# Scheduling Program Based on the Theory of Constraints

*Author: J.L. Malherbe*

Assignment presented in partial fulfilment of the requirements for the degree of Master in Industrial Engineering at the University of Stellenbosch

*Promoter: Mr. K. Bartel*

*March 2003*

# Declaration

I, the undersigned, hereby declare that the work contained in this assignment is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: Date:

## Projek Opsomming

Die doel van hierdie tesis is om die grondslag te skep vir die ontwerp en ontwikkeling van 'n sagteware pakket wat Goldratt se Doel Sisteem Algoritme, gebasseer op die 'Theory of Constraints', implementeer. Dit sluit die gedetaileerde beskrywing van die Doel Sisteem Algoritme in en 'n gedeeltelike implementasie van die algortime, deur gebruik te maak van 'n Microsoft Access databasis as databasis bestuur sisteem en Microsoft Visual C++ as 'n programerings taal. Die hoof klem is gelê op die ontwikkeling van die skedulerings algortime en die implementasie van die strukture wat deel van die kern uitmaak van die algoritme.

**Die hoof rede vir die ontwikkeling van so 'n pakket is sodat dit 'n produksie bestuurder van 'n klein to medium grootte vervaardigings besigheid sal help om 'n skedule vir produksie the genereer wat die vloer se deurset sal verhoog, terwyl dit voorraad en operasionele kostes sal verlaag. Met ander woorde dit sal die besigheid help om meer geld te maak huidiglik en in die toekoms.**

Met betrekking tot die tesis en die algehele projek doel is die volgende bereik:

1. Die hele projek is nagevors, uit een gesit en verduidelik.
2. Die hele program struktuur is gedefinieer en opgebreek in twee aparte modules; nl. die 'Data Mining and Conversion Module' en die 'TOC Scheduling Algorithm'.
3. Die databasis wat al die nodige MRP inligting bevat wat benodig word vir skedulering is ontwerp en geimplementeer deur gebruil te maak van 'n MS Access databasis met 'n ODBC konneksie. Daar is van 'n ODBC konneksie gebruik gemaak sodat as die nodig is, daar sonder enige moeite na ander databasis bestuurs sisteme oorgeskakel kan word.
4. Die 'TOC Scheduling Algorithm' is onwikkel en die volgende is geimplementeer.
   - A basisse gebruikers vlak is ontwikkel sodat al die nodig invoer data in die program ingevoer kan word.
   - 'n Geskakelde lys is ontwikkel en gebruik as die data struktuur om al die skedulerings informasie in geheue te stoor.
   - Die Doel Sisteem algorimte is in sy geheel verduidelik en gedokumenteer.

- Die Doel Sisteem algoritme is geimplementeer tot op die punt waar dit die primêre bottelnek skeduleer.
- Die pseudo kode vir die deel van die GS algoritme wat nie geimplementeer is nie is uitgelê in ingesluit as deel van die verslag.

Verdere ontwikkeling word nog benodig en 'n  beter gebruikers vlak moet nog geskep word om die projek te finaal af te handel, maar na dit gedoen is sal daar 'n TOC skedulering pakket bestaan wat heeltemal uniek is tot Suid-Afrika en 'n groot mark potensiaal sal hê.

## Project Summary

The goal of this thesis is to provide a stepping-stone for the design and development of a software package that implements the Goal System Algorithm, based on the Theory of Constraints (TOC). This includes the complete description and explanation of the Goal System Algorithm (GS), as well as the partial implementation of this algorithm using Microsoft Access as a Database Management System (DBMS) and Microsoft Visual C++ as programming language. The main development effort was put into the development of a scheduling algorithm and the implementation of a data structure that lies at the core of this algorithm.

**The reason for the development of such a package is that it will aid a production manager, working in a small to medium size job-shop, in generating a schedule for production that will increase throughput, while simultaneously reducing both inventory and operating expense thereby generating profits and cash flow.**

With regard to this thesis and the overall project goal the following have been achieved.

1. The complete project has been researched, scoped and each step has been explained.
2. The complete program structure has been defined and broken into two separate modules; the *Data Mining and Conversion Module* and the *TOC Scheduling Algorithm.*
3. The database containing all the MRP data necessary for scheduling has been designed and implemented using a MS Access database with an ODBC connection. An ODBC connection to the database was used so that a smooth transmission to other database management systems can be made.
4. The *TOC Scheduling Algorithm* has been developed and the following have been implemented:
   - A basic user interface has been created for the insertion of all the user input and to display the constraint schedule.
   - A data structure called a linked list has been developed and used to store the scheduling data in memory.
   - The complete GS algorithm had been researched and explained.

- The GS algorithm has been and implemented and tested up to the point where it schedules the constraint.
- The pseudo code for the part of the GS algorithm that was not implemented has been documented and included in this report.

More development needs to be done and a proper Graphical User Interface must also be created to complete this project, but after completion a TOC software package will exist that is completely unique in South Africa and the market potential for this package will be considerable.

# Table of Contents

# List of Illustrations

# Acknowledgements

I would like to thank Mr. Konrad Bartel for his guidance, input and patience through the whole duration of this project.

# Chapter 1

## *1. Introduction*

### 1.1 Background

The scheduling of all the individual jobs to produce a product and subsequently fill orders on time in a manufacturing environment is an interesting and complicated problem. The biggest factor influencing the throughput of a production line is the occurrence of bottlenecks on the line and the build up of work in process on the job floor because of this. The challenge is to schedule the various parallel workflows in the manufacturing system in such a way to increase the overall throughput and to ensure that orders are completed on time. This problem and a solution for this problem, in the form of the Theory of Constraints (TOC), was addressed by E. Goldratt in his book 'The Goal' that explains this philosophy and the benefits of the implementation of this philosophy. This philosophy is centred on the approach of identifying the resource that forms the bottleneck in the production system and creating a schedule for that bottleneck, and then scheduling the rest of the system around the schedule for the bottleneck. The reason for this is because the overall throughput of the system can only be as much as the throughput of the bottleneck. The challenge is to develop a software system that implements this philosophy as a solution.

**The reason for the development of such a package is that it will aid a production manager in generating a schedule for production that will increase throughput, while simultaneously reducing both inventory and operating expense thereby generating profits and cash flow.**

There are already a number of scheduling packages available on the market, but they are extremely expensive and are totally unaffordable to most small manufacturing companies in South Africa. Small manufacturing companies will generally not have the resources and systems available to provide the off the shelf packages with the data that it will need to schedule accurately.

The reason for this is that the implementation of TOC into a software package is a complex problem with a large amount of possible input variables required to make the package as generic as possible, so it that can be used by all types of manufacturing industries. Also none of these packages have been developed in South Africa. Because of these factors the packages currently available on the market are large and very expensive, and extensive training will also have to be provided the users. The benefit of these packages, however, is that they take every possible situation into account and basically do all the thinking, making it possible to schedule within a very high level of accuracy.

## 1.2 Goal of the Thesis

The goal of this thesis is to provide:

- A stepping-stone for the design and development of a software package that implements the Goal System Algorithm, based on the Theory of Constraints (TOC)
- A complete description and explanation of the Goal System Algorithm (GS).
- A partial implementation of this algorithm using Microsoft Access as a Database Management System (DBMS) and Microsoft Visual C++ as programming language.

The main development effort was put into the development of a scheduling algorithm and the implementation of a data structure that lies at the core of this algorithm.

## 1.3 The Procedure Followed

**This package was developed to be affordable to the small and medium size companies, and still be able to schedule to an acceptable level accuracy**. To be able to achieve this goal the complexity of the program must be reduced in such a way as to not influence the scheduling accuracy significantly. This is be made possible by making use of an 80/20 principle i.e., the scheduling approach would be to schedule the small portion of jobs (the 20%) that account for the bulk of the time in the system (the 80%) and allow for sufficient slack capacity to accommodate the work not scheduled in detail. So instead of developing sophisticated software to do the exploitation and subordination procedures this package was developed in such a way that the inputs from the users and their experience and knowledge of the shop floor can be used to drive the exploitation and subordination processes. It is then possible to make use of a what-if analysis approach to the scheduling. This approach can be effectively combined with a worker empowerment philosophy.

This is not a solution that will replace the workforce or take away their ability to think or make decisions. The job shop worker will use this package as a tool so that they can use their own experience, gained by working on the particular production line, to make decisions and manage and schedule the production line. Through their input it will also be possible test different what-if scenarios and schedules to help provide an optimal solution to their scheduling problem.

This TOC Scheduler implements and explains the TOC principles of:

- **Identifying the constraint.** A constraint is loosely defined as something that restricts system throughput.
- **Exploiting the constraint.** Involves always keeping the constraint producing and prioritising what it produces to maximise overall system throughput.
- **Subordinating the remaining resources.** Determines what the non-constraint resources should be doing
- **Elevating the constraint.** Increasing the capacity of the constraint after the previous step have been completed
- **Repeating the process.**

The steps discussed above are in short the steps that were taken to implement the scheduling algorithm. This was, however, not the start of the process. This thesis is a continuation of the work done by Mr. T Boezaart as part of his graduate thesis in Industrial Engineering. His goal was to improve competitiveness by implementing a short term scheduling system based on the Theory of Constraints for a manufacturing company called Rovic Inc, who is a manufacturer of axles. Rovic Inc. provided all the necessary manufacturing data needed for the scheduling of their shop floor. The system was developed by making use of MS Visual Basic for Applications using MS Excel as a platform and a MS Access database to store the relevant data. The original plan for this thesis was to translate the work done by Mr. Boezaart and use the algorithms developed as part of this thesis and implement it in a format that is commercially viable. After careful analysis of the work done and after significant additional research in the area of TOC it was found that the problem, however, was considerably more complex than anticipated at first. This was due to the fact that the solution implemented and the scheduling algorithm used, was not sufficient for the purpose of creating a generic TOC scheduler. It was also found that the data structures used could be improved upon.

This was highlighted when the work done by Mr. L. Louw in his thesis presented for his postgraduate degree in Industrial Engineering was researched. Mr. Louw did considerable work in the field of TOC and developed a TOC and implemented it using MS Visual Basic 4. His work provided additional insight into the problem of developing a solution for the scheduling algorithm, but the thought was that there were still room for improvement. The areas that were considered for improvements were the data structure used and the way it was represented in computer memory and the scheduling algorithm itself. A solution was found on how organise all the scheduling information into a data structure called 'The Net' and explained by Robert E. Stein in his book, 'Re-Engineering the Manufacturing System. *The Net* was then developed and implemented and formed the basis on which the scheduling algorithm was built on.

A solution for the development of the algorithm was found in an article called 'An Exposition of Multiple Constraint Scheduling as Implemented in the Goal System' by J.V Simons and W.P Simpson. This article provided a short description of the *Goal System Algorithm* where the problem of a constraint feeding itself was addressed. This combined with *The Net,* provided the foundation for the development of a generic scheduling algorithm for the scheduling of a production line. This is also what differentiates this algorithm from the ones already developed, because all the work done up to this point did not include the possibility that a manufacturing line might have a constraint that feeds itself and the consequences that this will have on the accuracy of the scheduled generated.

## 1.4 Scope and Challenges

This process, beginning with the initial research on the TOC and Mr. Boezaart's thesis up to the basis for solution for the algorithm as explained above was a learning curve that took 5 months. The implementation of the algorithm was also a very challenging problem, because the key to the success of such a computationally complex and resource intensive package is operating speed. C++ was used for the development of this package, because although it is a more complicated programming language than for example Visual Basic, the major benefit is that it is possible to generate code that is very fast and very efficient. Dynamic linked lists were used to represent *The Net* in memory.

Due to the complexity of the problem and the amount of time it took to find a feasible solution, it was impossible to develop a complete package with the necessary user interfaces that would transform it into a commercial product. This thesis does however complete the entire groundwork for the development of a commercial. This work done is:

- A feasible program and data structure have been developed.
- An algorithm has been implemented that can be built upon and re-used.
- All the development work that must still be completed to create a commercial product have been planned and documented.

## 1.5 Report Structure

Chapter 2 of this report will introduce and explain the program structure of the complete package. Chapter 3 will provide an explanation of the software and hardware requirements and the testing requirements. Chapter 4 specifies the input and output domain and defines *The Net*. Chapters 5, 6 and 7 explain the identification, exploitation and subordination of the constraints with regard to the GS algorithm and how it was implemented. The report is concluded in Chapter 8.

# Chapter 2

## 2. *Requirements Definition for the TOC Scheduler*

### *2.1 The Model*

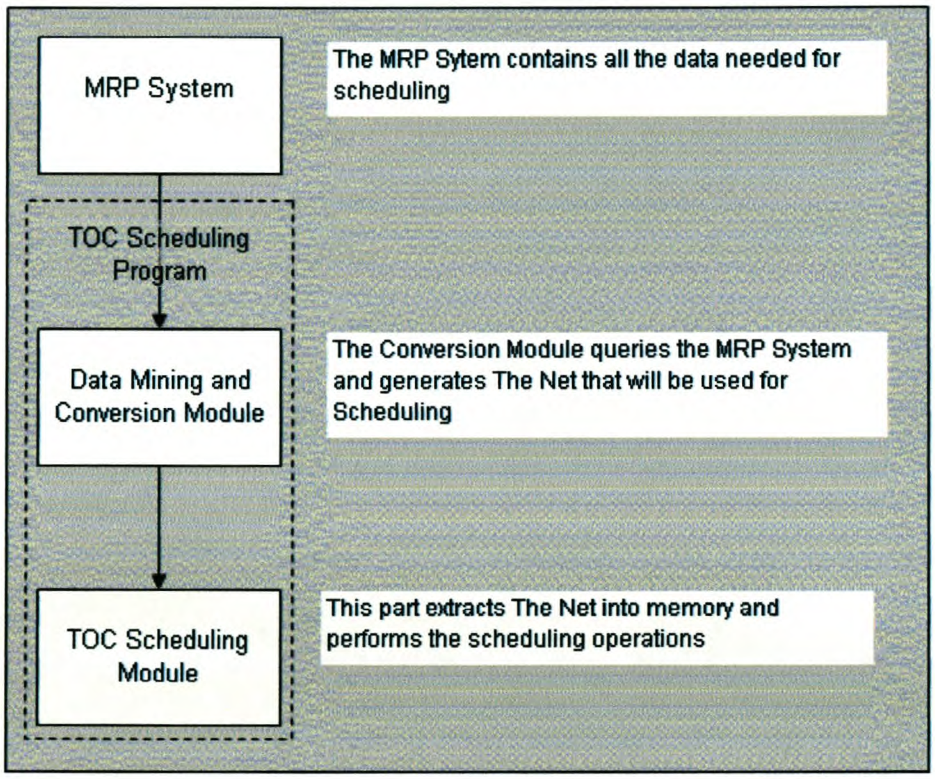

MRP System
The MRP Sytem contains all the data needed for scheduling
TOC Scheduling Program
Data Mining and Conversion Module
The Conversion Module queries the MRP System and generates The Net that will be used for Scheduling
TOC Scheduling Module
This part extracts The Net into memory and performs the scheduling operations

**Figure 2.1:** The Program Structure

As can be seen from the program structure shown in *Figure 2.1,* the complete scheduling program consists of two main modules, namely:

1. The Data Mining and Conversion Module (DMCM)
2. The TOC Scheduling Module (TOCSM)

The function of these modules and their integration with the MRP system are explained in the following section:

### 2.1.1 MRP System

All the information that will be needed for scheduling will be available in the MRP system of the particular company. The function of the Data Mining and Conversion Module is to identify mine and convert the necessary scheduling information from the MRP system into the correct format so that the Scheduling Module can interpret it.

