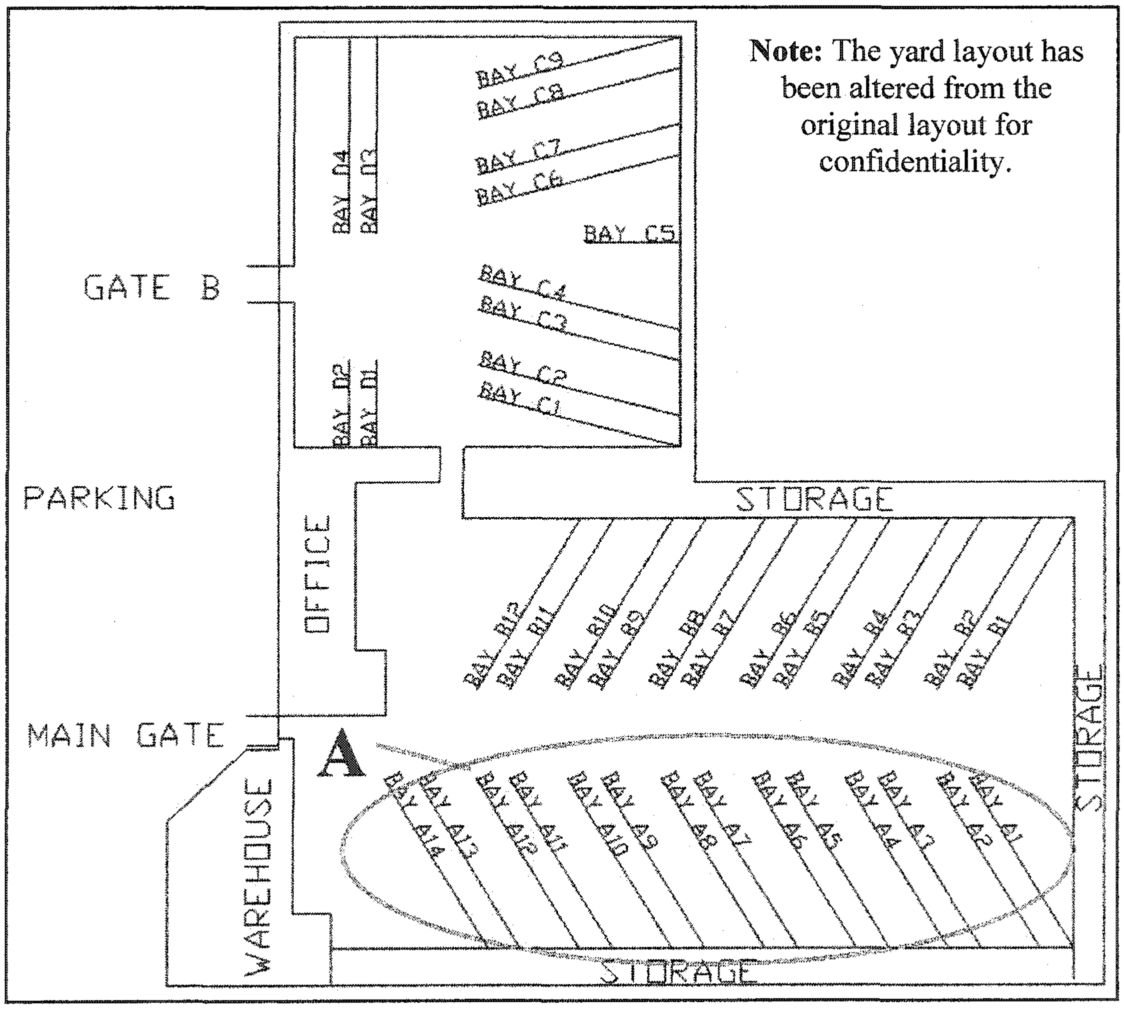
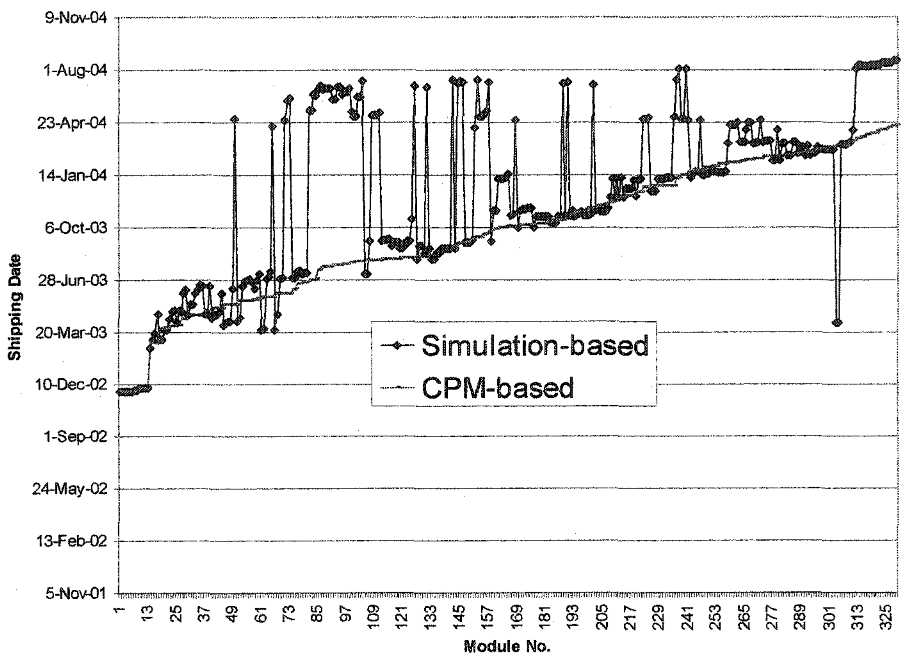


Figure 20 - PCL Module Yard Layout (bay area "A" only)

For this reason, the schedule proposed by the simulation shows a delay on shipping dates illustrated by the dramatic jumps on Figure 20. The manager may have the option to distribute the modules to bays in bay area "A" only while the remaining areas are kept empty for use when needed. Bay area "A", however, is not a good layout for the yard and further combinations should be explored to find the best layout.



Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 21 - CPM Schedule vs. Simulation Schedule (bay area "A" only)

Scenario 2: Module Yard Layout (bay areas "C" and "D" only): Similar to Scenario 1, this scenario provides the manager with the option of utilizing bay areas "C" and "D" only and leaving the remaining areas for use when needed. When only bay areas "C" and "D" are considered together (see Figure 22), the simulation schedule did not improve in comparison with the project schedule (see Figure 23). The whole yard consists of 36 full bays and 3 half bays. Bay areas "C" and "D" combined have only 10 full bays and 3 half bays. As a result, 329 modules are fabricated in a space of about 30% of the whole yard capacity. Similar to Scenario 1, the schedule proposed by the simulation shows a delay on shipping dates illustrated by the dramatic jumps on Figure 22.

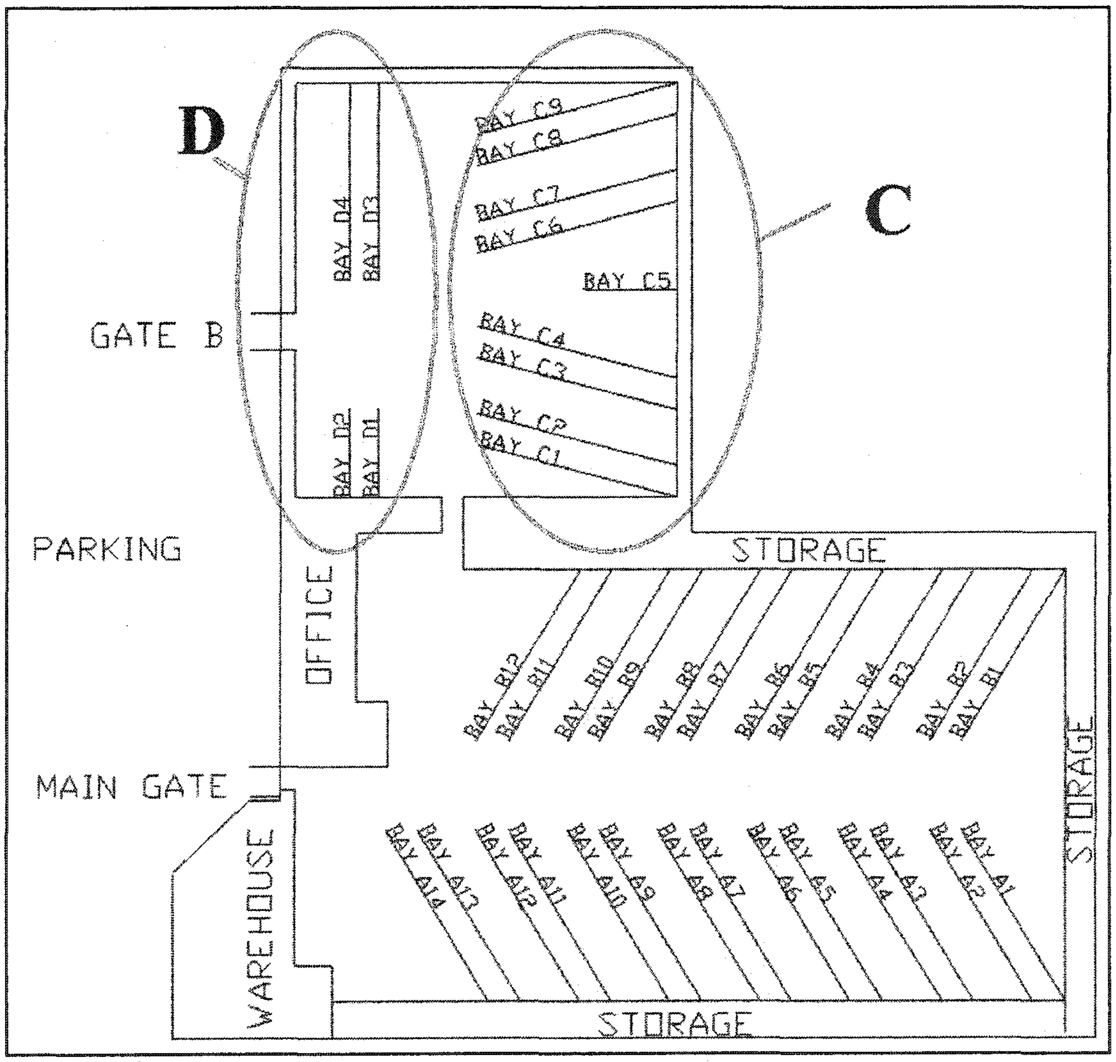
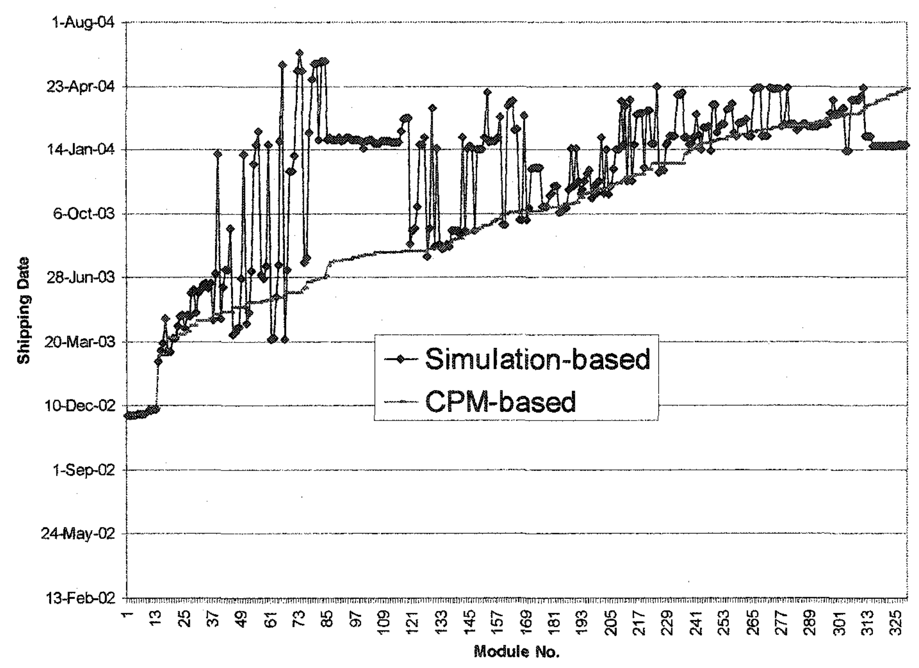


Figure 22 - PCL Module Yard Layout (bay area "C" and "D" only)

Therefore, bay areas "C" and "D" alone are not a good layout for the yard and further combinations should be sought to determine the best layout.



Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 23 - CPM Schedule vs. Simulation Schedule (bay area "C" and "D" only)

Scenario 3: Module yard layout (bay areas "A" and "B" only): Similar to the two previous scenarios, this scenario provides the manager with the option of utilizing bay areas "A" and "B" only and leaving the remaining areas for use when needed. If only bay areas "A" and "B" are considered together (see Figure 24), then the schedule proposed by the simulation will be similar to the schedule proposed when all four bay areas "A", "B", "C", and "D" were considered together (see Figure 25). The whole yard consists of 36 full bays and 3 half bays. Bay area "A" and bay area "B" combined have only of 26 full bays and 1 half bay. As a result, 329 modules are fabricated in a space of about 70% of the whole yard capacity.

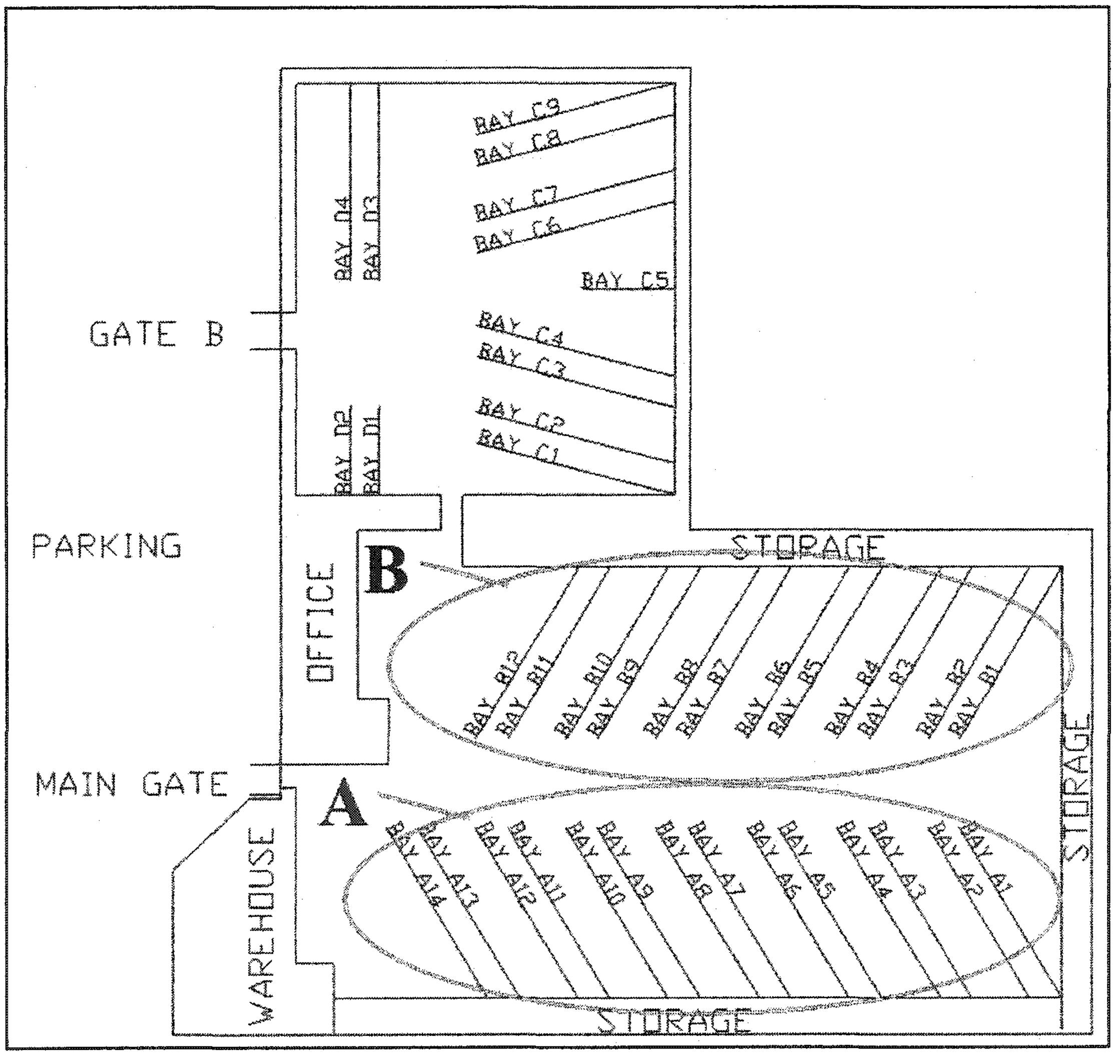
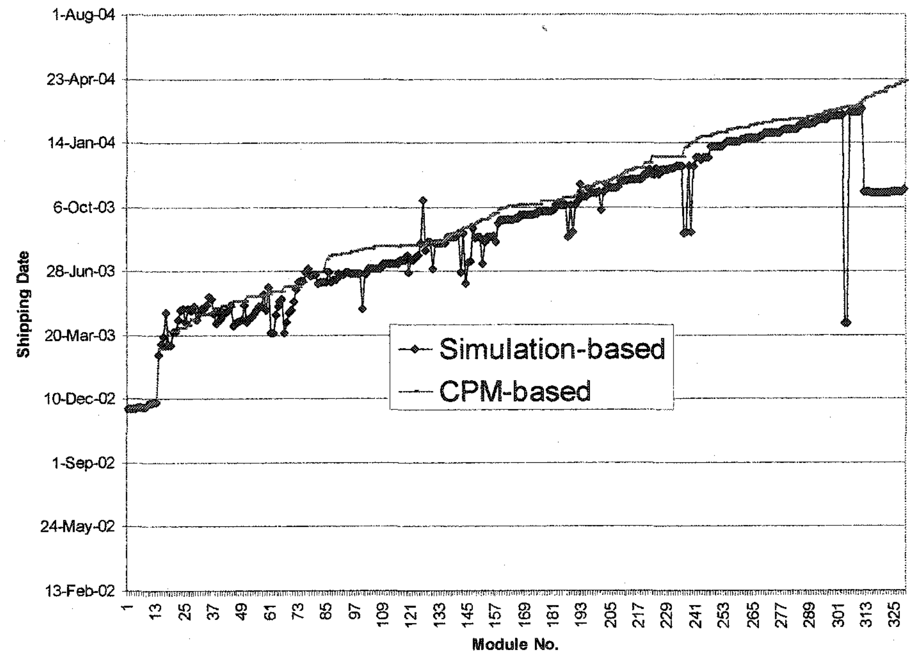


Figure 24 - PCL Module Yard Layout (bay area "A" and "B" only)

Therefore, bay areas “A” and “B” only are sufficient for accomplishing the fabrication of these 329 modules, which will leave approximately 30% of the yard space (bay areas “C” and “D”) available for other usage.



Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 25 - CPM Schedule vs. Simulation Schedule (bay area “A” and “B” only)

Scenario 4: Modules are shipped although the space in front is not empty: Module "A" has the potential of finishing its assembly process prior to module "B", which is located ahead of "A" in the same bay (see Figure 26).