### 2.1.2 The Data Mining and Conversion Module:

This module will query the MRP system and insert the scheduling information into a structure called *The Net. The Net* is a re-creation of the way in which products flow through the factory and is a summary of all the information needed for the scheduling algorithm combined into one database table. All the resource relationships are defined in *The Net* and form a critical part of the whole scheduling program and is the starting point for all the scheduling operations. It is the most critical structure in the scheduling program and will be explained and discussed in detail in *Section 4.2.1.*

The DMCM module will typically be a database program compiled in MS SQL or MS Access, depending on the size of the company. This will increase the stability of the program considerably, because it will ensure that no changes in the code will need to be made to the scheduling program if the structure of the MRP system changes. Only the queries in the conversion program will need to be updated. This will make the program considerably more flexible and because of this more compatible in different environments.

Within the scope of the thesis, only the necessary information and *The Net* structure was inserted into a database in the correct format, for interpretation by the Scheduling Module. For a commercial system a complete custom Conversion Module will have to be developed for the specific MRP system. Depending on the complexity of the MRP system, this should not be a complex exercise to complete.

### 2.1.3 The TOC Scheduling Module

This module extracts the needed scheduling information form the Conversion Module and executes the scheduling algorithm. The scheduling algorithm is based on Goldratt's Goal System Drum Buffer Rope Scheduling Algorithm and is derived directly from the first three focussing steps of identifying the constraint, exploiting the constraint and subordinating to the constraint

A schedule called a *drum* is built for each constraint to ensure the constraint will be used to its fullest effect. Since non-constraint resources have, by definition, more than sufficient capacity to keep pace with the constraint schedule, it is considered unnecessary to generate detailed schedules for their operations. Instead, a roughly estimated amount of time, referred to as a *buffer*, is built into the schedule to allow sufficient lead-time for non-constraint operations to occur. The term *rope* refers to the lead-time offset, based on the buffer size, used to generate upstream schedules.

This algorithm will also be able to schedule situations where a there is more than one constraint operation or where a constraint feeds itself. This was done by making use of rods to generate the necessary time lag between operations. The GS algorithm will be explained in *Chapters 5, 6 and 7.*

The structure on which the whole TOC Scheduling program is based is shown in *Figure 2.2.*

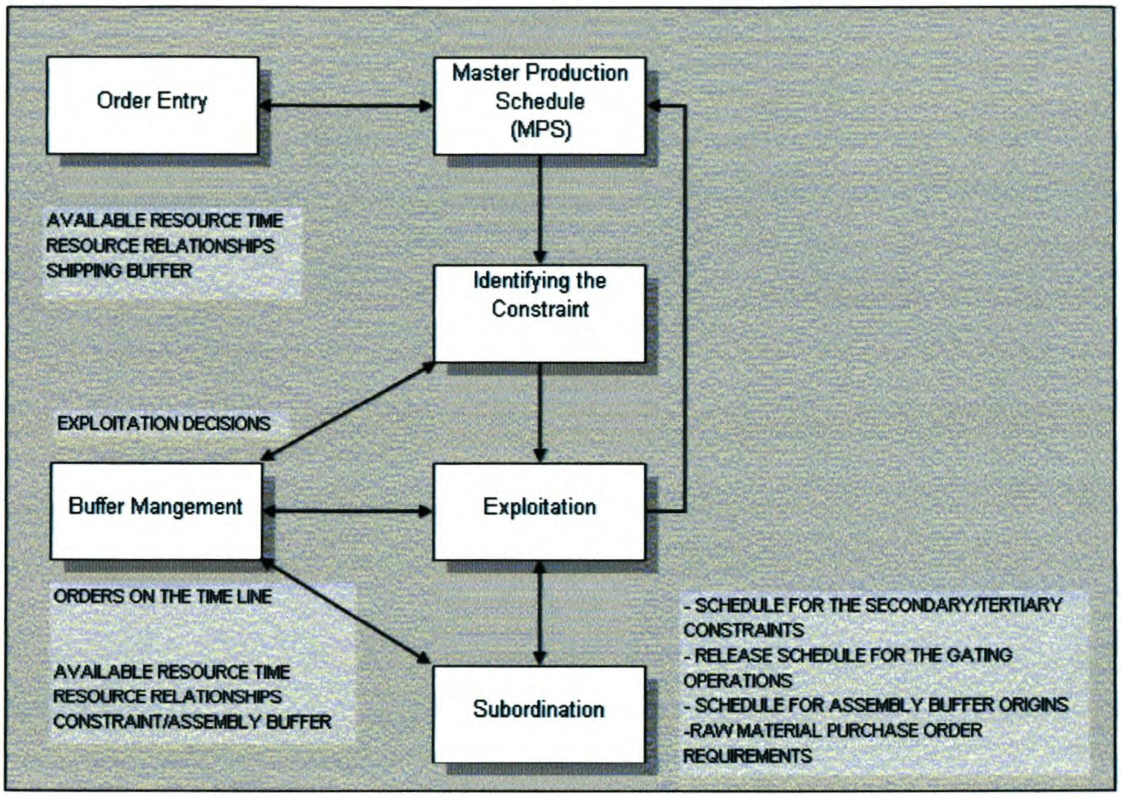

Order Entry
Master Production Schedule (MPS)
AVAILABLE RESOURCE TIME
RESOURCE RELATIONSHIPS
SHIPPING BUFFER
Identifying the Constraint
EXPLOITATION DECISIONS
Buffer Mangement
Exploitation
ORDERS ON THE TIME LINE
AVAILABLE RESOURCE TIME
RESOURCE RELATIONSHIPS
CONSTRAINT/ASSEMBLY BUFFER
Subordination
- SCHEDULE FOR THE SECONDARY/TERTIARY CONSTRAINTS
- RELEASE SCHEDULE FOR THE GATING OPERATIONS
- SCHEDULE FOR ASSEMBLY BUFFER ORIGINS
-RAW MATERIAL PURCHASE ORDER REQUIREMENTS

**Figure 2.2:** The Structure of a general TOC Scheduling program

# Chapter 3

## *3.1 The Software and Hardware Characteristics*

The resources needed to run this program successfully is subject to the size of *The Net* stored in the *Data Mining and Conversion Module.* Because the whole net structure is read into the computer memory and all calculations are done from memory, the speed of the computer processor determines the speed at which the calculations are preformed. The amount of free Random Access Memory (RAM) determines the size of *The Net* that the scheduling algorithm can handle.

For a small job shop the following resource requirements should be sufficient:

1. Intel Pentium, 333 MHZ (minimum requirement)
2. 64 MB RAM (minimum requirement)
3. Windows NT/95/98/2000
4. Microsoft Access 2000

For the implementation of the program as part of a thesis, it was decided to use MS Access 2000 as a database managements system instead of SQL Server 2000 for a number of reasons. They are:

1. MS Access is easily attainable and more freely available.
2. MS Access is more user-friendly and easier to use.
3. There will be no compatibility problems when the software is executed on different computers.
4. It will run faster on older, smaller stand-alone computers.

The software was developed making use of an ODBC connection to the database, because of this it will be very easy to convert to SQL Server or any other DBMS.

The computer language chosen to implement the program was Microsoft Visual C++ for the following reasons:

1. C++ is a powerful language that makes it possible to create fast and efficient programs.
2. It possible to make use of data structures like linked lists and pointers that are not available in none object orientated languages like Visual Basic.
3. To create a program that is commercially viable it must be written in a language that is an industry standard and C++ is such a language.

## 3.2 Software Quality Characteristics

The following software quality characteristics (Kolarik , 1995: 100 – 1004) were considered in the development of the program:

1. *This package must be easy to use and easy to understand.* Most people in a job shop environment are not highly computer literate; therefore it was important to develop a program that is easy to understand and easy to use.
2. *The program must be robust.* An inexperienced user must not be able to 'break' or 'crash' the program easily. The following types of possible human errors must be considered:
   - **Errors of omission.** Errors of omission are committed when not all the needed user input has been entered into the system.
   - **Errors of commission.** This type of error will occur when information is entered incorrectly. This can be correct information in the wrong place or incorrect information in the correct place.
   - **Extraneous acts.** This is when an action has been performed that should not have been performed.
   - **Sequential errors.** A sequential error occurs when a task is performed out of sequence.

   At this stage of the development of the program it is impossible for the user to break or crash the program.
3. *The program must be correct.* This is the extent to which the program satisfies its specifications and the precision to which it performs its numerical calculations.

4. *There must be looked at the program's interoperability.* This is a measure of the ease with which one subsystem can be coupled with another.
5. *The program must be flexible*: This is a measure of the ease with which the program can perform (or be modified to perform) functions beyond the scope of its original requirements.
6. *The program must be efficient*: Efficiency is a measure of the use of high-performance algorithms and conservative use of resources to minimise the cost of operations.
7. *The program must be valid*: This is the ability of a program to provide performance, functions, and interfaces that are sufficient for effective application in the intended user environment.

All of these quality issues were addressed in the development of the program.

## 3.3 Test Requirements

History has shown that up to 50% of the cost involved in developing a software system can be attributed to integration and testing (Moriguchi, 1997, 651). This would suggest that significant savings in time and money could be realised if more attention is paid to the analyses and design stage of the development process. In other words quality must be built in from the very beginning and through every step of the software development process.

This will minimise the amount of errors in the program. The program must, however, still be tested. Test cases must be developed that will test the program within its specifications and on the boundaries of its specifications.

The TOC Scheduling program was tested by testing each completed module exhaustively using a wide range of test cases developed by the programmer. Complete integration testing was also done.

# Chapter 4

## 4.1 Specification of the Input/Output Domain

The first step in the development and the design of this program was to develop a level 0 Data Flow Diagram (DFD) to identify the input data to the program, where it comes from and then output that it will provide. The DFD is shown in *Figure 4.1.* At this stage of the development the focus was on WHAT needed to be done and not on HOW to do it. In other words the program was seen as a 'black box' and no thought was given at this stage to the internal workings of a particular step.

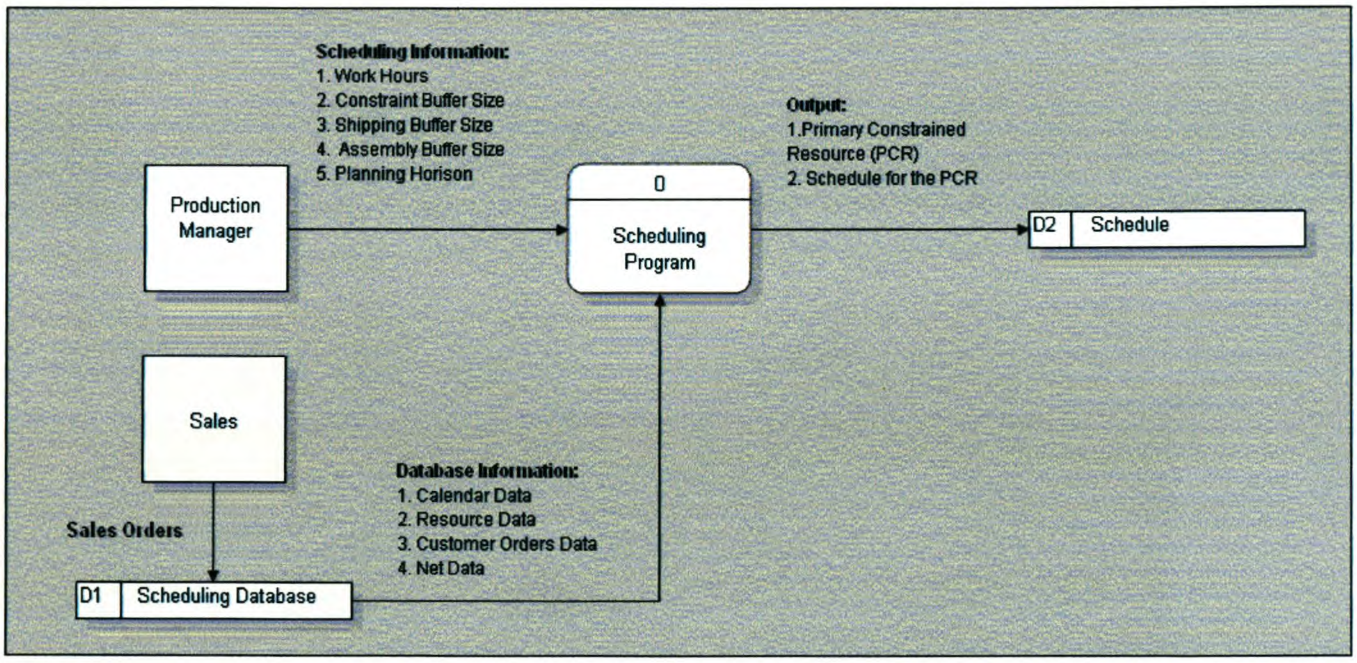


**Figure 4.1:** Level 0 Data Flow Diagram

Most of the information needed for this system will be retrieved from tables in the MS Access database in the form of *The Net*. These tables have the following fields:

1. Calendar data that includes:
    - **The Date.** Normal calendar date.
    - **General work days.** Specifies whether a calendar date is a work day or not.
2. Resource Data that inlcudes:
    - **Resource ID.** Specifies the Unique ID for each resource.
    - **Resource Quantity.** Specifies the quantity of each resource that is available.
3. Customer Orders that includes:
    - **Order ID.** Specifies the unique order ID.
    - **Product ID.** Specifies the product ID of the product that must be manufactured.
    - **Due Date.** Gives the order due date.
    - **Orders Placed.** Boolean value that specifies if a specific product has already occurred in the order list. This field is include so that for the identification of the constraint the setup for the same operations is included only once.
4. The Net table that includes:
    - **ID.** A Unique ID for each entry in the table.
    - **Due Date.** The order due date
    - **From Part/Operation.** Part and Operation ID of the child part.
    - **To Part/Operation.** Part and Operation ID of the parent part that the child part goes into.
    - **Resource ID.** Provides the name of the resource
    - **Setup Time.** Specifies the setup time of the particular resource in hours.
    - **Run Time.** Specifies the run time of the particular operation in hours.
    - **Quantity.** Specifies the number of parts that needs to be manufactured for the particular operation. This quantity includes work in progress

The generation of *The Net* is discussed in Chapter 4.

The following additional scheduling information will also be needed and must be provided by the user:

- **Work hours.** Specifies the Number of working hours available each day.
- **Segregations per day.** Specifies the number of sections each day is divided into.
- **Constraint buffer size.** Specifies the constraint buffer size in hours.
- **Shipping buffer size.** Specifies the shipping buffer size in hours.
- **Assembly buffer size.** Specifies the assemblly buffer size in hours.
- **Planning Horison.** Specifies the scheduling window in hours.

The scheduling program will implement the 5-step process of TOC described in the first section of this document and listed here again:

1. Identify the constraint
2. Exploit the constraint
3. Subordinate the remaining resources
4. Elevate the constraint
5. Display the results graphically

From a design perspective this makes things a little bit easier, because the creation of the system can then be done iteratively. Each of these steps will be described in detail in the following sections of this chapter.

The following will be discussed:

- The input to each step will be identified.
- The logic of what needs to be done to complete each step will be set out.
- The output of each step will be identified.

At this stage of the planning process no thought was given to how these steps were to be coded into a programming language, this was just to establish exactly what needed to be done.

## 4.1.1 The Program Flow Chart

The program flow chart is shown in *Figure 4.2* and illustrates the logical flow of the input the user must provide when the program is being used. It also shows the processes the program perform to generate the schedule as well as the different outputs at the different stages of the program.

Due to time constraints and the complexity of the problem, only the part of the chart that is enclosed in the dotted line has been coded as part of this thesis. The algorithms implemented will be described and laid out in detail in the following sections. The same applies to the parts of the program that was not coded. This will help with future development on this project.

The programming logic that was applied to implement the scheduling algorithm is described in the comments in the actual code of the program. This can be viewed by opening the program code supplied on the CD that is included with this report (Instructions on how to open and view the code is provided in *Appendix B*) or by referring to *Appendix A* where the relevant code has been included. As the GS algorithm is explained in this report, references will be made to the program code so that the reader will know how and where each step of the GS algorithm is implemented in the program code.

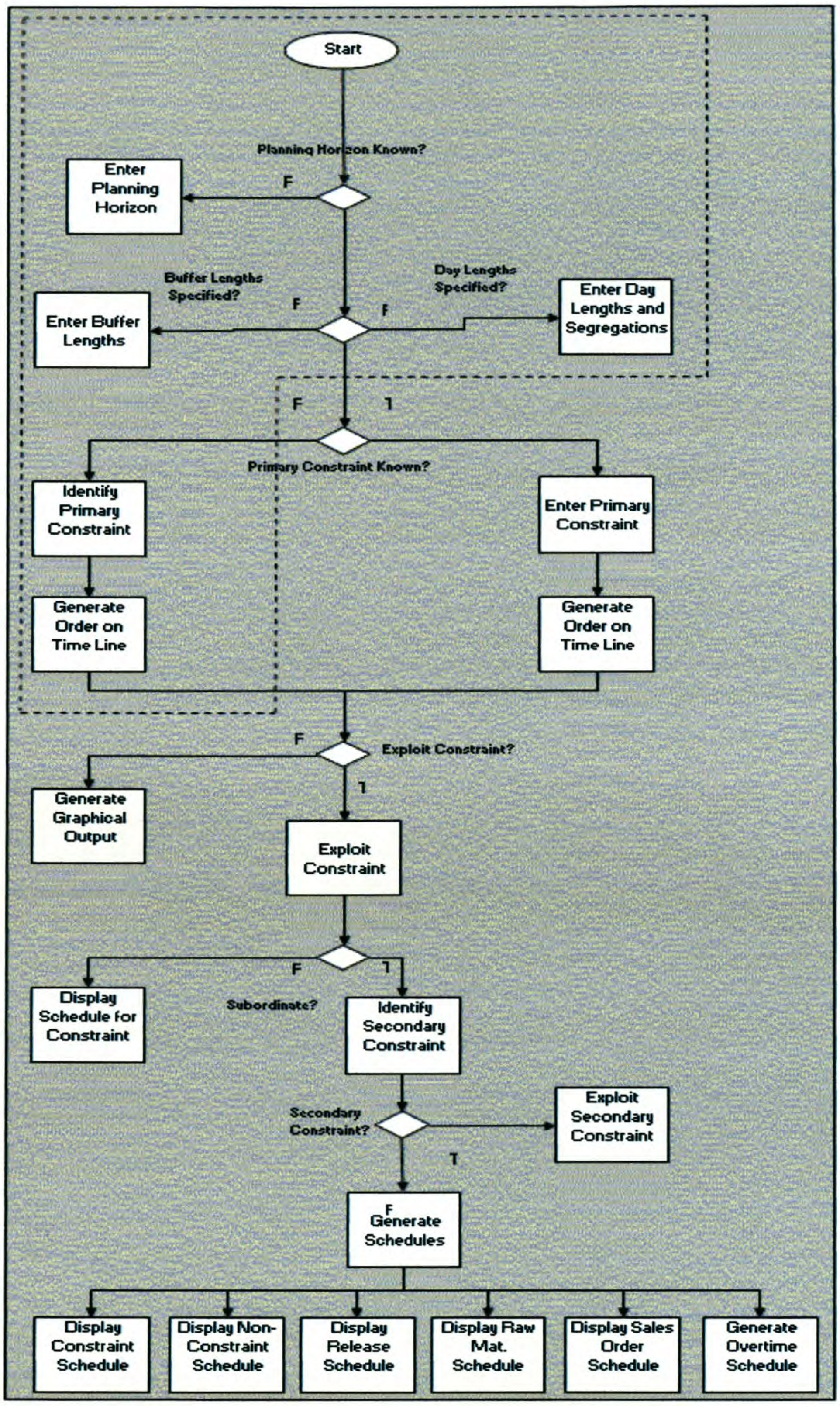

Start
Planning Horizon Known?
F
Enter Planning Horizon
Buffer Lengths Specified?
F
Enter Buffer Lengths
Day Lengths Specified?
F
Enter Day Lengths and Segregations
F
T
Primary Constraint Known?
Identify Primary Constraint
Enter Primary Constraint
Generate Order on Time Line
Generate Order on Time Line
F
Exploit Constraint?
T
Generate Graphical Output
Exploit Constraint
F
T
Subordinate?
Display Schedule for Constraint
Identify Secondary Constraint
Secondary Constraint?
Exploit Secondary Constraint
T
F
Generate Schedules
Display Constraint Schedule
Display Non-Constraint Schedule
Display Release Schedule
Display Raw Mat. Schedule
Display Sales Order Schedule
Generate Overtime Schedule

**Figure 4.2:** The Program Flow Chart

## 4.2 Identifying the Constraint

To properly schedule the factory, the system must be capacity sensitive and since a valid schedule must be generated on the system's constraints, the first step is to identify the primary constraint. To identify the constraint, the resource that has the greatest impact on the shipping buffer must be identified.

To identify the constriant the following information is needed:

- The Master Production Schedule (*The Net* table)
- Resource relationships (*The Net* table)
- Available resource capacity (*Calendar* table and user input)
- The shipping buffer (User input)

The identification phase must feed the exploitation phase with the following:

- The identitiy of the constraint
- Orders on a time line

One of the most important processes in the identification of the constraint is the generation of what is termed *The Net*. *The Net* is defined below, as well as the input needed to generate *The Net*.

### 4.2.1 The Net

The Net is a summary of all the information needed for the scheduling algorithm to function, combined into one database table (Stein 1998: 56 - 59, 225 – 235). This database table is housed in the *Data Mining and Conversion Module.*

All the resource relationships are defined in *The Net.* In other words it is a re-creation of the way in which products flow through the factory and includes:

- The combination of product flow diagrams for all products in the master schedule.
- Work in process inventory.
- Operational set-up and run times for which the demand appears in the Master Production Schedule (MPS).

It is used to:

1. Compute material requirements.
2. Determine which resources are to produce what products.
3. Understand how the resources interface.

*The Net* is the major source of data used to:

- Identify the constraint.
- Generate the schedule.
- Subordinate the remaining resources.

To maximise the efficiency of the program, *The Net* was generated and loaded directly into memory. This provides three major advantages:

1. The amount of times the *Data Mining and Conversion Module* will be accessed will be kept at a minimum. This saves a lot of processing time.
2. A very large number of data files can be reduced and stored in a very small amount of space within the system.
3. The time needed for creating a schedule or determining material requirements will be greatly reduced.

In short, the major advantage of *The Net* can be summarised in the fact that it will increase the overall speed of the system dramatically, because it is working directly from memory and the time consuming processes of reading and writing to disk or repeatedly accessing the database will be kept at a minimum.

*Figure 4.3* illustrates the transfer of data from the traditional system to *The Net* and where it fits into the scheduling process.

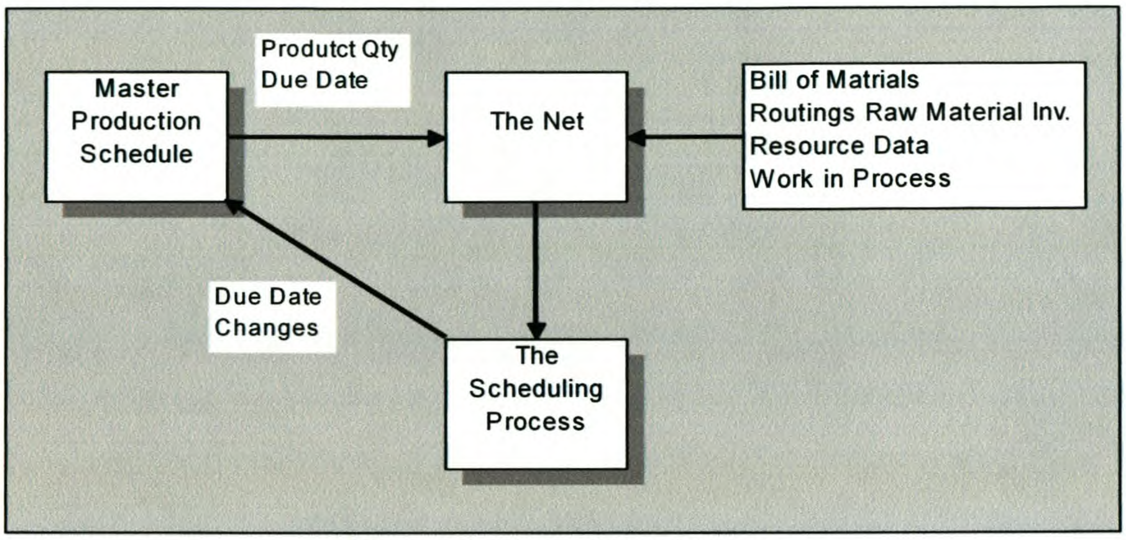


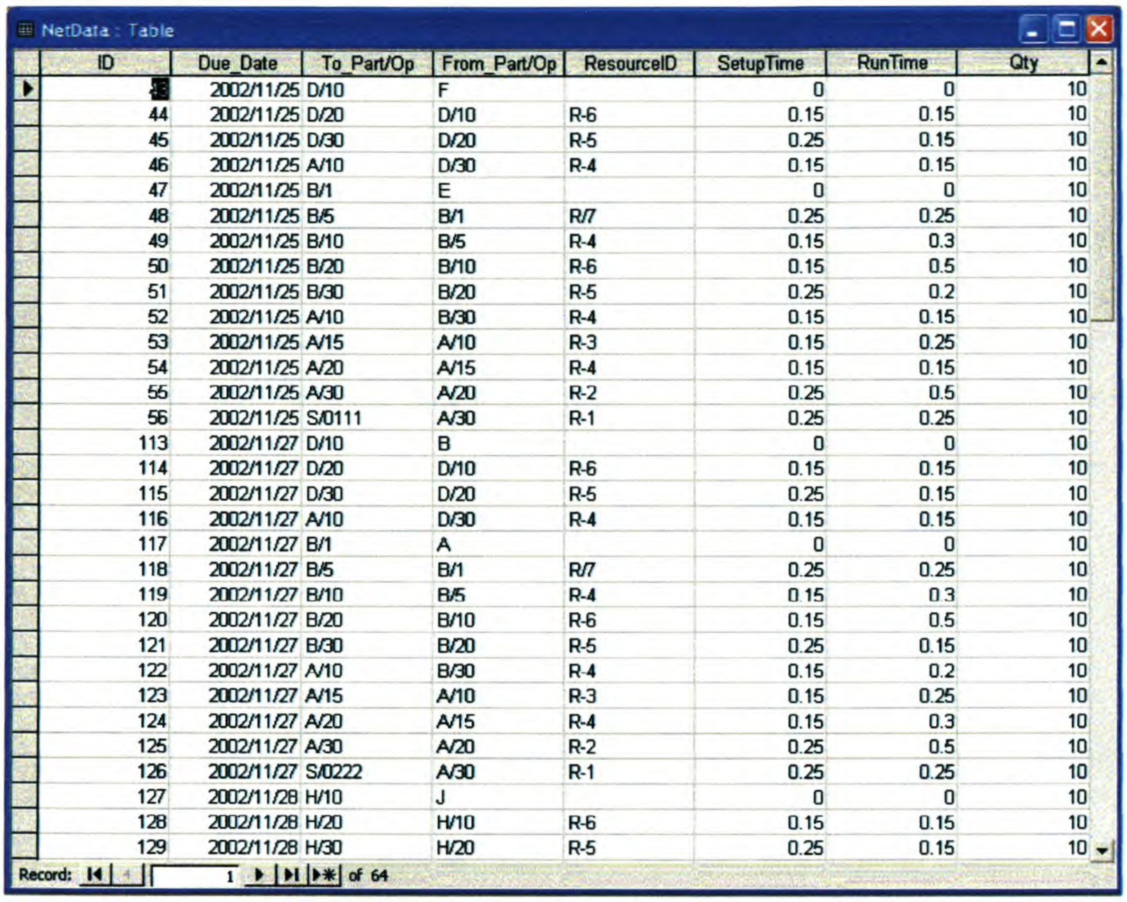
**Figure 4.3:** The TOC Based System

To generate *The Net*, the following information is needed from the following sources:

| Master Schedule | Raw Material | Resource Data | Part/Operation | Order of Processing |
|---|---|---|---|---|
| Product | RM Part | Resource | Part/Operation | To Operation |
| Quantity | Stock Quantity | Quantity | Resource | From Operation |
| Due Date | Lead Time | | Run Time | Quantity |
| | | | Setup Time | Scrap |
| | | | **Alt. Resource** | Rework |
| | | | Run Time | Rework Op. |
| | | | Setup Time | |
| | | | Inventory (WIP) | |