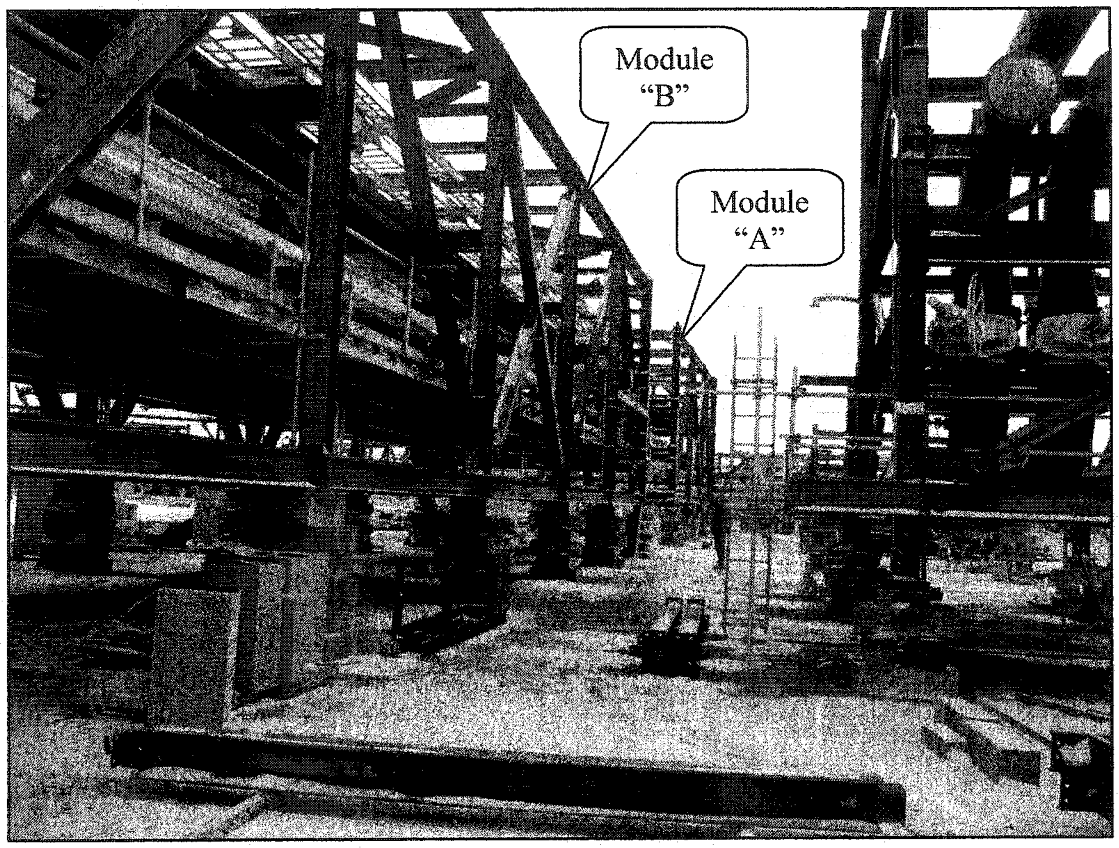
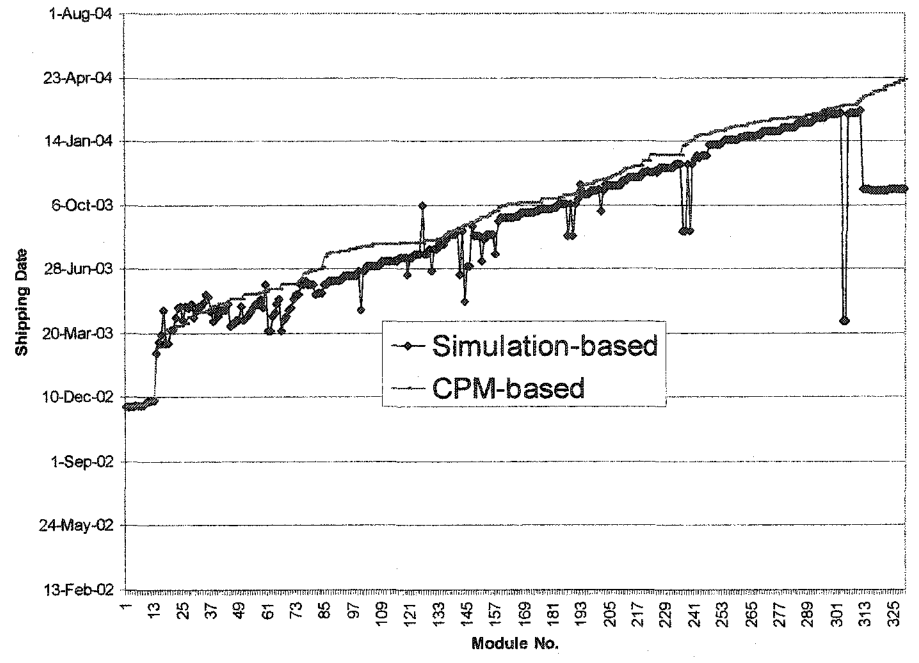


Figure 26 - PCL Module Yard (module "A" and module "B")

This scenario provides the manager with the option of employing larger cranes and more equipment in order to ship modules regardless of their position in the bay. When the simulation was performed without the constraint that modules could only be shipped when all the space in front of them is empty, it was found that the new proposed schedule is similar to the schedule proposed when the simulation followed the constraint about empty space (see Figure 27).

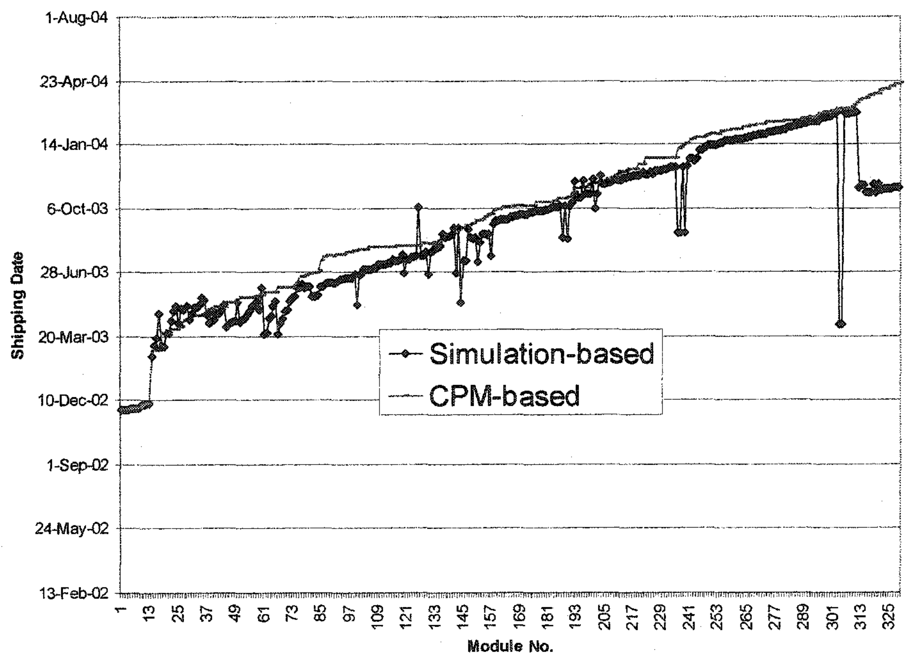


Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 27 - CPM Schedule vs. Simulation Schedule (Modules are Shipped Even Though the Space in front is not Empty)

This scenario supports the assumption that when the simulation follows the constraint about space in front is truly looking for the best places where to start the fabrication of each module. When no empty space is required to ship a module, the module can begin its process of fabrication anywhere within the yard and be shipped as soon as the process is complete. Since it is impossible to accomplish this in real life due to space limitations, it is better to employ a real life scenario in which modules wait until the space in front of them is completely empty for shipment. This process is engineered to obtain a schedule similar to the schedule that gives modules the ability to commence fabrication anywhere within the yard. The real life scenario schedule, however, will have the advantage of not using larger cranes or additional equipment to empty the bay, which will ultimately save money.

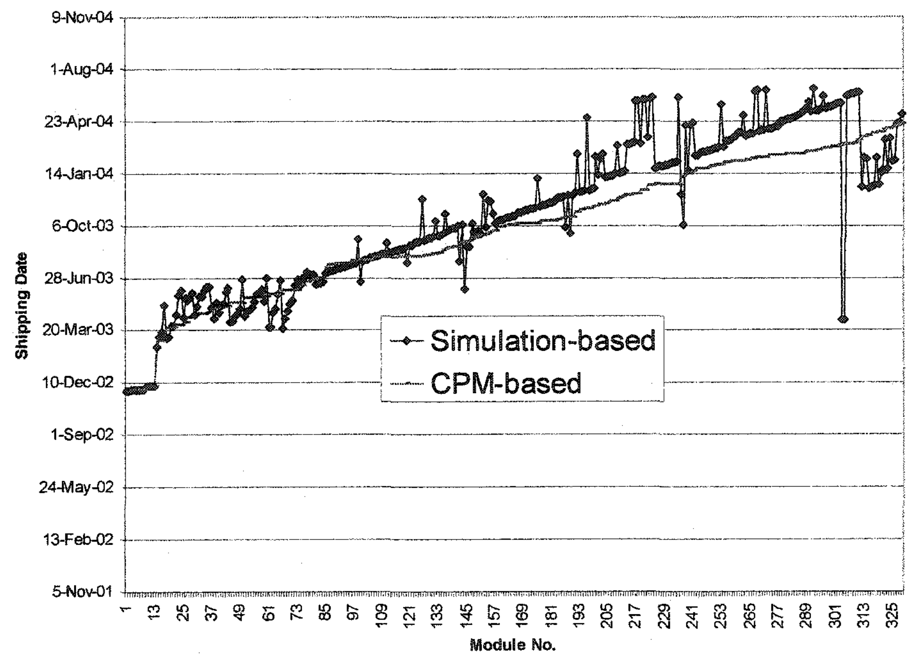
Scenario 5: Varying the number of shipments per day (from two to six): The number of shipments per day may affect the schedule. The main scenario allows six shipments per day; when this number was increased from seven to ten, no improvement was obtained. On the other hand, when the number of shipments per day was decreased, the scheduled appeared to remain in a consistent form regardless of the number of shipments varying between two to six per day (see Figure 28). In this case, two shipments per day seems to be the right choice; however, the scenario with only one shipment per day (Scenario 6) must be analyzed before arriving at this conclusion.



Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 28 - CPM Schedule vs. Simulation Schedule (Varying the Number of Shipments per Day from Two to Six)

Scenario 6: Varying the number of shipments per day to one shipment per day: Scenario 5 demonstrated that two shipments per day deliver modules is as sufficient as six shipments per day. When only one shipment was tested, the schedule showed a loss as compared to the project schedule (see Figure 29).