**Table 4.1:** Input to the net

All the information listed in *Table 4.1* is then combined into a single table in the database, ready for processing. This retrieval, combination and structuring of the information listed in *Table 4.1* is done by the DMCM discussed in *Section 2.1.2.* An example of what a populated *Net* structure looks like can be seen in *Figure 4.4.*



| ID | Due_Date | To_Part/Op | From_Part/Op | ResourceID | SetupTime | RunTime | Qty |
|---|---|---|---|---|---|---|---|
| 43 | 2002/11/25 | D/10 | F | | 0 | 0 | 10 |
| 44 | 2002/11/25 | D/20 | D/10 | R-6 | 0.15 | 0.15 | 10 |
| 45 | 2002/11/25 | D/30 | D/20 | R-5 | 0.25 | 0.15 | 10 |
| 46 | 2002/11/25 | A/10 | D/30 | R-4 | 0.15 | 0.15 | 10 |
| 47 | 2002/11/25 | B/1 | E | | 0 | 0 | 10 |
| 48 | 2002/11/25 | B/5 | B/1 | R/7 | 0.25 | 0.25 | 10 |
| 49 | 2002/11/25 | B/10 | B/5 | R-4 | 0.15 | 0.3 | 10 |
| 50 | 2002/11/25 | B/20 | B/10 | R-6 | 0.15 | 0.5 | 10 |
| 51 | 2002/11/25 | B/30 | B/20 | R-5 | 0.25 | 0.2 | 10 |
| 52 | 2002/11/25 | A/10 | B/30 | R-4 | 0.15 | 0.15 | 10 |
| 53 | 2002/11/25 | A/15 | A/10 | R-3 | 0.15 | 0.25 | 10 |
| 54 | 2002/11/25 | A/20 | A/15 | R-4 | 0.15 | 0.15 | 10 |
| 55 | 2002/11/25 | A/30 | A/20 | R-2 | 0.25 | 0.5 | 10 |
| 56 | 2002/11/25 | S/0111 | A/30 | R-1 | 0.25 | 0.25 | 10 |
| 113 | 2002/11/27 | D/10 | B | | 0 | 0 | 10 |
| 114 | 2002/11/27 | D/20 | D/10 | R-6 | 0.15 | 0.15 | 10 |
| 115 | 2002/11/27 | D/30 | D/20 | R-5 | 0.25 | 0.15 | 10 |
| 116 | 2002/11/27 | A/10 | D/30 | R-4 | 0.15 | 0.15 | 10 |
| 117 | 2002/11/27 | B/1 | A | | 0 | 0 | 10 |
| 118 | 2002/11/27 | B/5 | B/1 | R/7 | 0.25 | 0.25 | 10 |
| 119 | 2002/11/27 | B/10 | B/5 | R-4 | 0.15 | 0.3 | 10 |
| 120 | 2002/11/27 | B/20 | B/10 | R-6 | 0.15 | 0.5 | 10 |
| 121 | 2002/11/27 | B/30 | B/20 | R-5 | 0.25 | 0.15 | 10 |
| 122 | 2002/11/27 | A/10 | B/30 | R-4 | 0.15 | 0.2 | 10 |
| 123 | 2002/11/27 | A/15 | A/10 | R-3 | 0.15 | 0.25 | 10 |
| 124 | 2002/11/27 | A/20 | A/15 | R-4 | 0.15 | 0.3 | 10 |
| 125 | 2002/11/27 | A/30 | A/20 | R-2 | 0.25 | 0.5 | 10 |
| 126 | 2002/11/27 | S/0222 | A/30 | R-1 | 0.25 | 0.25 | 10 |
| 127 | 2002/11/28 | H/10 | J | | 0 | 0 | 10 |
| 128 | 2002/11/28 | H/20 | H/10 | R-6 | 0.15 | 0.15 | 10 |
| 129 | 2002/11/28 | H/30 | H/20 | R-5 | 0.25 | 0.15 | 10 |

Record: 1 of 64

**Figure 4.4:** Example of The Net

### 4.2.2 The Net Coding Structure Explained

As *The Net* is such an important part of this program a short explanation will be given on the way it was implemented in C++.

For this program to run as quickly and efficiently as possible, the method used to represent *The Net* in memory is very important. A dynamic linked list was used to do this. A linked list is an inherent feature of object orientated programming languages such as C++. Each object in the list is stored at a physical address in memory and is referenced by using pointers which point to the address in memory where the object is stored. An object in the list is represents all the values stored in a single row in the database table. The list can then be traversed by moving from one pointer to the next, thus moving from object to object consecutively. All the instances used in the linked list to store *The Net* are listed in the class, *CNet*, in the program code. The instances correspond closely to the fields defined in the *NetData* table in the database. The pointer variables used to traverse the list is also defined in this class.

The advantages of a structure like this are that it is completely dynamic and the programmer can add and remove instances to and from this list with ease. It is also memory efficient because there is no replication of data in memory. The data is also retrieved directly from where it is physically stored so there is no unnecessary passing of variables between different procedures. This makes data accessing extremely fast and memory usage very efficient. A graphical representation of a linked list is shown in *Figure 4.5*. The hexadecimal values, seen in *Figure 4.5,* are the actual memory addresses of where the objects in the list are stored. The arrows represent the pointer variables.

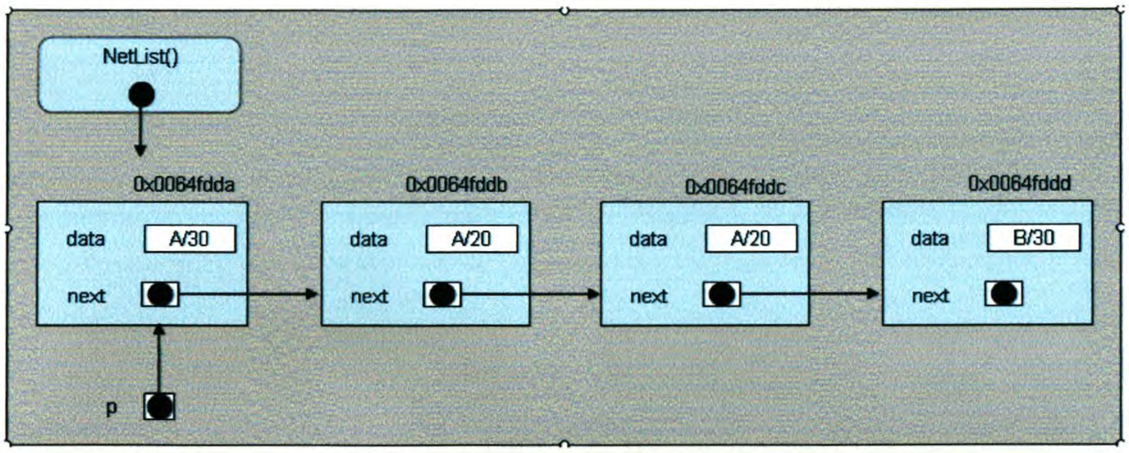


**Figure 4.5:** Linked List

The fact that the programmer was able to use structures like this was one of the main deciding factors in using an object orientated language such as C++. C++ provides the programmer with the ability to write code that is fast and efficient, which would not have been possible in a programming language like Visual Basic. The reason for this is that Visual Basic was basically developed for the development of front end applications with extensive the graphical user interfaces where the processing is not so resource intensive. An example of the power of C++ is that most Microsoft applications like MS Windows 2000 have been developed using Visual C++ or Embedded C++.

# Chapter 5

## 5.1 The Process of Identifying the Primary Constraint

The actual programming part of this thesis starts with the assumption that the MRP system has been queried by the *Data Mining and Conversion Module* and *The Net* has been generated for a list of orders and inserted into the database in the correct format. The *TOC Scheduling Module* is now ready to access the database and perform the scheduling procedure.

The first step in the actual development process was to use all the information identified for the scheduling process in the previous chapters to generate level 1 DFD showing a more detailed level of the processes and data required tot identify and schedule the primary constraint. This DFD is shown in *Figure 5.1.*

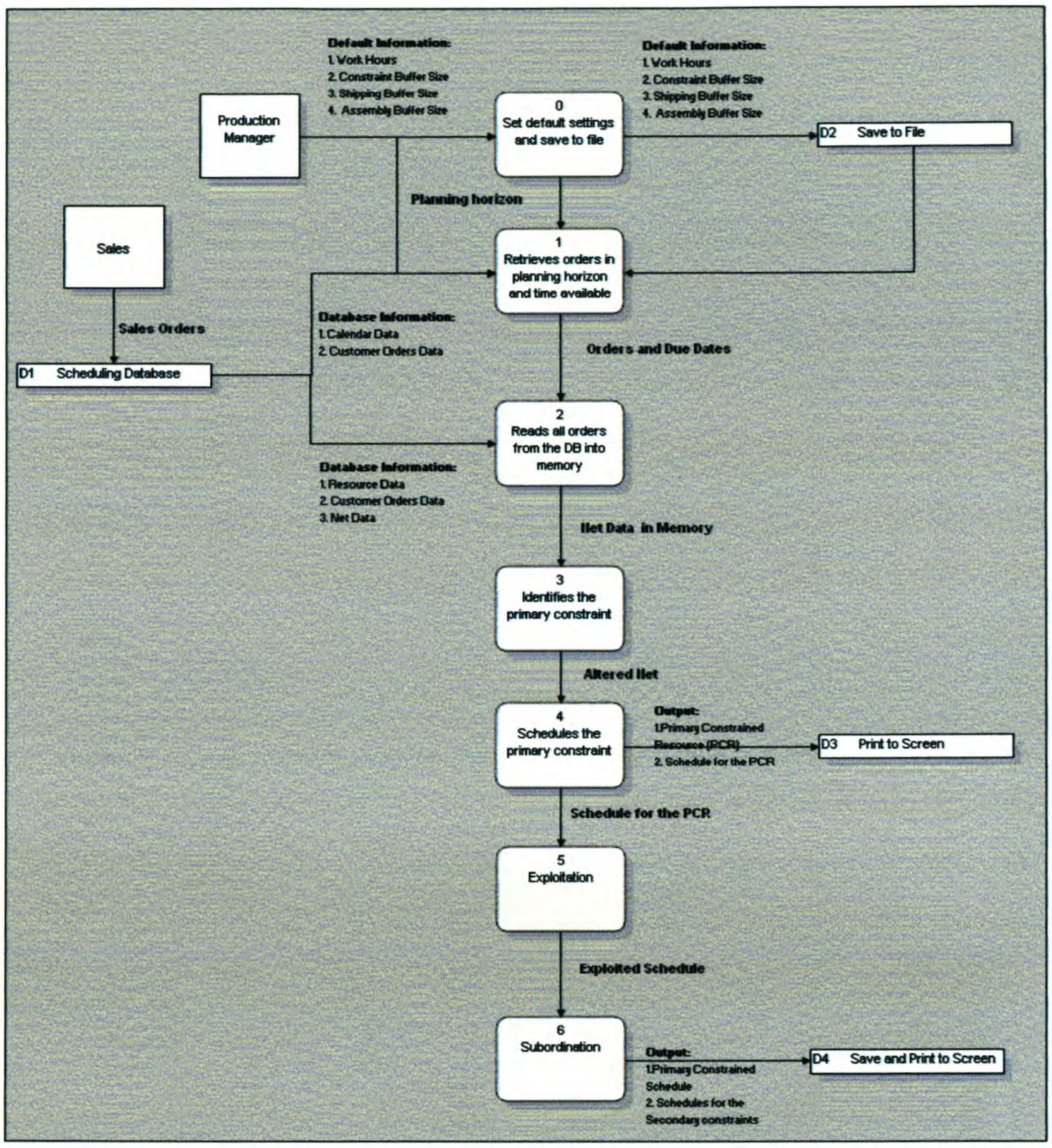


**Figure 5.1:** Level 1 DFD

Processes 3 and 4 on the DFD in *Figure 5.1,* for scheduling the primary constraint, was then expanded into more detailed steps defining what needed to be to identify the primary constraint. These steps are summarised in *Figure 5.2.* The rest of this chapter explains how these steps were implemented into programming code.

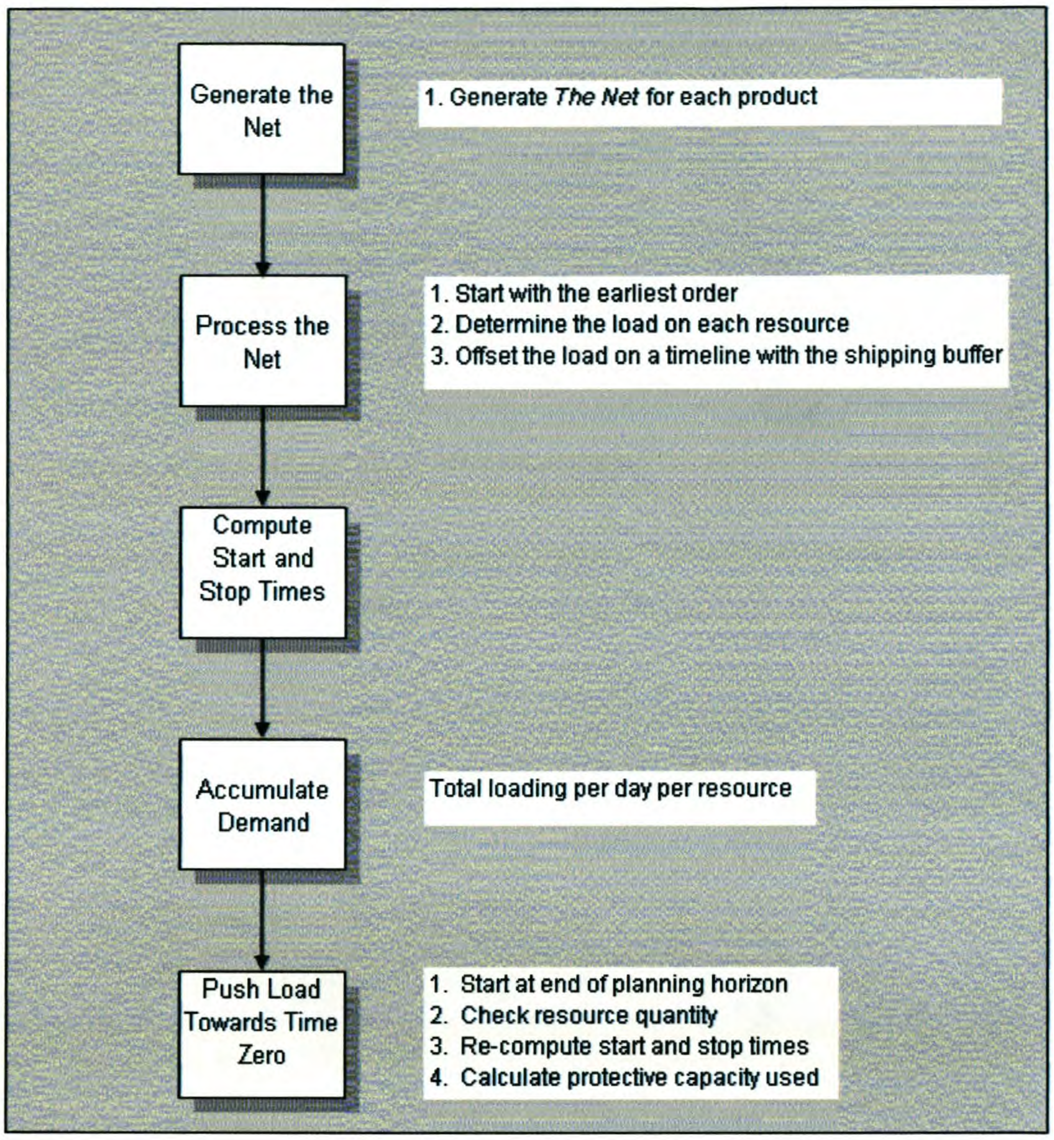

Generate the Net
1. Generate *The Net* for each product
Process the Net
1. Start with the earliest order
2. Determine the load on each resource
3. Offset the load on a timeline with the shipping buffer
Compute Start and Stop Times
Accumulate Demand
Total loading per day per resource
Push Load Towards Time Zero
1. Start at end of planning horizon
2. Check resource quantity
3. Re-compute start and stop times
4. Calculate protective capacity used

**Figure 5.2:** Identifying the Constraint

### 5.1.1 **Compute the Effective Planning Horizon**

The first step is to compute the effective planning horizon. The planning horizon is the time period over which the user wants to schedule by selecting a *Start Date* and an *Due Date* on the *Scheduling Parameters* form that can be selected from the main menu. All the orders between those dates are identified and the time available for each order to be completed is calculated, in hours, and then inserted into *The Net*. To do this the program references the *Calendar* table in the DMCM and checks whether it is a working day or not. All the working days between the *Start*

*Date* and the *Due Date* of the particular order is then calculated. (Code Ref: Class *Planning Horizon,* Procedure *GetWorkdayCount)* At this stage no individual daily capacities can be specified. The resources are available for the amount of time specified in the *Day Length* input variable or not at all. The capacity for each resource can be increased or decreased by changing the run time on each resource in *The Net* Table (MS Access Table Name: *NetData).* (Code Ref for the input variables: Class *CBufferLenght,* Procedure *SaveBuffers)*

A further refinement of this procedure would be to specify daily capacities for each resource in the *Calendar Table* and then calculate the work time available to complete each order in such a way. The effective planning horizon for each operation of each order is calculated as follow:

$$\text{Effective Planning Horizon} = \text{Planning Horizon} - \text{Shipping Buffer}$$

The shipping buffer is included as an additional time period of time allocated for performing any operation that must be accomplished following constraint processes, but before the job's due date. Adding the shipping buffer ensures that planning is accomplished for jobs whose constraint processing must be completed during the intended scheduling horizon, so that the shipping buffer before their order due dates is not eaten into.