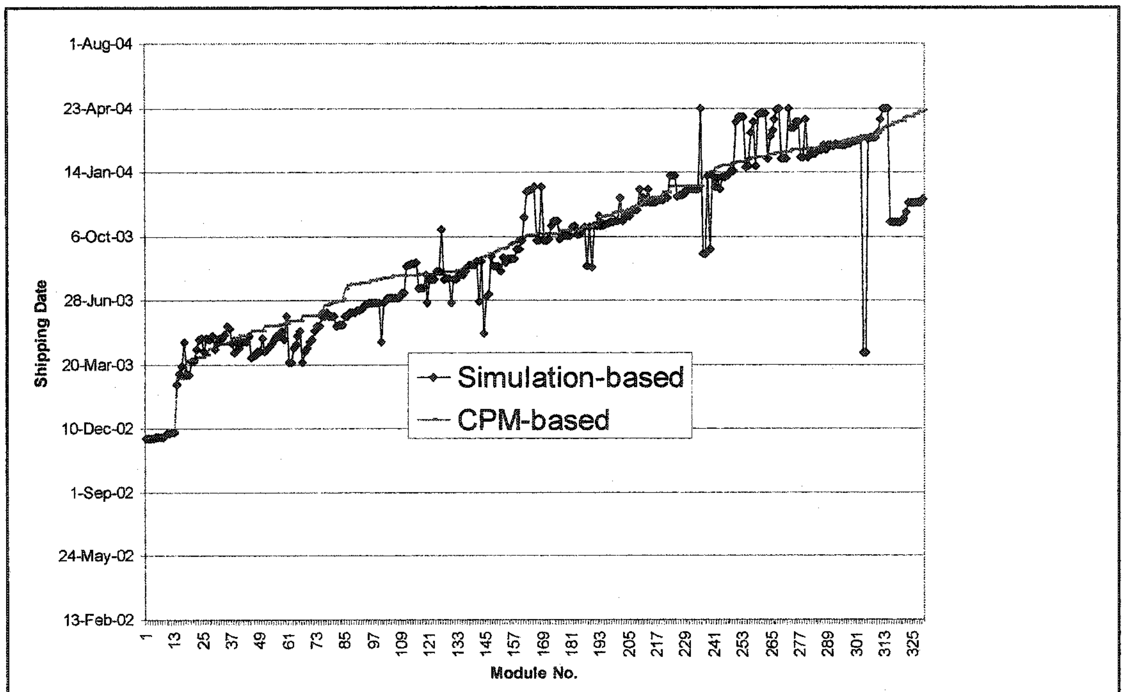


Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 29 - CPM Schedule vs. Simulation Schedule (One Shipment per Day)

It can be concluded that under the particular workload and circumstances, the schedule can be improved with only enough shipping equipment to perform two shipments per day lowering the actual cost of shipping.

Scenario 7: Modules are shipped immediately after assembly: The main scenario assumes that a module will wait for a maximum of five days before shipping once it has been completed. No significant change occurred by varying the number until this number was set to zero, that is, the module is shipped on the same day that it has been completed (see Figure 30).

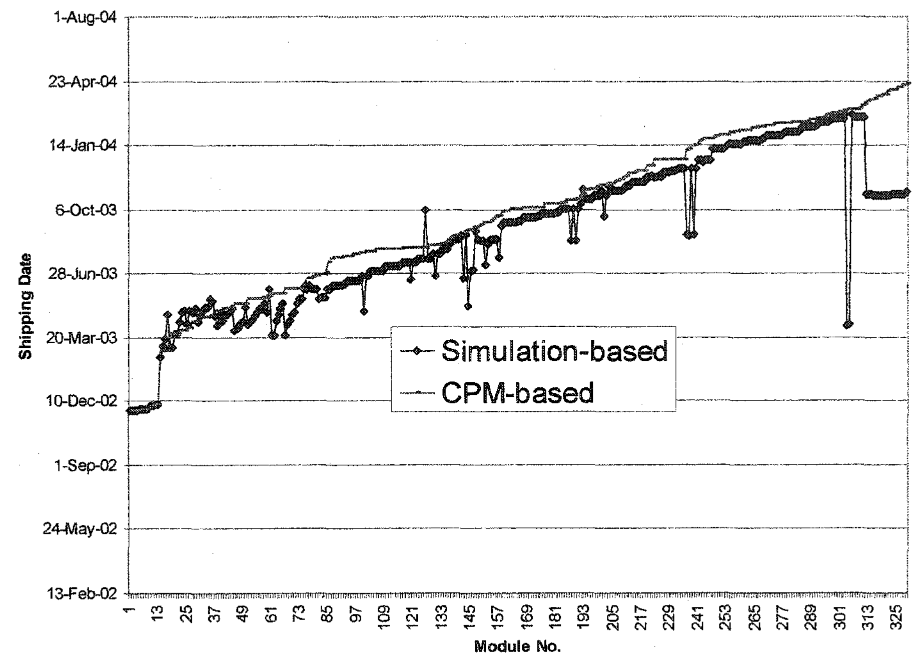


Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 30 - CPM Schedule vs. Simulation Schedule (Shipping Modules Immediately After Assembly)

Based on these findings, a module may only begin the fabrication process if its finishing date does not fall after the finishing date of a module already in fabrication within that same bay. When no tolerance is permitted, the model does not approximates to an optimize schedule in order to satisfy the constraint. Therefore, it is necessary to factor in the need for modules to wait for short periods of time before shipping in order to obtain the best results since the simulation will be more available for bay combinations.

Scenario 8: Modules are routed according to particular characteristics such as size rather than type: All the bays are able to handle any type or size of module. Based on the simulation, routing modules based on their size instead of on their type did not affect the schedule at all (see Figure 31).

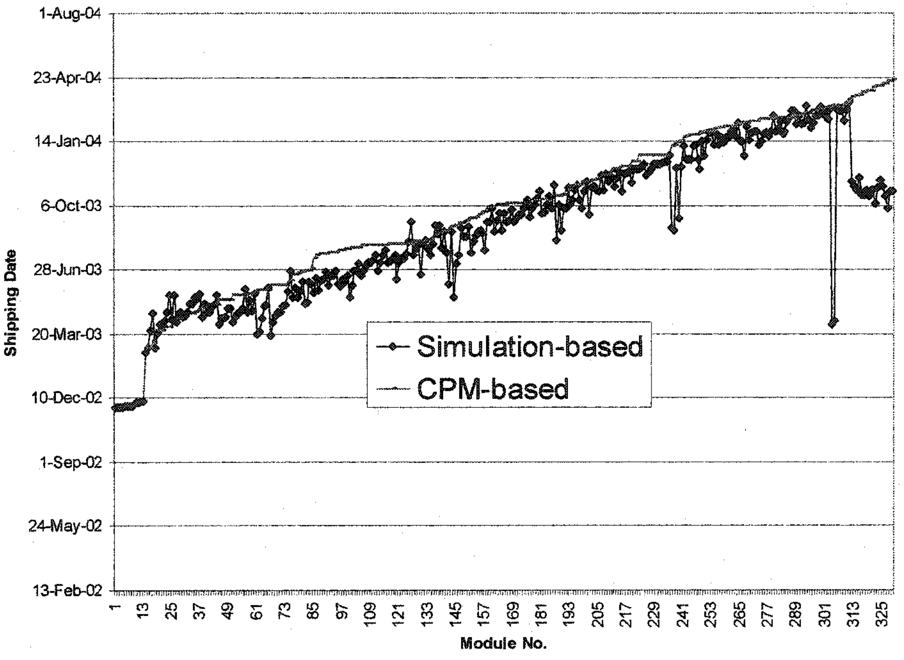


Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 31 - CPM Schedule vs. Simulation Schedule (Routing Modules Based on Size Rather than Type)

Therefore, the choice to route depending on a certain characteristic should be made based on what is more convenient for the company: to have all the modules that require the same type of equipment or material in close proximity to one another or to have modules close to one another based on their size.

Scenario 9: A distribution range is giving to the durations: The main scenario was run assuming fixed durations; however, in reality the durations may vary due to unpredictable circumstances. Scenario 9 in which durations were randomly given was tested (see Figure 32). Randomness has been created following standard input modeling techniques provided by PCL experts, based on min, max, and most likely values.



Note: The data used to obtain this graphs has been altered from the original data used by the company for confidentiality.

Figure 32 - CPM Schedule vs. Simulation Schedule (A Distribution Range is Given to the Durations)

The results show schedule improvement, albeit an improvement that varies due to the numerous duration changes, which occur as a result of the model's randomness. The simulation does provide a better schedule overall than the actual project schedule used thus far.

5.4 Summary and Conclusions

The results obtained from these scenarios indicate that the appropriate combination and determination of constraints will lead to better results in terms of schedule and budget. However, not all constraints can be altered in reality. For example, the yard layout depends on the yard size and may not be increased in a significant way due to physical limits; in this case, however, the layout can be reduced, meaning that the rest of the yard may be allocated for use elsewhere. An increase of the capability of shipping more modules per day was determined to be unnecessary since shipping only two modules per day already improved the schedule significantly. It is not necessary to determine a different way of removing a finished module even though the module in front in the bay is still under fabrication because it was proven that the schedule will not be more improved than it was when constrained by this requirement. It is recommended that modules having to wait a certain number of days before shipping rather than being shipped right away in order to obtain the best results, be modeled with a degree of flexibility. Finally, having a range in durations rather than fixed durations is a more realistic scenario.

5.5 Limitations

The development of the simulation model was limited by the data provided by PCL as well as the short-term needs that PCL prioritized for this stage of the research as the most important outputs. Also, the modeling of different crews and their interactions when performing the subtasks has not been evaluated due to the previously mentioned prioritization for short-term needs. Randomness has been created from the information provided. However, no additional information was available since this research was performed at the same time that the modules were being fabricated for the first time under the circumstances mentioned before regarding the logical and physical constraints.

Chapter 6. Conclusion

6.1 Summary of Research

This thesis presented a simulation-based technique to improve schedules of module assembly yards. The technique deals with the challenge of meeting the delivery dates set by the client and performing regular schedule updates. Developing a simulation-based scheduling process for use in the modularization industry has much potential as there is a substantial need to distribute modules in the yard more efficiently and effectively. The simulation-based technique provides a convenient and easy-to-use tool for allocating modules. The implementation of this research concept for generating a constrained schedule for use in the modularization industry was made possible using *Simphony* (AbouRizk and Hajjar 1998), a general purpose simulation tool that, among other functions, provides flexibility for module allocation. The simulation-based schedule integrates the given information into a database format, which processed the data using Visual Basic Application in Excel and the simulation model developed using *Simphony*.

There are a number of advantages in using the simulation-based scheduling. Simulation-based scheduling contributes to decision-making by providing an instrument to evaluate various scenarios of interest and provides perspective. Actual physical and logical constraints as well as the logical and heuristic rules used by yard superintendents were analyzed and incorporated into this approach. The scheduling rules employed in this research create a feasible schedule. The approach begins by identifying and prioritizing the modules to be processed. The resource availability is checked and modules are scheduled one by one in order of priority until all the modules are scheduled. The model allows experimentation with the rules. This experimentation then provides scenarios out of which the best schedule is obtained based on actual yard and resource limitations. The experimentation presented in this thesis was done through a case study undertaken in cooperation with PCL in Edmonton, Canada. The results obtained demonstrated significant improvements in the module assembly schedule when compared to traditional scheduling techniques using CPM. It has been estimated that obtaining the schedule and performing the allocation of modules using the simulation-based technique rather than manually allocating modules using Primavera could save up to $12,500 per year. These savings correspond to the employ of less effort while updating the schedule (less man-hours spent during the update process). Cost savings due to earlier delivery times may vary depending on the total time saved per module.

6.2 Research Contributions

The research contributions are as follow:

- Development of an integrated approach for optimizing the scheduling process of a module yard, and
- Automation of this approach and implementation with the industry.