After this operation has been completed, *The Net* for those orders within the planning horizon will be queried from the *Data Mining and Conversion Module* and read into the dynamic linked list into memory, ready for further processing.

### 5.1.2 Subordinate All Resources to the Market

As *The Net* is being read into memory, starting with the earliest order date, each job's operational requirements are subtracted from the remaining capacity of each affected resource. (Code Ref: Class *CIDConstraint,* Procedure *RetrieveNetData)* This is demonstrated in *Figure 5.3,* for a single resource. Notice that the end time for each operation remains the same for each load placed on the resource. This seems illogical but the objective of this exercise is not to create a schedule for the factory but to determine which resource poses the biggest threat to the creation of throughput. This resource is also the constrained resource.

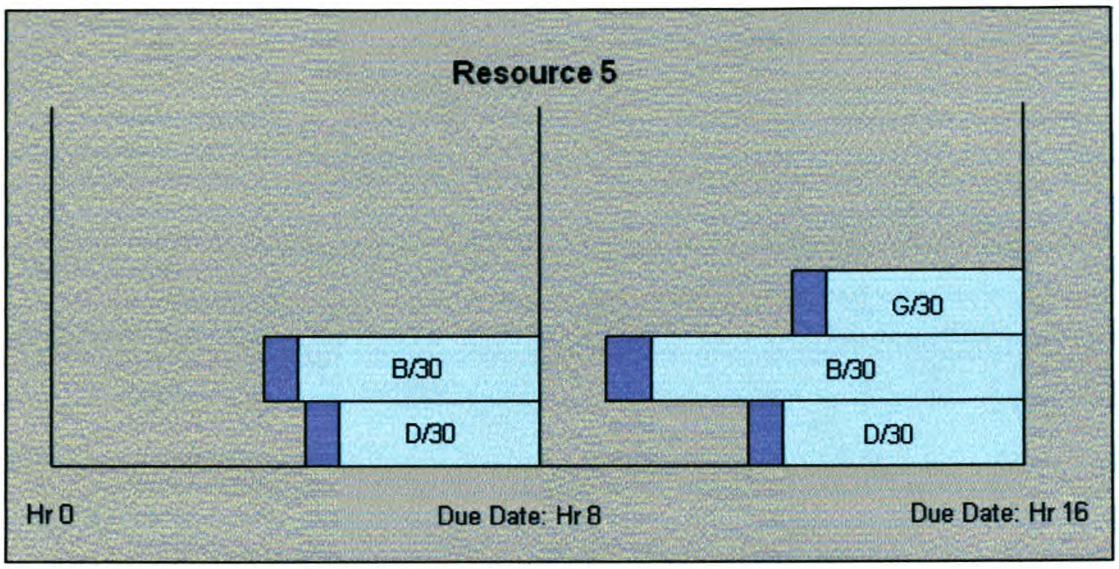


**Figure 5.3:** Accumulating Demand

The next step in the process is to level the load by pushing the load on each resource toward time zero. Since the protection exists to the right, the process of levelling the load begins at the end of the planning horizon and move constantly towards time zero. This is done by beginning with the latest due date and subtracting each job's operational requirements from the remaining capacity of each affected resource. A graphical representation is shown in *Figure 5.4.* (Code Ref: Class *CIDConstraintt,* Procedure *IdendityFDLPeaks)*

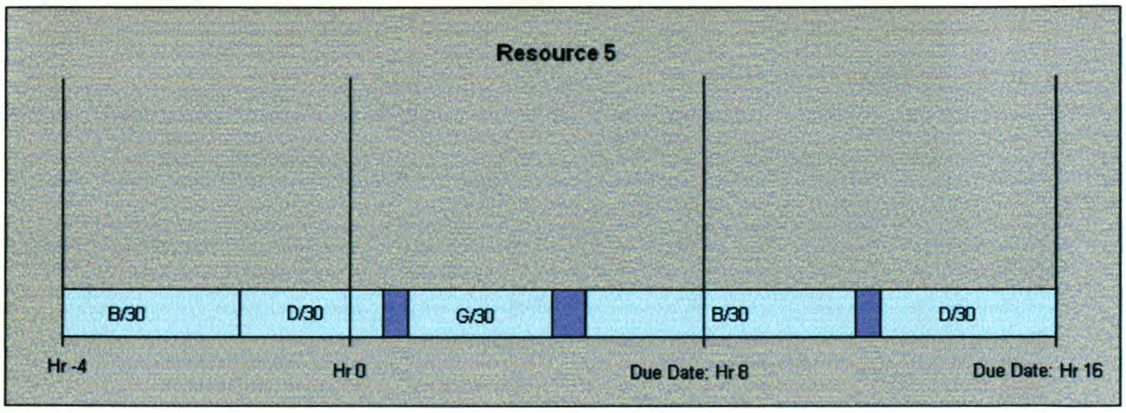


**Figure 5.4:** Rough cut capacity checking

Notice that setup times are only included once for the same operations on different orders. This is because setup times are variable and not always accurate and can have an influence in the determination of the constraint (Stein, 1998: 76). Setup savings, however, plays an important part in the exploitation phase of the scheduling exercise and it can be manipulated to good effect to create an optimum schedule for the primary constraint and the system. A short description of the techniques used for the exploitation of the schedule will be discussed in *Section 6.1.2.*

The step described above is only rough cut capacity checking and determines when each particular job must start so that it will finish on time. This rough cut capacity checking allows the user to set effective capacity levels at some level lower than the apparent capacity, to accommodate timing complications within the work day.

When resources have excessive loads, the net effect will be an accumulation of workload flowing backward. When the load on a resource has been pushed backward so that is must complete work in the past, then the resource might be a Capacity Constrained Resource (CCR). In other word a resource might be a CCR when it has a load on the first day, greater than its available capacity. Such a condition is referred to as a FDL (First Day Load) peak (Simons, Wendell, 1998: 8). If it happens that more than one FDL peak occurs; the resource with the largest FDL peak is the PCR (Primary Constrained Resource). In this case it is the resource that has been scheduled to start work the furthest into the past. (Code Ref: Class *CIDConstraintt,* End of Procedure *IdendityFDLPeaks)*

After the primary constraint has been identified the setup times that were not previously included, must now be incorporated into the schedule. This is done by recalculating the *total time* of each operation and the levelling the loads again. (Code Ref: Class *CIDConstraintt,* End of Procedure *InsertSetupTimes)*

## 5.2 Creating a Drum Schedule for the Primary Constrained Resource

The GS algorithm makes provision for when the PCR feeds itself further down the line. This happens when a job is processed by the constrained resource and moves on to be processed on another non-constrained resource, and then returns to the original constrained resource for a different operation. This might cause a problem if insufficient time is allocated for the intervening non-constraint operations. The operations on the constraint must be scheduled far enough apart so that the constraint's schedule will not be disrupted by delay, because the constraint is unable to begin with the second operation.

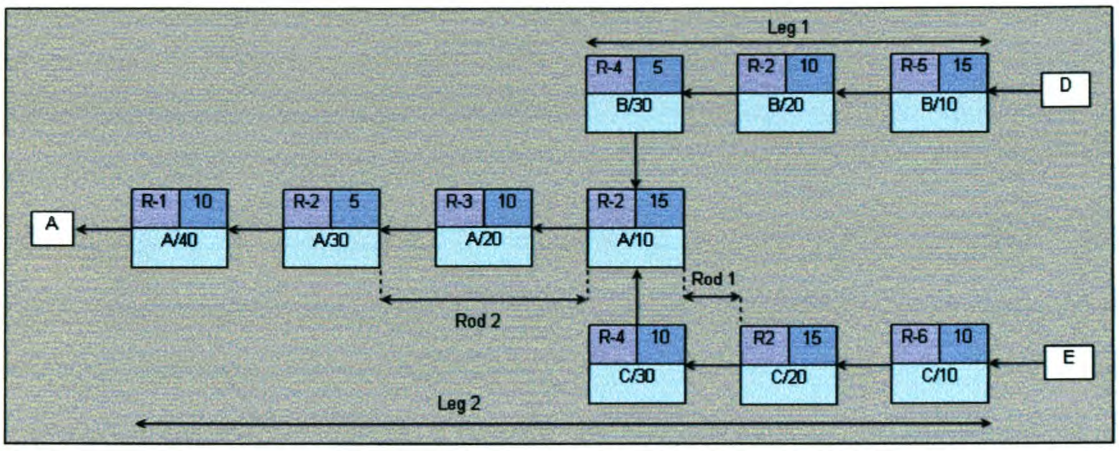


**Figure 5.5:** Product Example

To solve this problem rods are introduced into the system. A rod is nothing more than a time lag between the constraint operations and ensures that there is a minimum time gap between constrained operations on the same job (Simons, Wendell, 1998: 7 - 8). *Figure 5.5* illustrates where rods need to be inserted between jobs where the product is assembled from more than one production leg. Rods are only inserted between more than one occurrences of the PCR on the same production leg. The product leg that is scheduled first is dependent on the way it was inserted into *The Net.*

The type of rod used in this case is called a batch rod, since it provides the required tie between two processes. The rod length may be arbitrarily set at one half the constraints normal buffer size. The rod's placement in the constraint schedule is a function of 2 factors:

1. *Transfer batch size.* This is the number of units that must be completed on a job before they can be transferred to the next operation.
2. *Relative per unit processing in the two constraint operation being spaced apart.* When the processing time per unit is smallest for the earlier constraint operation, the rod is placed between the scheduled completion of the first unit on the first constraint and the scheduled start of processing of the first unit on the second constraint operation. This is illustrated in

*Figure 5.5.* To determine the actual delay between the end of the first operation and the start of the second operation for this case, the following formula was used:

Delay = ½ * Constraint Buffer – (First Operation( Batch Time – Setup Time – Run Time ))
- Second Operation( Setup Time )

This formula was derived from *Figure 5.6.* (Code Ref for Inserting Rods: Class *CIDConstraintDlg,* Procedure *InsertRods)*

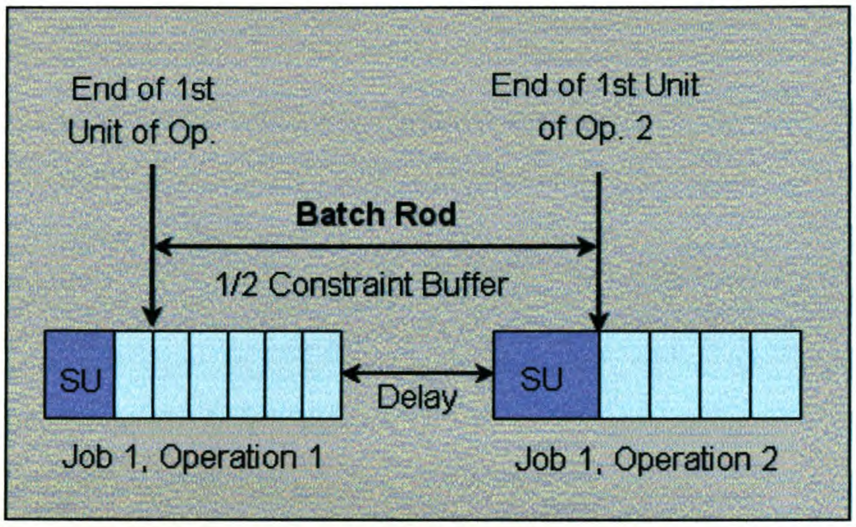


**Figure 5.6:** Batch rod with smaller time per part for the earlier operation

On the other hand, when the processing time per unit is largest for the earlier constraint operation, the rod is placed between the scheduled completion of the last unit on the first constrained operation and the scheduled start of processing of the last unit on the second constrained operation. This is illustrated in *Figure 5.7.* The delay between the end of the first operation and the beginning of the second was derived from *Figure 5.7* and calculated as follow:

Delay = ½ * Constraint Buffer – Second Operation( Batch Time – Run Time)

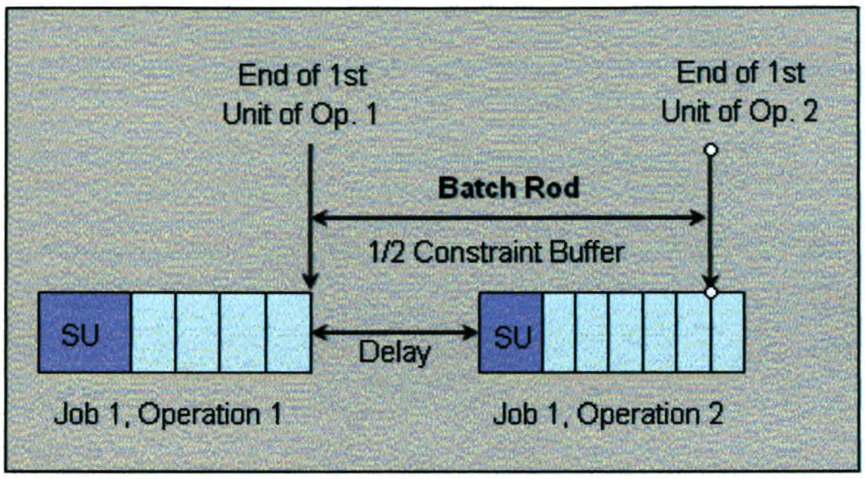


**Figure 5.7:** Batch rod with larger time per part for the earlier operation

The process of inserting the rods has the effect that the orders on the time line are spread over a longer period of time and are moved back in time, so that it can happen that the some of the orders will be scheduled to happen in the past. This is solved by moving all the orders far enough forward in time so that no work is scheduled in the past. This step completes the first iteration of the scheduling of the constraint and is also up to where the GS algorithm was implemented and coded. The rest of the report will continue describe the rest of the GS algorithm and provide the programming steps on how to complete the scheduling code.

# Chapter 6

## *6.1 The Exploitation Process*

Section 5 ended after the constraint has been scheduled for the first time. This was illustrated in *process 4* in the DFD in *Figure 5.1*. It is most likely that after this process that some jobs have less than the desired shipping buffer or they exceed their order due dates. If this happens, this is the place where the constraint must be exploited so that the orders do not exceed their due dates. Exploitation is not included a part of the GS algorithm but the process of exploitation will be described because it is a necessary and important step in the TOC philosophy. For the constraint to be exploited so that order due dates can be achieved on time, the constraint schedule must be revised in such a way that:

- The constraint must be constantly operating to generate throughput.
- Whenever and wherever possible, additional constraint time must be found.
- Those orders which may have problems must be identified so that action can be taken.

To be able to exploit the constraint the following information is needed:

- The sequencing of orders on the time line.
- Decisions of how to exploit the constraint.
- Setup and run times for each operation.
- Available resource capacity.

The exploitation phase will provide a schedule for the primary constraint which will include start and stop times for every order within the planning horizon.

The exploitation process include in *Figure 5.1* as *process 5* is broken down into more detailed steps and is illustrated in *Figure 6.1*.

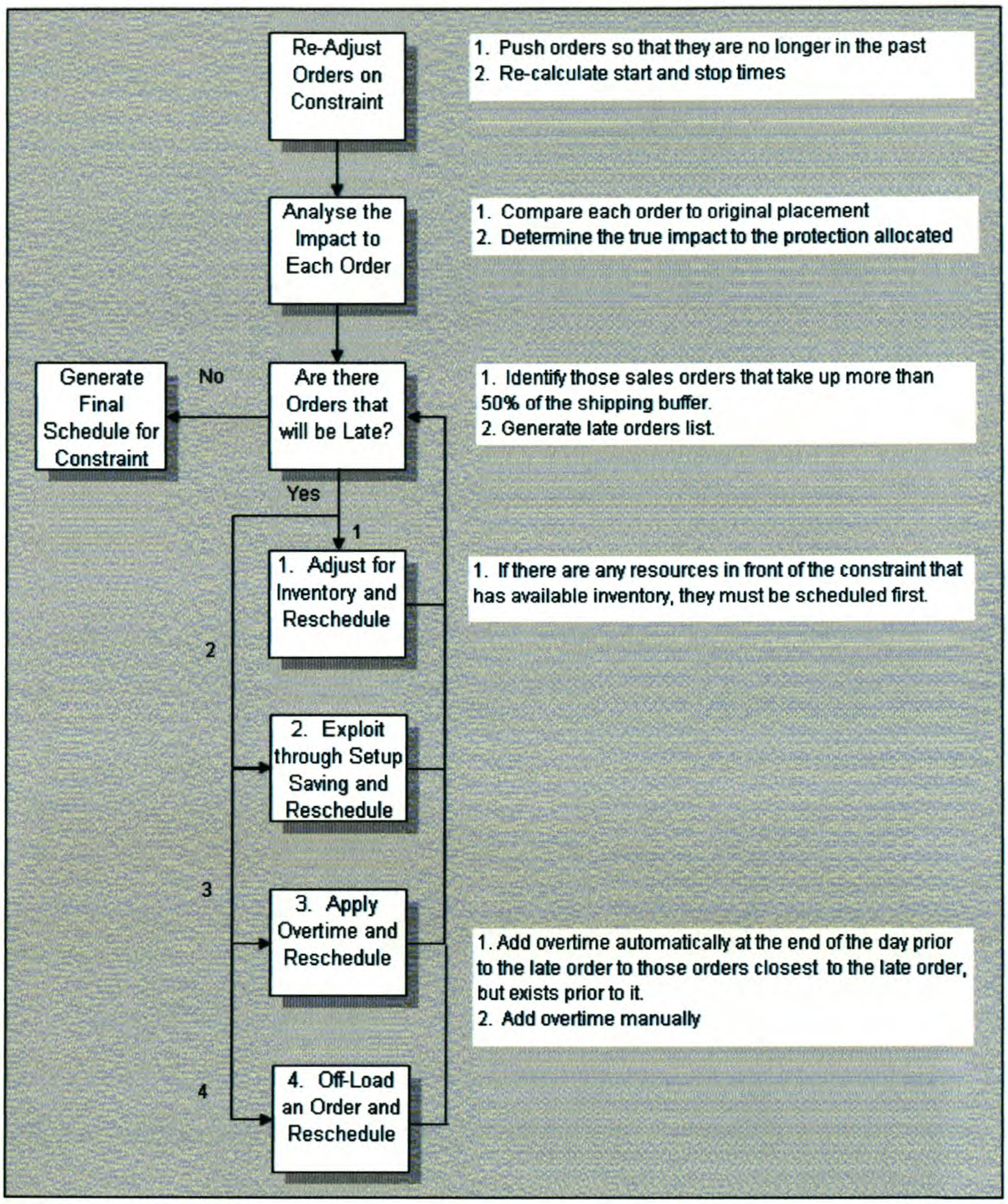

Re-Adjust Orders on Constraint
1. Push orders so that they are no longer in the past
2. Re-calculate start and stop times
Analyse the Impact to Each Order
1. Compare each order to original placement
2. Determine the true impact to the protection allocated
Generate Final Schedule for Constraint
No
Are there Orders that will be Late?
1. Identify those sales orders that take up more than 50% of the shipping buffer.
2. Generate late orders list.
Yes
1
1. Adjust for Inventory and Reschedule
1. If there are any resources in front of the constraint that has available inventory, they must be scheduled first.
2
2. Exploit through Setup Saving and Reschedule
3
3. Apply Overtime and Reschedule
1. Add overtime automatically at the end of the day prior to the late order to those orders closest to the late order, but exists prior to it.
2. Add overtime manually
4
4. Off-Load an Order and Reschedule

**Figure 6.1:** The Exploitation Process

As can be seen from *Figure 6.1,* there are four possible ways that the constraint might be exploited. The four exploitations actions also provide the opportunity for the shop floor workers and the production managers to use their experience in an empowered environment. A brief description of these methods will be given in the following sections.

### 6.1.1 Adjusting for Inventory

The PCR can be exploited by scheduling operations on the PCR that has inventory or work in progress (WIP) available first, before operations that do not. To provide sufficient protection for the constraint in reality, enough orders with inventory must be put in front of the constraint to cover two-thirds of the length of the constraint buffer.

### 6.1.2 Exploiting the Constraint through Setup Savings

Additional time on the constrained resource can be found by time savings on setup. *Figure 6.2* illustrates how time on the constrained resource can be optimized by grouping the same operations together so that multiple setups for the same operation are eliminated. Setup savings are the most effective in situations where the sizes of individual setups are large or the number of setups within the planning horizon is high.

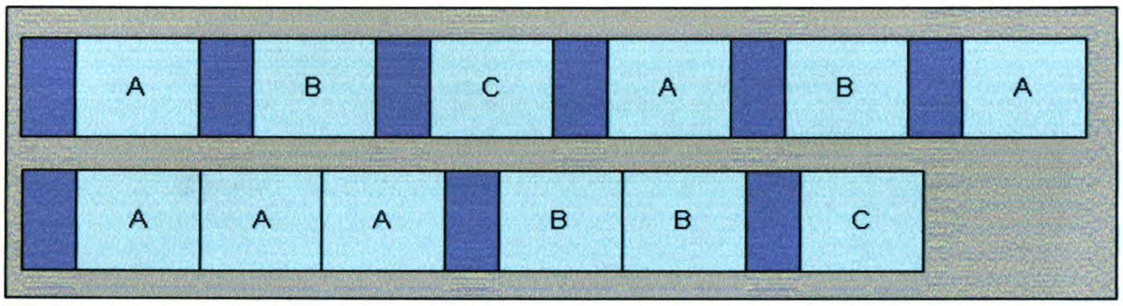


**Figure 6.2:** Setup Savings

It is, however very important to remember that whenever two orders are combined, another order's start date is postponed. From *Figure 6.2 it* can be clearly seen that after the adjustments for setup savings have been made, the first occurrence of operation B and operation C is scheduled to start at a later time. The biggest impact on start time is on operation C.

### 6.1.3 Applying Overtime

The goal of applying overtime is to increase throughput on the PCR while keeping inventory and operating expense to a minimum. Before overtime can be applied the late sales orders must first be identified.

Sales orders are considered to have serious timing problems whenever the due date of the order crossing the constraint uses over 50% of the shipping buffer. Late orders must be identified in this manner. Only if the constraint has already been exploited by adjusting for inventory and making use of setup savings, and there are still orders that are late, must overtime be applied.

Overtime must be applied at the correct time and place so that inventory and operating expense can be held at a minimum while maximizing throughput. If overtime is applied to early, it will increase inventory or if it is applied to late it will increase operating expense without affecting the throughput.

To be able to apply overtime correctly it is important to identify when and where to place the overtime and the impact it will have on the order due dates and what resources are involved.

### 6.1.4 Off-Loading

If all the previous attempts at exploitation have failed, it might be a good idea to off-load an order to another resource which is capable of handling it. The objective of off-loading is to relieve the strain on the constrained resource. It is important to choose an order or a number of orders, which exists early in the schedule and approximately equals the total amount of the late buffer minutes and for which there is a resource that will not be negatively affected.

Unfortunately the only way to understand the impact of an order which has been off-loaded is to subordinate the resource on which the order is now being processed with the schedule of the primary constraint. It may be found that it is impossible to schedule the remaining resources to the additional capacity generated.

An alternative solution might be to split lots and only off-loading part of the particular batch. It may be that in off-loading less that the entire amount the user is able to prevent problems occurring while attempting to subordinate to the constraints schedule.

The exploitation methods described above have not been built into the scheduling program. The constrained resource can however be exploited by editing *The Net* table in the data base manually. In a completed version of the scheduling program the user will be able to exploit the constraint through a user friendly graphical interface.

# Chapter 7

## 7.1 Subordination

As mentioned earlier, the subordination process was not implemented in the scheduling package that was developed as part of this thesis due to timing constraints. The goal of this chapter is to explain the subordination process in such a manner as to provide the reader with a basic understanding of the process as well as to give insight into the logic that must be used for further development on the package.

The objective of the subordination phase is to:

- Coordinate the activity of all other resources within the factory to ensure that the schedule established for the constraint is protected.
- That shipping is completed on time
- That raw material is released to the floor in synchronization with the way in which it will be consumed.

The subordination process must perform the following functions:

- Determine whether there is enough protective capacity on the non-constraints to protect the schedule and automatically increase the size of the buffer for those buffer origins needing additional capacity.
- Identify those resources for which protective capacity is inadequate and which threaten the schedule of the primary constraint.
- Exploit those resources identified as additional physical constraints to maximize their ability to deliver to the primary constraint.
- Ensure that there are no conflicts between the schedules created.
- Prepare the release schedule for raw material into the gating operations and synchronizing the activities of the non-constraints.

The input to the subordination phase must include:

- The schedule of the primary constraint
- The relationships between the resources
- Available resource capacity
- Constraint/assembly buffers
- Setup and run times

The output of the subordination phase includes:

- The schedule of the secondary, tertiary and remaining constraints
- The schedule for assembly buffer origins
- The release schedule for the gating operations
- Raw material purchasing requirements

The output after the subordination process has been completed is summarised in *Table 7.1*:

| **Constraint(s) Schedule** | **Non-Constr. Schedule** | **Release Schedule** | **Raw Mat. Schedule** | **Sales Order Schedule** | **Overtime Schedule** |
|---|---|---|---|---|---|
| Constraint ID | Resource ID | Resource ID | RM Part | Sales Order | Resource |
| Order ID | Order ID | Part/Op | Stock Qty. | Due Date | Date |
| Part/Op | Start Date.Time | Qty. | Order Date | Qty. | Amount |
| Order/Prod | Due Date/Time | Start Date/Time | Delivery Date | | |
| Qty | | | Order Nr. | | |
| Run Time | | | | | |
| Setup Time | | | | | |
| Start Date/Time | | | | | |
| Finish Date/Time | | | | | |
| Orig. Date/Time | | | | | |

**Table 7.1:** Output from the Scheduling Process

The subordination process illustrates as *process 6* in *Figure 5.1* has been broken down into detailed steps of the process and is described in *Figure 7.1:*

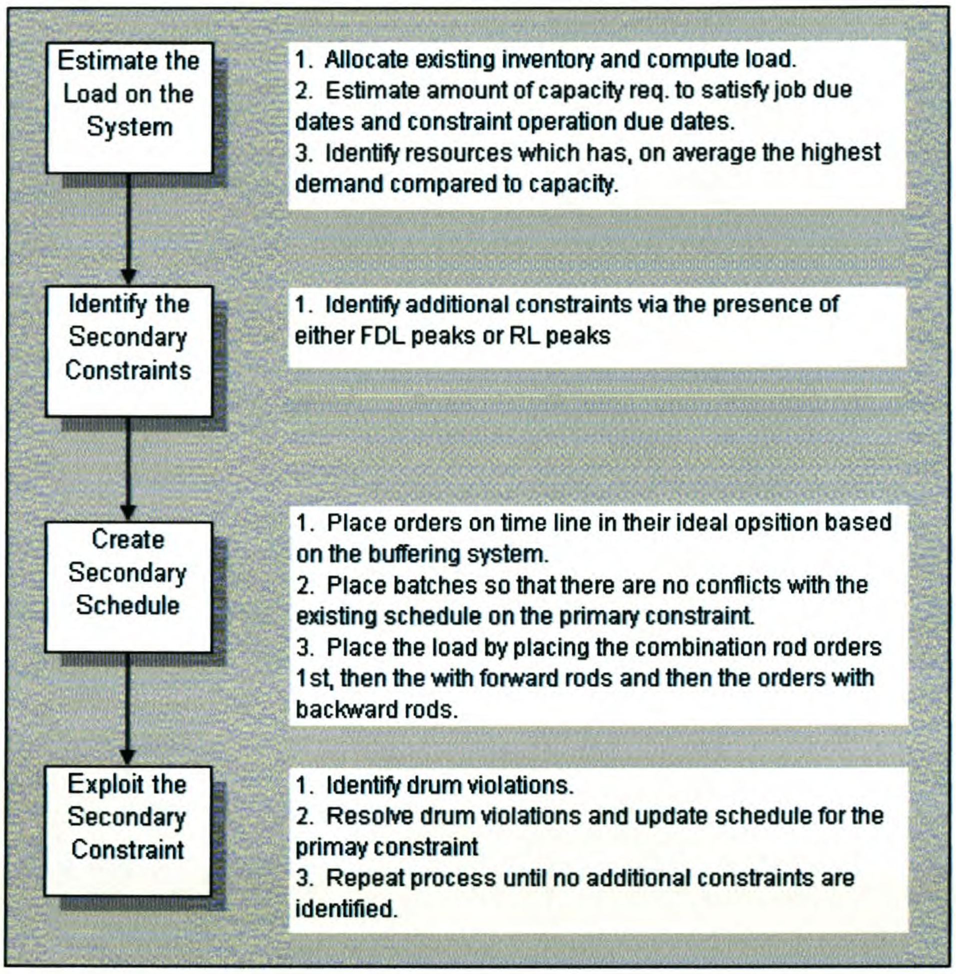

Estimate the Load on the System
1. Allocate existing inventory and compute load.
2. Estimate amount of capacity req. to satisfy job due dates and constraint operation due dates.
3. Identify resources which has, on average the highest demand compared to capacity.
Identify the Secondary Constraints
1. Identify additional constraints via the presence of either FDL peaks or RL peaks
Create Secondary Schedule
1. Place orders on time line in their ideal opsition based on the buffering system.
2. Place batches so that there are no conflicts with the existing schedule on the primary constraint.
3. Place the load by placing the combination rod orders 1st, then the with forward rods and then the orders with backward rods.
Exploit the Secondary Constraint
1. Identify drum violations.
2. Resolve drum violations and update schedule for the primay constraint
3. Repeat process until no additional constraints are identified.