These contributions have been achieved by:

- Developing a simulation model for a module assembly yard.
- Integrating simulation with schedule for instant evaluation of yard utilization and schedule updates.
- Integrating the simulation model with uncertainty in the schedule.
- Integrating the simulation with the database, as a result two new *Simphony* elements have been created and added to the simulation model to automate this process.
- Incorporating graphics for yard layout and yard utilization.
- Automating the schedule using VBA when updating the inputs and obtaining the results.

6.3 Recommendations for Future Research

The simulation-based technique demonstrates the feasibility of managing a module assembly yard. Using the approach described in this thesis, future work can be undertaken to study the effects of incorporating different calendars and shifts into modules when scheduling their assembly process as well as to provide the auto-generate layout graphical interface in AutoCAD.

The following areas are available for future research:

- *Scheduling based on calendars:* The changes in schedules using a different calendar for each module, allows for various shift configurations to require examination as this will provide better perception into the proper selection of scheduling calendars depending on module characteristics and conditions.
- *User interface enhancement:* Data input, simulation running, and the auto-generate layout graphical interface can be further enhanced to provide a more 'user-friendly' interface. The data input interface can be enhanced to include database forms that display calendars in which dates ate chosen by the user by clicking on the desired date rather than manually typing the date. The simulation can be run using the database or the VBA instead of actually opening *Simphony* and performing the run manually. The auto-generate layout graphical interface can be improved by using an actual AutoCAD module yard layout.
- *3D model:* The 3D model may be front-loaded with module assembly yard information as well as information and characteristics for each module.
- *4D model:* Including the element of time to the 3D model where the dates obtained during the simulation would, therefore, add an extra dimension.
- *Shipping:* The model provides the manager with the necessary information to schedule shipping dates. However, the model does not consider the delivery method nor the delivery times from Nisku, Alberta to Fort McMurray, Alberta. A further investigation into these issues may improve the overall modularization process.
- *Lean theory:* Since a module assembly yard schedule is based on the appropriate utilization of resources, the effect of lean theory is an available avenue of investigation.

- *Resource utilization and leveling:* Resources play an important role in a module assembly yard. Making the model capable of performing resource leveling can further improve the results.
- *Capacity and productivity studies:* A method of computing the complexities of the work based on a set of features can be taken up; a neural network-based approach can be utilized to obtain expected durations for each of the subtasks that can be used for scheduling.
- *Weather effects:* Located in an open area in Nisku, Alberta, the module assembly yard can face the effects of the weather to which module fabrication is exposed. A study of the fabrication of modules under extreme weather conditions can be carefully analyzed to improve productivity during winter.
- *Genetic algorithms:* The development of a module assembly yard schedule, which includes genetic algorithms for enhancing the simulation-based technique is another available future research.

References

AbouRizk, S. M. (1992). "A Stochastic Bidding Game for Construction Management." In *Second Canadian Conference on Computing in Civil Engineering*, CSCE, Ottawa, Ontario, 576-587.

AbouRizk, S. M. (2000). *Simphony's User Guide*, NSERC/Alberta Construction Industry Research Chair, Edmonton, Canada.

Au, Y., Bostleman, R., and Parti, E. (1969). "Construction Management Game – Deterministic Model." In *ASCE Journal of Construction Division*. Vol. 95, 25-38.

Banks, J. (1998). *Handbook of simulation*. John Wiley and Sons, New York, N.Y.

Borcherding, John D. (1977). "Cost Control Simulation and Decision Making." In *Journal of the Construction Division*. Vol. 103, No. 4: 577-591.

Burke, G., and Miller, R. (1998). "Modularization speeds construction." In *Power Engineering*, Vol. 102. No. 1: 20-22.

Cheng, M. Y., and O'Connor, J. T. (1996). "ArcSite: Enhanced GIS for Construction Site Layout." In *Journal of Construction Engineering and Management*, Vol. 122, No. 4: 329-336.

Chong, C. S., A. I. Sivakumar, and Gay, R. (2003). "Simulation Based Scheduling Using a Two-Pass Approach." In *Proceedings of the 2003 Winter Simulation Conference*, S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (eds.), 1433-1439.

Hajjar, D., and AbouRizk, S. M. (1998). "A framework for applying simulation in construction." In *Canadian Journal of Civil Engineering*. 25(3): 604–617.

Hajjar, D., and AbouRizk, S. M. (1999). "*Simphony*: An Environment for Building Special Purpose Construction Simulation Tools." In *Proceedings of the Winter Simulation Conference*. IEEE, Piscataway, N.J: 998-1006.

Hajjar, D., and AbouRizk, S. M. (2002). "Unified Modeling Methodology for Construction Simulation." In *Journal of Construction Engineering and Management*, Vol. 128, No. 2: 174–185.

Halpin, D. W. (1973). *An investigation of the use of simulation networks for modeling construction operations*. PhD thesis, University of Illinois, at Urbana-Champaign, Ill.

Halpin, D. W., and Woodhead, R. W. (1973). *Constructo - A Heuristic Game for Construction Management*, University of Illinois Press, Champaign, Illinois. pp. 195.

Hegazy, T., and Elbeltagi, E. (1999). "EVOSITE: Evolution-Based Model for Site Layout Planning." In *Journal of Computer in Civil Engineering*, Vol. 13, No. 3: 198–206.

Hegazy, T., and Elbeltagi, E. (2001). "A Hybrid AI-Based System for Site Layout Planning in Construction." In *Computer-Aided Civil and Infrastructure Engineering*, Vol. 16: 79–93.

Hegazy, T., and Ersahin, T. (2001). "Simplified Spreadsheet Solutions. I: Subcontractor Information System." In *Journal of Construction Engineering and Management*, ASCE. Vol. 127, No. 6: 461–468.

Hegazy, T., and Ersahin, T. (2001a). "Simplified Spreadsheet Solutions. II: Overall Schedule Optimization." In *Journal of Construction Engineering and Management*, ASCE. Vol. 127, No. 6: 469–475.

Laufer, A. (1996). *Simultaneous management: managing projects in a dynamic environment*. AMA-COM, American Management Association.

Li, S. (1996). "New Approach for Optimization of Overall Construction Schedule." In *Journal of Construction Engineering and Management*, ASCE. Vol. 122, No. 1: 7–13.

Maru, A., and Kawahata, J. (2002). "Hitachi Modularization Technology." In *Nuclear Plant Journal*, Vol. 20, No. 5: 39-42.

Mawdesley, M. J., Al-jibouri, S. H., and Yang, H. (2002). "Genetic Algorithms for Construction Site Layout in Project Planning." In *Journal of Construction Engineering and Management*, Vol. 128, No. 5: 418–426.

Oglesby, C. H., Parker, H. W., and Howell, G. A. (1989). *Productivity Improvement in Construction*. McGraw-Hill (Series in Engineering and Project Management), New York, N.Y.

Osman, H. M., Georgy, M. E., and Ibrahim, M. E. (2003). "A hybrid CAD-based construction site layout planning system using genetic algorithms." In *Automation in Construction*, Vol. 12: 749-764.

Panwalkar, S. S., and Iskander, W. (1977). "A Survey of Scheduling Rules." In *International Journal of Operations Research*, Vol. 25, No. 1: 45-61.

PCL (2003). "Pipe Fabrication and Module Construction Pamphlet." *PCL Industrial Constructors, Nisku, Alberta.*

Schimmoller, B. (1998). "Power plants go modular." In *Power Engineering*, Vol. 102. No. 1: 14-18.

Tam, C. M., and Leung A. W . T. (2002). "Genetic Algorithm Modeling Aided with 3D Visualization in Optimizing Construction Site Facility Layout." In *Proceedings of the International Council for Research and Innovation in Building and Construction Conference*. Aarhus, Denmark: 1-9.

Teicholz, P. M. (1963). *A simulation approach to the selection of construction equipment*. PhD thesis, Stanford University, Stanford, CA.

Web 1 - *Simulation in Construction Using CYCLONE and MicroCYCLONE*. < http://bridge.ecn.purdue.edu/CEM/Sim/> (June 2004).

Yeh, I.-C., (1995). "Construction-Site Layout Using Annealed Neural Network." In *Journal of Computer in Civil Engineering*, Vol. 9, No. 3: 201–208.

Appendix 1: *Simphony's* Common Template Modifications

In Chapter 4, the creation of two new elements and the modification of three existing elements from *Simphony*'s Common Template were explained. In this appendix, the code written to create and modify those elements is listed.

Creation of the New Elements

Database Link Element:

```
Public Function DatabaseLink_OnCreate(ob As
CFCSim_ModelingElementInstance, x As Single, y As Single) As Boolean
ob.OnCreate x,y,True
DatabaseLink_OnCreate=True
   Dim myDB As Database
   Dim myRS As Recordset
   Dim numAttr As Integer
   Dim i As Integer
   Dim j As Integer
   Dim k As Integer
ob.AddAttribute "Fired","Entites CreateEntd so far",CFC_Numeric,
CFC_Single,CFC_Hidden
ob.AddAttribute "Database","Database
Source",CFC_Text,CFC_Single,CFC_ReadWrite
ob.AddAttribute "Table1","Table/Query for Product Definition",CFC_Text,
CFC_Single,CFC_ReadWrite
ob.AddAttribute "NumRows","Number of Rows in the Table",CFC_Numeric,
CFC_Single, CFC_ReadOnly
ob.AddAttribute "NumColumns","Number of Columns in the Table",CFC_Numeric,
CFC_Single, CFC_ReadOnly
```

```
                    ob("Name")="Name"
                    ob("Database")="C:\"
                    ob("Table1")="Query"
ob.AddAttribute "CAttr", "",CFC_Array, CFC_Table, CFC_ReadOnly
ob.AddAttribute "NumAttr","Number of Attributes",CFC_Numeric,
CFC_Single,CFC_Hidden
ob.AddAttribute "NumCom","Number of Components",CFC_Numeric,
CFC_Single,CFC_Hidden
          ob.SetNumCoordinates 2
          ob.CoordinatesX(0)=x
          ob.CoordinatesY(0)=y
          ob.CoordinatesX(1)=x+90
          ob.CoordinatesY(1)=y+50
ob.AddConnectionPoint "Out",x+100,y+25,COutput,5
End Function


Public Sub DatabaseLink_OnSimulationInitialize(ob As
CFCSim_ModelingElementInstance)
                    Dim numAttr As Integer
                    Dim i As Integer
                    Dim j As Integer
                    Dim myDB As Database
                    Dim myRS As Recordset
'Setup database connection
Set myDB = OpenDatabase(ob!Database)
Set myRS = myDB.OpenRecordset(ob!Table1, dbOpenDynaset)
                    numAttr=CInt(myRS.Fields.Count)
                    ob("NumAttr")=numAttr
                    myRS.MoveLast
                    ob("NumCom")=myRS.RecordCount
ob("NumColumns")=ob("NumAttr")
```

```
                    ob("NumRows")=ob("NumCom")
ob("CAttr").SetRC(ob("NumCom"),ob("NumAttr"))
                    'Read attribute names
                    For i=0 To ob("NumAttr")-1
ob("CAttr").ColumnLabel(CInt(i))= CStr(myRS.Fields(CInt(i)).Name)
ob.AddAttribute "Attr" & i & "Name",CStr(myRS.Fields(CInt(i)).Name),CFC_Text,
CFC_Single, CFC_Hidden
                    Next I
                    'Clean the Table
                    With ob("CAttr")
                              For i=0 To ob("NumCom")-1
                                        For j=0 To ob("NumAttr")-1
                                        .ValueRC(i,j)=""
                                        Next j
                              Next i
                    End With
          ob.AddEvent "FireEntity"
End Sub
```

```
Public Sub DatabaseLink_OnSimulationInitializeRun(ob As
CFCSim_ModelingElementInstance, RunNum As Integer)
Dim newEntity As CFCSim_Entity
          Dim myDB As Database
          Dim myRS As Recordset
          Dim i As Integer
Set myDB = OpenDatabase(ob!Database)
Set myRS = myDB.OpenRecordset(ob!Table1, dbOpenDynaset)
          myRS.MoveFirst
          Set newEntity = ob.AddEntity
          ob("Fired")=0
With ob("CAttr")
```

```
          For i=0 To ob("NumAttr")-1
If IsNull(myRS.Fields(i).Value) Or IsEmpty(myRS.Fields(i).Value) Then
                                   .ValueRC(ob("Fired"),CInt(i))=""
                    Else
          .ValueRC(ob("Fired"),CInt(i))=myRS.Fields(i).Value
                    newEntity("NumColumns")=ob("NumAttr")
                    newEntity("NumRows")=ob("NumCom")
          newEntity(ob("CAttr").ColumnLabel(CInt(i)))=myRS.Fields(i).Value
Tracer.Trace "Entity: '" & newEntity.Id & "' has been assigned a value of '" &
newEntity(ob("CAttr").ColumnLabel(CInt(i))) & "' For attribute: '"  &
ob("CAttr").ColumnLabel(CInt(i)) & "'", "Simulation"
                    End If
          Next i
End With
ob.ScheduleEvent ob.AddEntity,"FireEntity", myRS.Fields(4).Value '(4) represents
the fifth column on the database query where the EarlyStartDate is stored
          ob("Fired")=0
```
End Sub

```
Public Sub DatabaseLink_OnSimulationProcessEvent(ob As
CFCSim_ModelingElementInstance, MyEvent As String, Entity As CFCSim_Entity)
          Dim newEntity As CFCSim_Entity
          Dim myDB As Database
          Dim myRS As Recordset
          Dim myProcess As Recordset
          Dim numAttr As Integer
          Dim i, j, k As Integer
   i=0
   j=0
Set myDB = OpenDatabase(ob!Database)
Set myRS = myDB.OpenRecordset(ob!Table1, dbOpenDynaset)
```

```
          numAttr=CInt(myRS.Fields.Count)
          ob("NumAttr")=numAttr
          myRS.MoveLast
          ob("NumCom")=myRS.RecordCount
ob("CAttr").SetRC(ob("NumCom"),ob("NumAttr"))
          Tracer.Trace "Number of attributes: " & ob("NumAttr")
          Tracer.Trace "Number of components: " & ob("NumCom")
          myRS.MoveFirst
'Read attribute values
If ob("Fired")> ob("NumCom")-1 Then Exit Sub
ob("fired")=ob("fired")+1
                    Set newEntity = ob.AddEntity
                    If ob("fired")>1 Then
                              For j=2 To ob("fired") 'j=2 to start from the second entity
                              myRS.MoveNext
                              Next j
End If
With ob("CAttr")
          For i=0 To ob("NumAttr")-1
                    'Read component attributes
          If IsNull(myRS.Fields(i).Value) Or IsEmpty(myRS.Fields(i).Value) Then
                              .ValueRC(ob("Fired")-1,CInt(i))=""
          Else
                              .ValueRC(ob("Fired")-1,CInt(i))=myRS.Fields(i).Value
newEntity("NumColumns")=ob("NumAttr")
newEntity("NumRows")=ob("NumCom")
newEntity(ob("CAttr").ColumnLabel(CInt(i)))=myRS.Fields(i).Value
Tracer.Trace "Entity: '" & newEntity.Id & "' has been assigned a value of '" &
newEntity(ob("CAttr").ColumnLabel(CInt(i))) & "' For attribute: '"  &
ob("CAttr").ColumnLabel(CInt(i)) & "'", "Simulation"
          End If
```

```
        Next i
End With
                    ob.TransferOut newEntity
                    If ob("fired")<ob("NumCom") Then
                        myRS.MoveNext
                    End If
ob.ScheduleEvent Entity, "FireEntity", myRS.Fields(4).Value – SimTime '(4)
represents the fifth column on the database query where the EarlyStartDate is stored
Tracer.Trace "Entity: " & newEntity.Id & "  Created","Simulation"
End Sub
```

Results Element: There are basically two different subroutines from the Database Link Element: OnSimulationTransferIn, and OnSimulationPostRun.

```
Public Sub Results_OnSimulationTransferIn(ob As
CFCSim_ModelingElementInstance, Entity As CFCSim_Entity, SrcCp As
CFCSim_ConnectionPoint, DstCp As CFCSim_ConnectionPoint)
                Dim numAttr As Integer
                Dim i As Integer
                Dim j As Integer
numAttr=ob.CurrentEntity.Attr("NumColumns")
ob("NumColumns")=numAttr
ob("NumRows")=ob.CurrentEntity.Attr("NumRows")
        With ob("CAttr")
                For i=0 To ob("NumRows")-1
                    If .ValueRC(i,0)="" Then
                        For j=0 To ob("NumColumns")-1
If IsNull(ob.CurrentEntity.Attr(ob("CAttr").ColumnLabel(CInt(j)))) Or
IsEmpty(ob.CurrentEntity.Attr(ob("CAttr").ColumnLabel(CInt(j)))) Then
                                .ValueRC(i,j)="0"
Else
```

```
.ValueRC(i,j)=ob.CurrentEntity.Attr(ob("CAttr").ColumnLabel(CInt(j)))
End If
                    Next j
                    Exit Sub
               End If
          Next i
     End With
End Sub

Public Sub Results_OnSimulationPostRun(ob As
CFCSim_ModelingElementInstance, RunNum As Integer)
          Dim numAttr As Integer
          Dim i As Integer
          Dim j As Integer
          Dim sql As String
          Dim myDB As Database
          Dim myRS As Recordset
      'Setup database connection
      Set myDB = OpenDatabase(ob!Database)
      'Clean the Results Table
      If RunNum=1 Then
               sql="Delete * From " & CStr(ob!Table1_Results)
          myDB.Execute sql
      End If
   Set myRS = myDB.OpenRecordset(ob!Table1_Results, dbOpenDynaset,
dbAppendOnly)
      numAttr=ob.CurrentEntity.Attr("NumColumns")
      ob("NumColumns")=numAttr
      ob("NumRows")=ob.CurrentEntity.Attr("NumRows")
     With ob("CAttr")
          For i=0 To ob("NumRows")-1
```

```
                        myRS.AddNew
                        For j=0 To ob("NumColumns")-1
myRS(CStr(ob("CAttr").ColumnLabel(CInt(j))))= ob("CAttr").ValueRC(i,j)
                        Next j
                        myRS.Update
                Next i
        End With
End Sub
```

Simphony Common Template Modifications

Three of the elements found in the existing common template were modified to suit the special requirements of this simulation. The modifications to those elements are written in bold font in the following *Simphony* code.

Declare Resources Element:

Public Function Resource_OnCreate(ob As CFCSim_ModelingElementInstance, x As Single, y As Single) As Boolean

```
Resource_OnCreate=True
ob.OnCreate x,y,True
ob.AddAttribute "ResName","Resource Description",CFC_Text,
CFC_Single,CFC_ReadWrite
ob.AddAttribute "Total","Total Number of Resources",CFC_Numeric,CFC_Single,
CFC_ReadWrite,0,1000000
ob.AddAttribute "Current","Current Number of Available Resources",CFC_Numeric,
CFC_Single, CFC_ReadOnly
'This attribute will keep track of the finishing date of the module in the back of the
bay
```

ob.AddAttribute "**EndBackModule**","End of Module in the Back of the Bay",CFC_Numeric, CFC_Single,CFC_ReadOnly

```
'This attribute will keep track of the finishing date of the module to be built
ob.AddAttribute "FinishCurrentModule","End of Module to be
built",CFC_Numeric, CFC_Single,CFC_ReadOnly
ob.AddAttribute "ModuleCounter","Number of Modules in the Bay",CFC_Numeric,
CFC_Single,CFC_ReadOnly
ob.AddAttribute "SpaceInFront","Space in front for more Modules",CFC_Numeric,
CFC_Single,CFC_ReadOnly
    ob("ResName")= "Res"
    ob("Total")=1
    ob("Current")=1
ob.AddStatistic "Utilization","Resource Utilization",True,False
End Function

Public Sub Resource_OnSimulationInitializeRun(ob As
CFCSim_ModelingElementInstance, RunNum As Integer)
    ob("Current")=ob("total")
    ob("EndBackModule")=1000
    ob("FinishCurrentModule")=1000
    ob("ModuleCounter")=0
    ob("SpaceInFront")=1
If ob!Total<>0 Then ob.stat("Utilization").Collect 100 * (1-
(ob("Current")/ob("Total")))
End Sub
```