**Figure 7.1:** The Subordination Process

## *7.2 Subordinate Non-Constraint Resources*

To complete this step, rough-cut capacity checking must be done for the non-constraint resources. This is done in essentially the same manner as the rough-cut capacity checking method explained in *Section 5.2.* The only difference is that the non-constraints are now determined by both the original job due dates and the operation due dates required to support the constraint schedule and not just the order due dates as explained earlier. In other words all the operations that happen before the constraint must be evaluated against the constraint schedule and all the operations happening after the constraint must be evaluated against the original order due date. This is referred to as a FDL peaks and a RL (Red Lane) peaks respectively (Simons, Wendell, 1998: 11).

It will not necessarily happen that the same resource will be identified as a constraint, because of the fact that the schedule for the primary constraint has been modified by inserting rods and exploitation. This means that additional timing constrains have been generated to support the drum schedule because the overall completion dates of orders has been modified or moved further into the future.

### 7.2.1 Identifying the Secondary Constraint

The secondary constraint is that resource which has created the greatest threat to the protection provided by the buffers. In other words the secondary constraint is that resource which has done the most damage to the protection needed for the system to perform the current demand.

To determine the secondary constraints the FDL peaks must be identified in exactly the same manner as the FDL peaks were identified for the PCR in *Chapter 5*. The only difference is that the due date that must be used is not the order due dates but the start date and time of the operations on the constraint.

To identify the FDL peak only the start and end times of the operations that happen before the constraint need to be considered and changed. The programming steps to complete this function are listed below and are illustrated graphically in *Figure 7.2*:

1. Make one backward pass through *The Net* for each resource except the PCR.
2. Ignore all operations in the order that occurs subsequent to the PCR.
3. Set the end time of the first resource that occurs before the PCR equal to the start time of the PCR.
4. Calculate the new start time of the resource.
5. Move to the next occurrence of the particular resource and set its end time equal to the start time of the previous occurrence of the resource.
6. Calculate the new start time of the resource.
7. Repeat the process until the start of the list has been reached.

8. Repeat the process for all the remaining resources in the list.

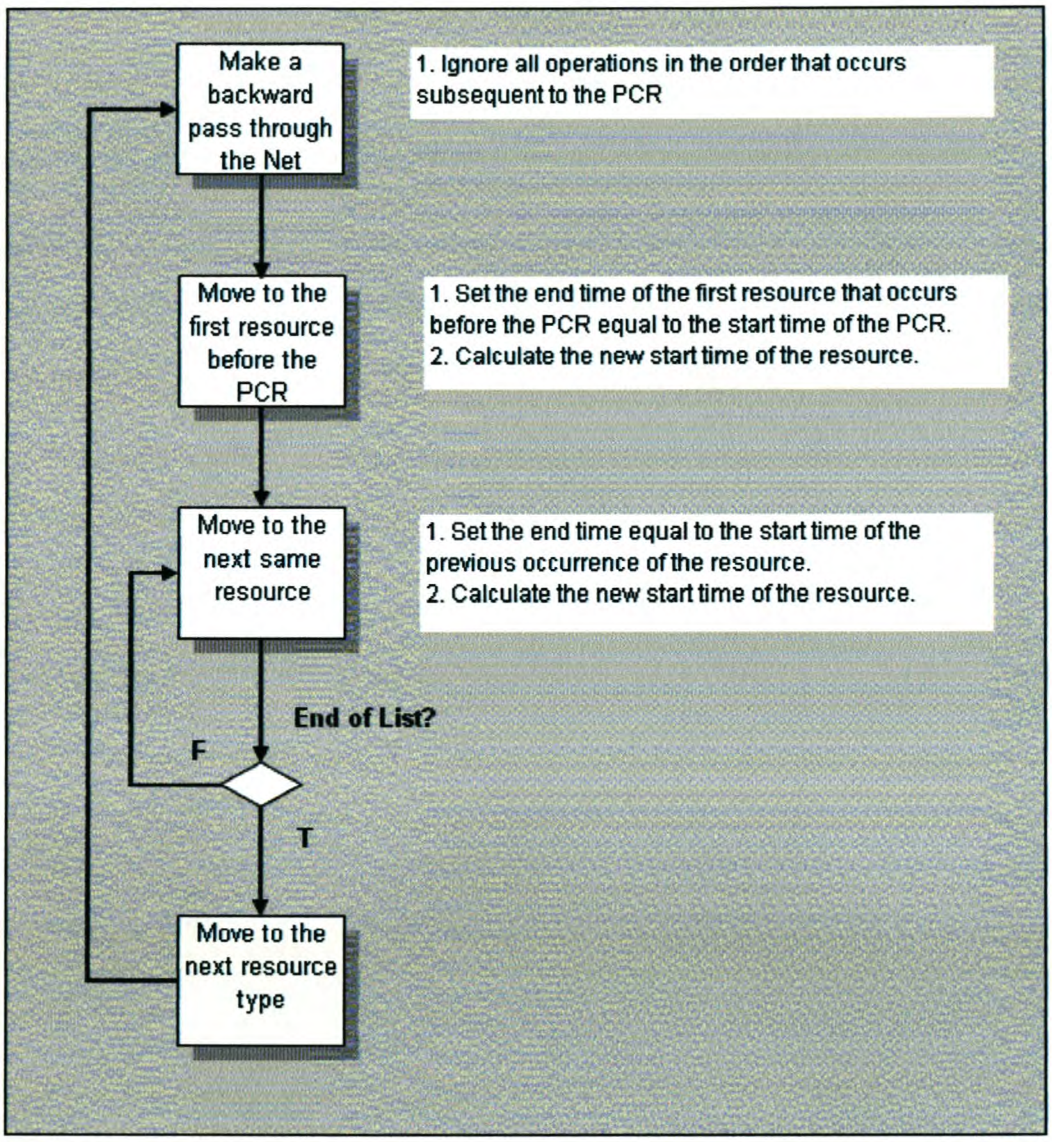

Make a backward pass through the Net
1. Ignore all operations in the order that occurs subsequent to the PCR
Move to the first resource before the PCR
1. Set the end time of the first resource that occurs before the PCR equal to the start time of the PCR.
2. Calculate the new start time of the resource.
Move to the next same resource
1. Set the end time equal to the start time of the previous occurrence of the resource.
2. Calculate the new start time of the resource.
End of List?
F
T
Move to the next resource type

**Figure 7.2:** Identifying the FDL peaks for the secondary constraint

The resource that has the largest FDL peak or that is scheduled the furthest past time zero might be your secondary constraint. To be sure all operations in the red lane must also be evaluated.

## 7.2.2 Dealing with Peak Load in the Red Lane

The red lane is defined as that area of *The Net* which occurs between shipping and the primary constraint. Determining the RL peaks makes use of the same logic that was described in the previous section. To identify the RL peak only the start and end times of the operations that happen after the constraint need to be considered and changed. The programming steps to complete this function are listed below and are illustrated graphically in *Figure 7.3*:

1. Make one forward pass through *The Net* for each remaining resource.
2. Ignore all operations in the order that occurs before the PCR.
3. Set the start time of the first resource that occurs after the PCR equal to the end time of the PCR.
4. Calculate the new end time of the resource.
5. Move to the next occurrence of the particular resource and set its start time equal to the end time of the previous occurrence of the resource.
6. Calculate the new end time of the resource.
7. Repeat the process until the end of the list has been reached.
8. Repeat the process for all the remaining resources in the list.

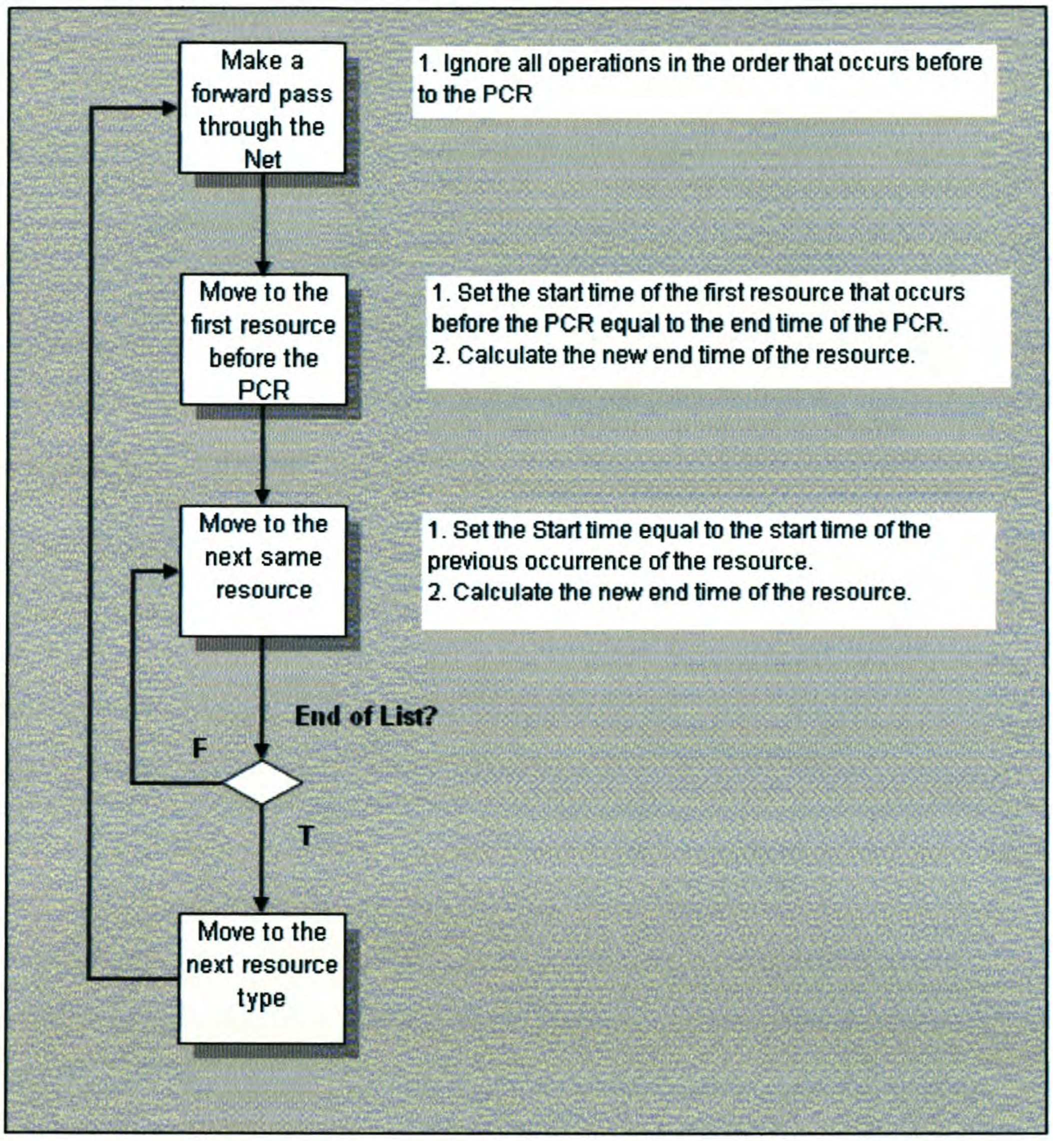

Make a forward pass through the Net
1. Ignore all operations in the order that occurs before to the PCR
Move to the first resource before the PCR
1. Set the start time of the first resource that occurs before the PCR equal to the end time of the PCR.
2. Calculate the new end time of the resource.
Move to the next same resource
1. Set the Start time equal to the start time of the previous occurrence of the resource.
2. Calculate the new end time of the resource.
End of List?
F
T
Move to the next resource type

**Figure 7.3:** Identifying the RL peaks for the secondary constraint

The secondary constraint is that resource that is the most heavily loaded. In other words it is that resource that is scheduled the furthest past time zero or is scheduled the furthest past the order due date.

If there is no secondary constraint the process can be terminated and the drum schedule can be implemented. Otherwise a schedule must be built for the additional constraint.

## 7.3 Building a Schedule for the Secondary Constraint

The schedules for the secondary constraint derive their priority from the buffer origin they are trying to protect. They can be required to feed the primary constraint, the shipping schedule, the assembly buffer or another secondary constraint.

The secondary constraint schedule must be supportive of the schedule built for the PCR. This is accomplished by the use of time rods, which are placed where necessary to ensure that sufficient buffer time is provided between the two constraint schedules. Rods will be inserted in three places:

1. Between the secondary and the primary constraint, where the secondary constraint feeds the primary constraint.
2. Between the secondary and the primary constraint, where the primary constraint feeds the secondary constraint.
3. As before batch rods must also be inserted to between multiple occurrences of the secondary constraint where the secondary constraint feeds itself.

The rod timing is computed in exactly the same way as explained in *Section 5.3.*

The algorithm and logic that was used to insert the rods for the primary constraint can be reused to insert the rods for the secondary constraint, as described in point 1 and 2 in the previous paragraph, by only making a few minor code changes.

The logic to insert the rods where the secondary constraint feeds the primary constraint is as follows and is illustrated graphically in *Figure 7.4*:

1. Make a backward pass through *The Net* and ignore all occurrences of the secondary constraint that are placed in front of the primary constraint.
2. Step through *The Net* until the first/next occurrence of the primary constraint in the order is found and save its start and end times in temporary variables.
3. Step further through *The Net* until the first occurrence of the secondary constraint is found.
4. Calculate the time gap between the primary and the secondary constraint and insert a rod if necessary. (Code Ref for Inserting Rods: Class *CIDConstraintDlg,* Procedure *InsertRods)*
5. Adjust start and end time of the operations that occur on the secondary constraint.
6. Repeat the steps 2, 3, 4 and 5 until the backward pass has been completed.