Waiting File Element:

```
Public Sub Waiting_File_OnSimulationProcessEvent(ob As
CFCSim_ModelingElementInstance, MyEvent As String, Entity As CFCSim_Entity
Select Case MyEvent
'**************
' TO DO:
```

```
' To optimize the processing time, a check of all the entities in the file should NOT be
done
' unless a change in the number of available resource in any of the declared resources
has happened
' if the check is triggered by an entity added to the file, it should only check the
availability of resources
' for that entity
'***************
                Case "Check"
                        Dim SrvEnt As CFCSim_Entity
                        Dim RqstElmnt As CFCSim_ModelingElementInstance
                        Dim Rqst As CFCSim_ModelingElementInstance
                        Dim ResElmnt As CFCSim_ModelingElementInstance
                        Dim ResAvailable As Boolean
                        'To request resources if the module will finish on time
                        'Dim RqstEnd As CFCSim_ModelingElementInstance
                        ob.DeleteEntity Entity
Tracer.Trace "*******File Check Started *******","res","File"
        With ob.File("Waiting_File")
                If .Length= 0 Then Exit Sub
                        .MoveFirst
                                '*** Check the waiting entities one by one
                                While (.EOF=False And .Length>0)
                                ResAvailable=True
                                Set SrvEnt=.entity
                                Set RqstElmnt = SrvEnt("CEM_Common_RqstElmnt")
Tracer.Trace "File Length=" & .Length & " and EOF= "& .EOF & " and Current Ent
is # " & SrvEnt.Id ,"res","File"
                '*** If the request element from where the entity came is satisfied
                '               for all  the single requests inside it Then
                '*** grant the request for that element
```

```
          Select Case RqstElmnt("RqstType")
                    Case "ALL"     '*** All resources are required
                              '*** First check if all the resources are available
                              For Each Rqst In RqstElmnt.ChildElements
          'The following line is tTo allow linking the required number of resources
          ' to entity attributes
          'Rqst.OnSimulationTransferIn SrvEnt,Nothing,Nothing
If Rqst("ResName")<>"**Linked to Entity Attribute**" Then  '*** Case not linked
                                        Set ResElmnt=Rqst("ResOb").Reference
Else                          '**** Case linked
                                        For Each ResElmnt In Elements
'ob.Parent.ChildElements
                                                  If  ResElmnt.ElementType="Resource" Then
                              If ResElmnt("ResName")=SrvEnt(Rqst("EntAttr")) Then Exit
For
                                                  End If
                                        Next
End If
                                        If        ResElmnt("Current")<Rqst("NumRes") Then
                                                            ResAvailable=False
                              Exit For
                                        End If
                              Next
                                        If ResAvailable Then
'*** If all are available then decrease each's availabe number by the requested number
                              For Each Rqst In RqstElmnt.ChildElements
          'The following line is tTo allow linking the required number of resources
          ' to entity attributes
          'Rqst.OnSimulationTransferIn SrvEnt,Nothing,Nothing
If Rqst("ResName")<>"**Linked to Entity Attribute**" Then  '*** Case not linked
                                                            Set ResElmnt=Rqst("ResOb").Reference
```

```
Else                    '**** Case linked
                        For Each ResElmnt In Elements 'ob.Parent.ChildElements
                                If  ResElmnt.ElementType="Resource" Then
                If ResElmnt("ResName")=SrvEnt(Rqst("EntAttr")) Then Exit For
                                End If
                        Next
End If
        ResElmnt("Current")=ResElmnt("Current")-Rqst("NumRes")
                                        If ResElmnt("Total")<>0 Then
ResElmnt.stat("Utilization").Collect 100 * (1-
(ResElmnt("Current")/ResElmnt("Total")))
                        Next
'*** Then remove the entity from the file and schedule a Granted Request event in its
original capture element
                                        .Remove SrvEnt
RqstElmnt.ScheduleEvent SrvEnt,"RqstGrntd",0

Tracer.Trace ">>>>>>>Rqst. of ALL res. GRANTED for Entity #"&
SrvEnt.Id,"res","File","Granted"
                '*** If no resources are available for this entity move to the next one
                                        Else
Tracer.Trace "Reqst ALL denied and moving to the next ent","res","File","Denied"
                                        .MoveNext
                                        End If
                Case "ANY"  '*** Any of the requesed resources is enough
                                '*** First check if any of the resources is available
                        For Each Rqst In RqstElmnt.ChildElements
If Rqst("ResName")<>"**Linked to Entity Attribute**" Then  '*** Case not linked
                                        Set ResElmnt=Rqst("ResOb").Reference
Else                    '**** Case linked
```

```
                                For Each ResElmnt In Elements
'ob.Parent.ChildElements
                                        If  ResElmnt.ElementType="Resource" Then
                                If ResElmnt("ResName")=SrvEnt(Rqst("EntAttr"))
Then Exit For
                                        End If
                                Next
End If
'Quick fix for resource request problem
'to enable requesting # of resources based on entity attribute (formula)
'====================
Dim numRes As Integer
If SrvEnt("Auxiliary")<1 Then
If Rqst.Attr("NumRes").Calculation=CFC_Formula Then
                If Not IsNumeric(Rqst("NumRes")) Then
                        numRes=SrvEnt(Rqst("NumRes"))
                Else
                        numRes=SrvEnt("UnitsRequested")
                End If
Else
                numRes=Rqst("NumRes")
End If
Else
If SrvEnt("Auxiliary")=1 Then ''' to capture space and resources for shipping
        If Not IsNumeric(Rqst("NumRes")) Then
                numRes=SrvEnt(Rqst("NumRes"))
        Else
                numRes=Rqst("NumRes")
        End If
End If
If SrvEnt("Auxiliary")=3 Then ''' to capture Manhours_Struct
If Rqst.Attr("NumRes").Calculation=CFC_Formula Then
        If Not IsNumeric(Rqst("NumRes")) Then
                numRes=SrvEnt(Rqst("Manhours_Struct"))
        Else
                numRes=Rqst("NumRes")
        End If
End If
End If
If SrvEnt("Auxiliary")=4 Then ''' to capture Manhours_Piping
If Rqst.Attr("NumRes").Calculation=CFC_Formula Then
        If Not IsNumeric(Rqst("NumRes")) Then
                numRes=SrvEnt(Rqst("Manhours_Piping"))
        Else
                numRes=Rqst("NumRes")
        End If
End If
End If
If SrvEnt("Auxiliary")=5 Then ''' to capture Manhours_Cable_Tray
If Rqst.Attr("NumRes").Calculation=CFC_Formula Then
        If Not IsNumeric(Rqst("NumRes")) Then
                numRes=SrvEnt(Rqst("Manhours_Cable_Tray"))
        Else
                numRes=Rqst("NumRes")
        End If
End If
End If
If SrvEnt("Auxiliary")=6 Then ''' to capture Manhours_EHT
If Rqst.Attr("NumRes").Calculation=CFC_Formula Then
        If Not IsNumeric(Rqst("NumRes")) Then
                numRes=SrvEnt(Rqst("Manhours_EHT"))
        Else
```

```
                numbRes=Rqst("NumRes")
        End If
End If
End If
If SrvEnt("Auxiliary")=7 Then ''' to capture Manhours_Insulation
If Rqst.Attr("NumRes").Calculation=CFC_Formula Then
        If Not IsNumeric(Rqst("NumRes")) Then
                numRes=SrvEnt(Rqst("Manhours_Insulation"))
        Else
                numRes=Rqst("NumRes")
        End If
End If
End If
If SrvEnt("Auxiliary")=8 Then ''' to capture Manhours_Fireproof
If Rqst.Attr("NumRes").Calculation=CFC_Formula Then
        If Not IsNumeric(Rqst("NumRes")) Then
                numRes=SrvEnt(Rqst("Manhours_Fireproof"))
        Else
                numRes=Rqst("NumRes")
        End If
End If
End If

'Then change ["Rqst("NumRes")] in the statements marked with >> to
[numRes]
'========================
'
If ResElmnt("Current")>=numRes Then
        If SrvEnt("Auxiliary")= 0 Then
                If (ResElmnt("EndBackModule")+SrvEnt("WaitingDays))>=
                (SrvEnt("Duration_Override_Struct")+
          SrvEnt("Duration_Override_Piping")+
          SrvEnt("Duration_Override_Cable_Tray")+
          SrvEnt("Duration_Override_EHT")+
          SrvEnt("Duration_Override_Insulation")+
          SrvEnt("Duration_Override_Fireproo)+SimTime) Then
          If (ResElmnt("FinishCurrentModule")+
          SrvEnt("WaitingDays))>=
               (SrvEnt("Duration_Override_Struct")+
               SrvEnt("Duration_Override_Piping")+
               SrvEnt("Duration_Override_Cable_Tray")+
               SrvEnt("Duration_Override_EHT")+
               SrvEnt("Duration_Override_Insulation")+
               SrvEnt("Duration_Override_Fireproof")+SimTime) Then
                    ResAvailable=True
               '**** Record the requested resource for automatic release
                    Set SrvEnt("CEM_Common_RqstdRes")=ResElmnt
               SrvEnt("CEM_Common_NumRqstdRes")=numRes
               ResElmnt("Current")=ResElmnt("Current")-numRes
               SrvEnt("Assembly")=numRes
               SrvEnt("Space")=ResElmnt("Current")
               SrvEnt("Total")=SrvEnt("Assembly")+SrvEnt("Space")
               SrvEnt("Bay")=ResElmnt("ResName")

                    If ResElmnt("EndBackModule")=1000 Then
ResElmnt("EndBackModule")=(SrvEnt("Duration_Override_Struct")+
SrvEnt("Duration_Override_Piping")+
SrvEnt("Duration_Override_Cable_Tray")+
SrvEnt("Duration_Override_EHT")+SrvEnt("Duration_Override_Insulation")
+SrvEnt("Duration_Override_Fireproof")+SimTime)
                         ResElmnt("ModuleCounter")=0
                    End If
```

```
ResElmnt("ModuleCounter")=ResElmnt("ModuleCounter")+1
SrvEnt("No_of_Modules_behind")=ResElmnt("ModuleCounter")-1
                        If CInt(ResElmnt("Current"))>0 Then
                                SrvEnt("Space_in_front")="Yes"
                        Else
                                If CInt(ResElmnt("Current"))=0 Then
                                        SrvEnt("Space_in_front")="No"
                                End If
                        End If
If
ResElmnt("FinishCurrentModule")>(SrvEnt("Duration_Override_Struct")+
SrvEnt("Duration_Override_Piping")+
SrvEnt("Duration_Override_Cable_Tray")+
SrvEnt("Duration_Override_EHT")+
SrvEnt("Duration_Override_Insulation")+
SrvEnt("Duration_Override_Fireproof")+SimTime) Then
ResElmnt("FinishCurrentModule")=(SrvEnt("Duration_Override_Struct")+
SrvEnt("Duration_Override_Piping")+
SrvEnt("Duration_Override_Cable_Tray")+
SrvEnt("Duration_Override_EHT")+
SrvEnt("Duration_Override_Insulation")+
SrvEnt("Duration_Override_Fireproof")+SimTime)
End If
If ResElmnt("Total")<>0 Then ResElmnt.stat("Utilization").Collect 100 * (1-
(ResElmnt("Current")/ResElmnt("Total")))
'*** Then remove the entity from the file and schedule a Granted Request event in its
original capture element
                .Remove SrvEnt
                        RqstElmnt.ScheduleEvent SrvEnt,"RqstGrntd",0
Tracer.Trace ">>>>>>>Rqst. for ONE of the res. is GRANTED for Entity #"&
SrvEnt.Id,"res","File","Granted"
```

```
                    Exit For
          Else
               ResAvailable=False
               End If
          Else
               ResAvailable=False
          End If
          Else
```
If SrvEnt("Auxiliary")= 1 Or SrvEnt("Auxiliary")> 2 Then ''' to grant resources in other captures besides capturing assembly space
```
                    ResAvailable=True
            .....................................................................................................
            ............................'**** Record the requested resource for automatic release
                         Set SrvEnt("CEM_Common_RqstdRes")=ResElmnt
```
SrvEnt("CEM_Common_NumRqstdRes")=numRes

ResElmnt("Current")=ResElmnt("Current")-numRes
```
If ResElmnt("Total")<>0 Then ResElmnt.stat("Utilization").Collect 100 * (1-
(ResElmnt("Current")/ResElmnt("Total")))
'*** Then remove the entity from the file and schedule a Granted Request event in its
original capture element
                         .Remove SrvEnt
                         RqstElmnt.ScheduleEvent SrvEnt,"RqstGrntd",0
Tracer.Trace ">>>>>>>Rqst. for ONE of the res. is GRANTED for Entity #"&
SrvEnt.Id,"res","File","Granted"
                    Exit For
Else
          ResAvailable=False
End If
                    End If
               Else
```
If SrvEnt("Auxiliary")=2 Then ''' to capture assembly space for modules that have already started their fabrication process

```
If Rqst.Attr("NumRes").Calculation=CFC_Formula Then
            If Not IsNumeric(Rqst("NumRes")) Then
                  numRes=SrvEnt(Rqst("NumRes"))
            Else
                  numRes=SrvEnt("UnitsRequested")
            End If
Else
            numRes=Rqst("NumRes")
End If
                  ResAvailable=True
'**** Record the requested resource for automatic release
      Set SrvEnt("CEM_Common_RqstdRes")=ResElmnt
      SrvEnt("CEM_Common_NumRqstdRes")=numRes
      ResElmnt("Current")=ResElmnt("Current")-numRes
      SrvEnt("Assembly")=numRes
  SrvEnt("Space")=ResElmnt("Current")
  SrvEnt("Total")=SrvEnt("Assembly")+SrvEnt("Space")
  SrvEnt("Bay")=ResElmnt("ResName")

If SrvEnt("Task")=1 Then
ResElmnt("EndBackModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Struct")+S
rvEnt("Duration_Override_Piping")+SrvEnt("Duration_Override_Cable_Tray
")+SrvEnt("Duration_Override_EHT")+SrvEnt("Duration_Override_Insulatio
n")+SrvEnt("Duration_Override_Fireproof")+SimTime)
ResElmnt("FinishCurrentModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Struct")+S
rvEnt("Duration_Override_Piping")+SrvEnt("Duration_Override_Cable_Tray
")+SrvEnt("Duration_Override_EHT")+SrvEnt("Duration_Override_Insulatio
n")+SrvEnt("Duration_Override_Fireproof")+SimTime)
End If
```

```
If SrvEnt("Task")=2 Then
ResElmnt("EndBackModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Piping")+S
rvEnt("Duration_Override_Cable_Tray")+SrvEnt("Duration_Override_EHT")
+SrvEnt("Duration_Override_Insulation")+SrvEnt("Duration_Override_Firep
roof")+SimTime)
ResElmnt("FinishCurrentModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Piping")+S
rvEnt("Duration_Override_Cable_Tray")+SrvEnt("Duration_Override_EHT")
+SrvEnt("Duration_Override_Insulation")+SrvEnt("Duration_Override_Firep
roof")+SimTime)
End If
If SrvEnt("Task")=3 Then
ResElmnt("EndBackModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Cable_Tra
y")+SrvEnt("Duration_Override_EHT")+SrvEnt("Duration_Override_Insulati
on")+SrvEnt("Duration_Override_Fireproof")+SimTime)
ResElmnt("FinishCurrentModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Cable_Tra
y")+SrvEnt("Duration_Override_EHT")+SrvEnt("Duration_Override_Insulati
on")+SrvEnt("Duration_Override_Fireproof")+SimTime)
End If
If SrvEnt("Task")=4 Then
ResElmnt("EndBackModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_EHT")+Sr
vEnt("Duration_Override_Insulation")+SrvEnt("Duration_Override_Fireproof
")+SimTime)
ResElmnt("FinishCurrentModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_EHT")+Sr
vEnt("Duration_Override_Insulation")+SrvEnt("Duration_Override_Fireproof
")+SimTime)
```

```
End If
If SrvEnt("Task")=5 Then
ResElmnt("EndBackModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Insulation"
)+SrvEnt("Duration_Override_Fireproof")+SimTime)
ResElmnt("FinishCurrentModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Insulation"
)+SrvEnt("Duration_Override_Fireproof")+SimTime)
End If
If SrvEnt("Task")=6 Then
ResElmnt("EndBackModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Fireproof")
+SimTime)
ResElmnt("FinishCurrentModule")=(SrvEnt("NoOfDaysSinceStart")-
SrvEnt("NoOfDaysAlreadyOnTask")+SrvEnt("Duration_Override_Fireproof")
+SimTime)
End If
          If              ResElmnt("Total")<>0              Then
ResElmnt.stat("Utilization").Collect        100        *        (1-
(ResElmnt("Current")/ResElmnt("Total")))
'*** Then remove the entity from the file and schedule a Granted Request event
in its original capture element
                                  .Remove SrvEnt
                        RqstElmnt.ScheduleEvent SrvEnt,"RqstGrntd",0
Tracer.Trace ">>>>>>>Rqst. for ONE of the res. is GRANTED for Entity #"&
SrvEnt.Id,"res","File","Granted"
                        Exit For
Else
ResAvailable=False
End If
End If
End If
Else
                    ResAvailable=False
             End If
                    Next
                           If Not ResAvailable Then
Tracer.Trace "Reqst ANY denied and moving to the next ent" ,"res","File","Denied"
                                  .MoveNext
                           End If
      End Select
                           Wend
      End With
             Tracer.Trace "^^^^^^^^ File Check Ended ^^^^^^^^","res","File"
End Select
End Sub
```

Release Resources Element:

```
Public Sub Release_OnSimulationProcessEvent(ob As
CFCSim_ModelingElementInstance, MyEvent As String, Entity As CFCSim_Entity)
             Dim RelRes As CFCSim_ModelingElementInstance
             Dim ResElmnt As CFCSim_ModelingElementInstance
             Dim File As CFCSim_ModelingElementInstance
             Dim ResID
Select Case MyEvent
Case "AutoRelease"
If Entity("CEM_Common_RqstElmnt")("RqstType")="ALL" Then
      For Each RelRes In Entity("CEM_Common_RqstElmnt").ChildElements

If RelRes("ResName")<>"**Linked to Entity Attribute**" Then    '*** Case not
linked
```

```
                        Set ResElmnt=RelRes("ResOb").Reference
Else                    '**** Case linked
        For Each ResElmnt In Elements 'ob.Parent.ChildElements
                If  ResElmnt.ElementType="Resource" Then
                If ResElmnt("ResName")=Entity(RelRes("EntAttr")) Then Exit For
                End If
        Next
End If
        ResElmnt("Current")=ResElmnt("Current")+RelRes("NumRes")
'If the bay is empty again, the finishing date available should be large again
If ob.CurrentEntity("Auxiliary")=9 Then
        If ResElmnt("Current")=ResElmnt("Total") Then
                        ResElmnt("EndBackModule")=1000
                        ResElmnt("FinishCurrentModule")=1000
                        ResElmnt("ModuleCounter")=0

        End If
End If
If ResElmnt("Total")<>0 Then ResElmnt.stat("Utilization").Collect 100 * (1-
(ResElmnt("Current")/ResElmnt("Total")))
        Next
Else
Entity("CEM_Common_RqstdRes")("Current")=Entity("CEM_Common_RqstdRes")
("Current")+Entity("CEM_Common_NumRqstdRes")
'If the bay is empty again, the finishing date available should be large again
If ob.CurrentEntity("Auxiliary")=9 Then
        If Entity("CEM_Common_RqstdRes")("Current") =
        Entity("CEM_Common_RqstdRes")("Total") Then
        Entity("CEM_Common_RqstdRes")("EndBackModule")=1000
        Entity("CEM_Common_RqstdRes")("FinishCurrentModule")=1000
        Entity("CEM_Common_RqstdRes")("ModuleCounter")=0
```

```
          End If
End If
Entity("CEM_Common_RqstdRes").stat("Utilization").Collect 100 * (1-
(Entity("CEM_Common_RqstdRes")("Current")/Entity("CEM_Common_RqstdRes")
("Total")))
End If
          Case "Release"
'**** Increase the number of resources by the number defined in each single-Res-
Release
For Each RelRes In ob.ChildElements
'Make the current entity of the single releases same as parent
'RelRes.OnSimulationTransferIn Entity,Nothing,Nothing
If RelRes("ResName")<>"**Linked to Entity Attribute**" Then    '*** Case not
linked
                    Set ResElmnt=RelRes("ResOb").Reference
Else                          '**** Case linked
For Each ResElmnt In Elements 'ob.Parent.ChildElements
          If  ResElmnt.ElementType="Resource" Then
                    If ResElmnt("ResName")=Entity(RelRes("EntAttr")) Then Exit For
          End If
Next
End If
                    ResElmnt("Current")=ResElmnt("Current")+RelRes("NumRes")
'If the bay is empty again, the finishing date available should be large again
If ob.CurrentEntity("Auxiliary")=9 Then
          If ResElmnt("Current")=ResElmnt("Total") Then
                    ResElmnt("EndBackModule")=1000
                    ResElmnt("FinishCurrentModule")=1000
                    ResElmnt("ModuleCounter")=0
          End If
End If
```

```
If ResElmnt("Total")<>0 Then ResElmnt.stat("Utilization").Collect 100 * (1-
(ResElmnt("Current")/ResElmnt("Total")))
Next
End Select
'**** Schedule a check event for all the files in the model
          For Each File In Elements 'ob.Parent.ChildElements
                    If File.ElementType = "Waiting_File" Then
                              File.ScheduleEvent ob.AddEntity,"Check",0
                    End If
          Next
          Tracer.Trace "Entity "& Entity.Id &"Released the resources","res","Release"
          ob.TransferOut Entity

End Sub
```