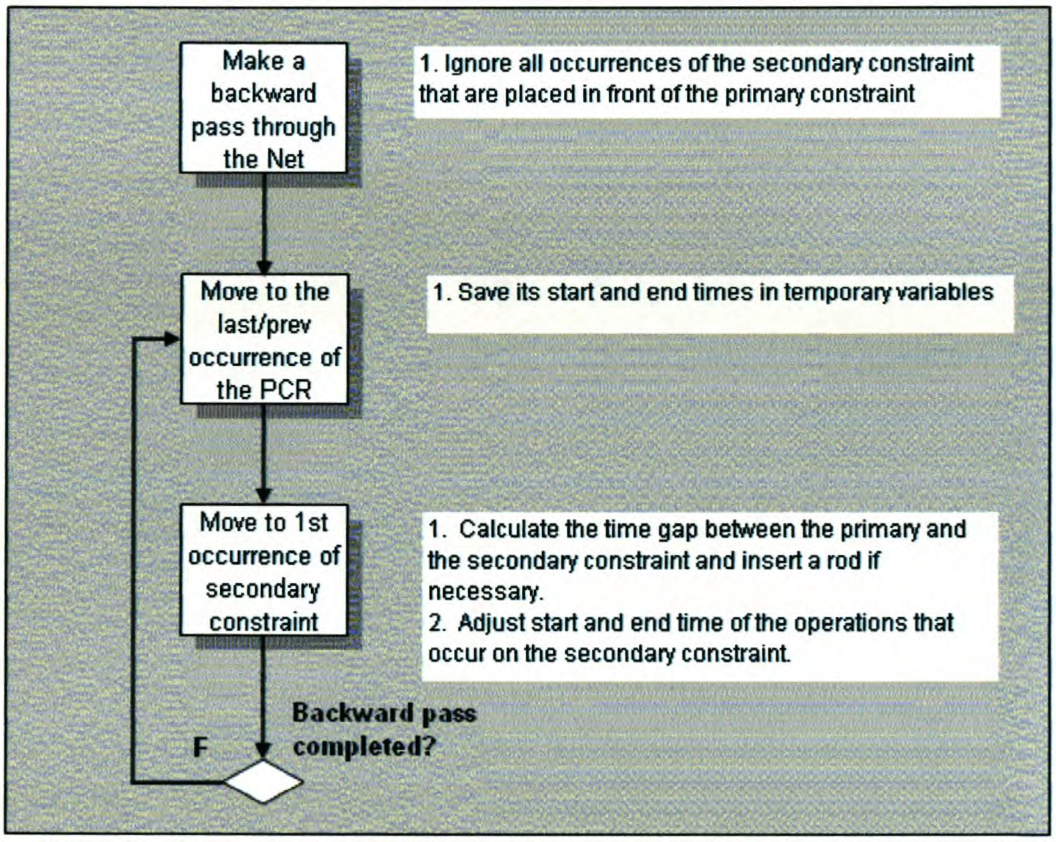

Make a backward pass through the Net
1. Ignore all occurrences of the secondary constraint that are placed in front of the primary constraint
Move to the last/prev occurrence of the PCR
1. Save its start and end times in temporary variables
Move to 1st occurrence of secondary constraint
1. Calculate the time gap between the primary and the secondary constraint and insert a rod if necessary.
2. Adjust start and end time of the operations that occur on the secondary constraint.
Backward pass completed?
F

**Figure 7.4:** Inserting rods where the secondary constraint feeds the primary constraint

The logic to insert the rods where the primary constraint feeds the secondary constraint is as follows and is illustrated graphically in *Figure 7.5*:

1. Make a forward pass through *The Net* and ignore all occurrences of the secondary constraint that are placed before the primary constraint.
2. Step through *The Net* until the last/previous occurrence of the primary constraint in the order is found and save its start and end times in temporary variables.
3. Step further through *The Net* until the first occurrence of the secondary constraint is found.
4. Calculate the time gap between the primary and the secondary constraint and insert a rod if necessary. (Code Ref for Inserting Rods: Class *CIDConstraintDlg,* Procedure *InsertRods)*
5. Adjust start and end time of the operations that occur on the secondary constraint
6. Repeat the steps 2, 3, 4 and 5 until the forward pass has been completed.

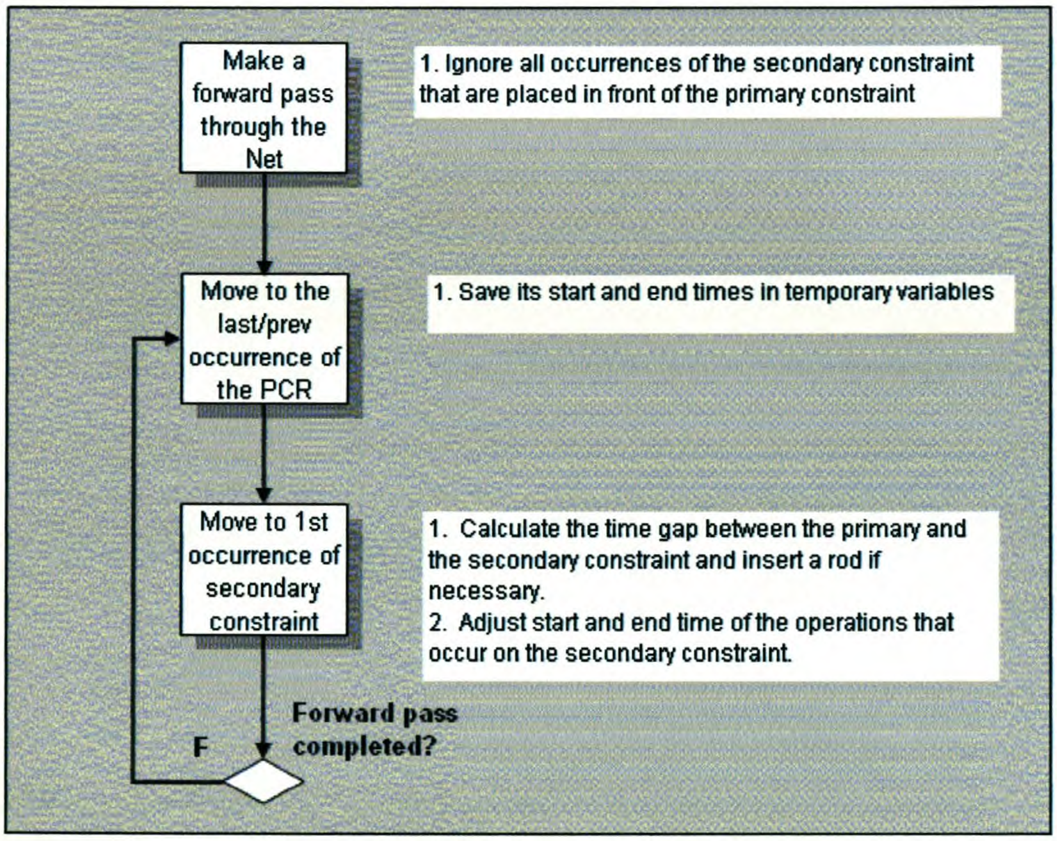

Make a forward pass through the Net
1. Ignore all occurrences of the secondary constraint that are placed in front of the primary constraint
Move to the last/prev occurrence of the PCR
1. Save its start and end times in temporary variables
Move to 1st occurrence of secondary constraint
1. Calculate the time gap between the primary and the secondary constraint and insert a rod if necessary.
2. Adjust start and end time of the operations that occur on the secondary constraint.
Forward pass completed?
F

**Figure 7.5:** Inserting rods where the primary constraint feeds the secondary constraint

The same algorithm used to insert rods for the PCR can be use to insert rods for the secondary constraint, if the situation occurs where the secondary constraint feeds itself.

The schedule created for the SC is infeasible whenever the time rods that were inserted are too restrictive to permit the generation of feasible schedule for the SC. If there are no infeasibilities the subordination process must be repeated to recheck for any other additional constraints.

One example of infeasibility is when the SC was placed between two occurrences of the PRC. With the algorithms implemented *Rod 1* would have been inserted between the SC and the PCR as shown in *Figure 7.2.* If the gap between operation 1 and operation 2 were not big enough the SC would have been placed to close to operation 1 and the schedule will then be infeasible.

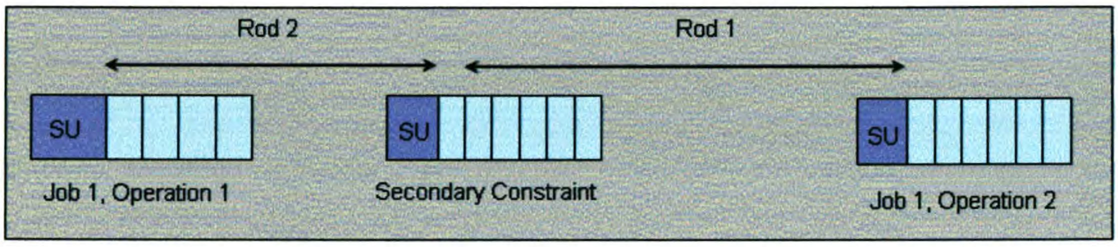


**Figure 7.6:** SC between 2 occurrences of the PCR

If this happens the primary constraint schedule must be adjusted. This is nothing more than the rebuilding of the primary schedule during which the constraint operations are shifted forward in the schedule by the amount of the difference of what delay between the operations should be and what the current delay is. After the adjustments the subordination process must be repeated.

The iteration of the subordination process could occur multiple times, depending on the impact of the violations on the ability of the other resources to support the created schedules. The end result is that the CCR schedules will be produced for one or more resources and all remaining resources will have been determined to have sufficient capacity during the scheduling horizon to support the CCR schedules.

# Chapter 8

## 8. *Conclusions and Recommendations*

This thesis was the first step in the completion of a larger project whose goal is to develop an affordable TOC scheduling package bases on Goldratt's Goal System algorithm for small to medium size businesses. The cost of this package and the complexity will be reduced by empowering the work force on the shop floor to make use of their experience to provide the TOC Scheduler with educated input and thereby increasing its scheduling accuracy. With regard to this thesis and the overall project goal the following have been achieved:

1. The complete project has been researched, scoped and each step has been explained.
2. The complete program structure has been defined and broken into two separate modules; the *Data Mining and Conversion Module* and the *TOC Scheduling Algorithm.*
3. The database containing all the MRP data necessary for scheduling has been designed and implemented using a MS Access database with an ODBC connection. An ODBC connection to the database was used so that a smooth transmission to other database management systems can be made.
4. The *TOC Scheduling Algorithm* has been developed and the following have been implemented:
   - A basic user interface has been created for the insertion of all the user input and to display the constraint schedule.
   - A data structure called a linked list has been developed and used to store the scheduling data in memory.
   - The GS algorithm has been and implemented and tested up to the point where it schedules the constraint.
   - The complete GS algorithm had been researched and explained.
   - The pseudo code for the part of the GS algorithm that was not implemented has been documented and included in this report.

At this point the most complex programming has been completed but to complete this project up to a point where it is a marketable product, the following work must still be done:

1. The subordination part of the GS algorithm must be coded as explained in the pseudo code in Chapter 7.
2. A user friendly GUI must be developed so that a person that has had only basic computer training can operate the system and easily make changes and be able test different scenarios without affecting the functioning of the program.
3. The most important part of the GUI will be a Gant chart like display of the schedule where each job is represented by a bar on the Gant Chart with a setup time, start time and an end time. The user must then be able to make changes to the schedule by making use of drag-and-dropping functionality so that he/she can move different jobs backwards and forwards in time, into different time slots and then update the schedule. This will make the schedule completely dynamic and the user is then empowered to use his/her experience on the shop floor to make educated changes to the schedule and thereby increasing accuracy.
4. Code and user interfaces must also be added so that the constraints can be exploited without having to change the database manually.

After completion of the above a TOC software package will exist that is completely unique in South Africa and the market potential for this package will be considerable.

# List of Abbreviations

| | |
|---|---|
| BOM | Bill of Materials |
| CCR | Capacity Constrained Resource |
| DFD | Data Flow Diagram |
| DMCM | Data Mining and Conversion Module |
| FDL | First Day Load |
| GS | Goal System |
| GUI | Graphical User Interface |
| MPS | Master Production Schedule |
| PCR | Primary Constrained Resource |
| RAM | Random Access Memory |
| RL | Red Lane |
| SC | Secondary Constraint |
| SDLC | Software Development Life Cycle |
| TOC | Theory of Constraints |
| TOCSM | Theory of Constrains Scheduling Module |

## References

Archer, Tom; Leinecker, Richard C. :The Visual C++ Bible". IDG Books Worldwide Inc., 1998

Budd, Timothy A.: "Classic Data Structures in C++", Addison-Wesley Publishing Company, 1994

Goldratt, E.M., Cox J.: "The Goal", Second Revised Edition, North River Press, Croton-on-Hudson, N.Y., 1992

Kolarik, William J.: "Creating Quality: Concepts, Systems, Strategies and Tools", McGraw-Hill International Editions, 1995

Hubbard, John R. Ph.D. "Programming with C++ second edition", McGraw-Hill International Editions, 1996

Moriguchi, Shigeichi,: "TQM Guide to Software Excellence", Productivity Press, Portland, Oregon, 1997

Stein, Robert E.: "Re-Engineering the Manufacturing System", Marcell Dekker Inc., N.Y. 1996

Simons, Jacob V.; Simpson, Wendell P.: "An explosion of Multiple constraind Scheduling as Implemented in the Goal System", Department of Management, Georgia South University, Production and Operarions Management Vol. 6 No.1 Spring 1997

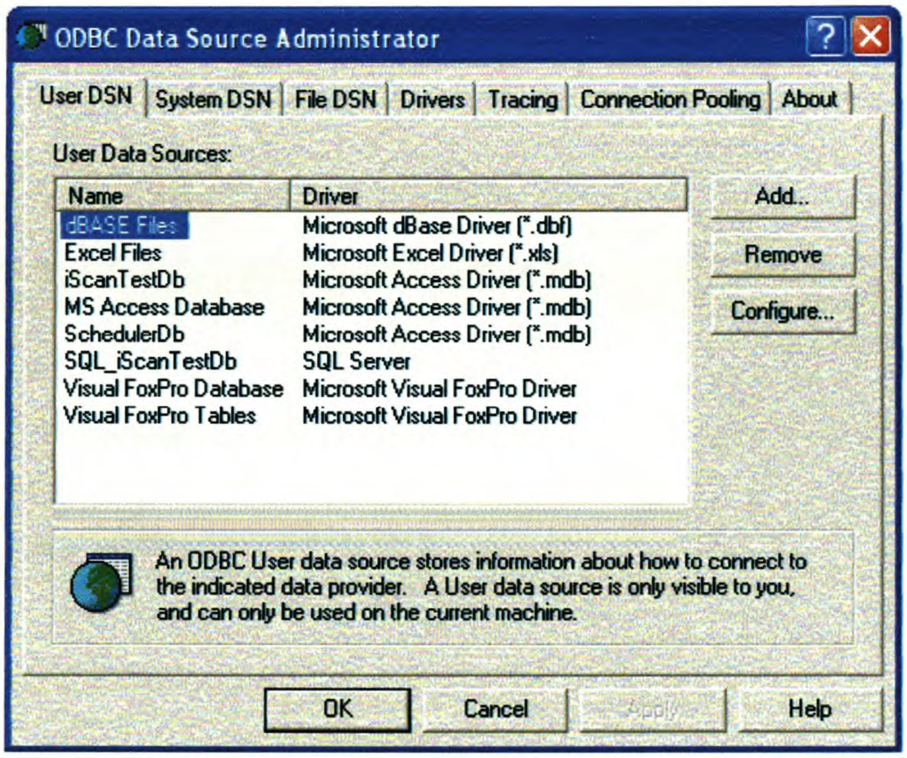
Wang, Paul S.: "C++ with Object Orientated Programming", PWS Publishing Company, Boston, 1994

# Appendix A

## *Program Setup*

To be able to run the scheduling program the following procedure must be followed:

1. Insert the cd provided, with this report, into your cd-rom drive.
2. Copy the folder, *TOC_Scheduler,* from the cd onto the root folder of your C drive. (Note: if the folder is copied to another location the program code will need to be edited so that the program will work)
3. An ODBC connection to the database must now be established. The first step is to go the *Windows Control Panel* and open the *Administrative Tools* window by double clicking on the *Administrative Tools* folder.
4. Open the *ODBC Data Source Administrator by* double clicking on the *Data Sources (ODBC)* icon.



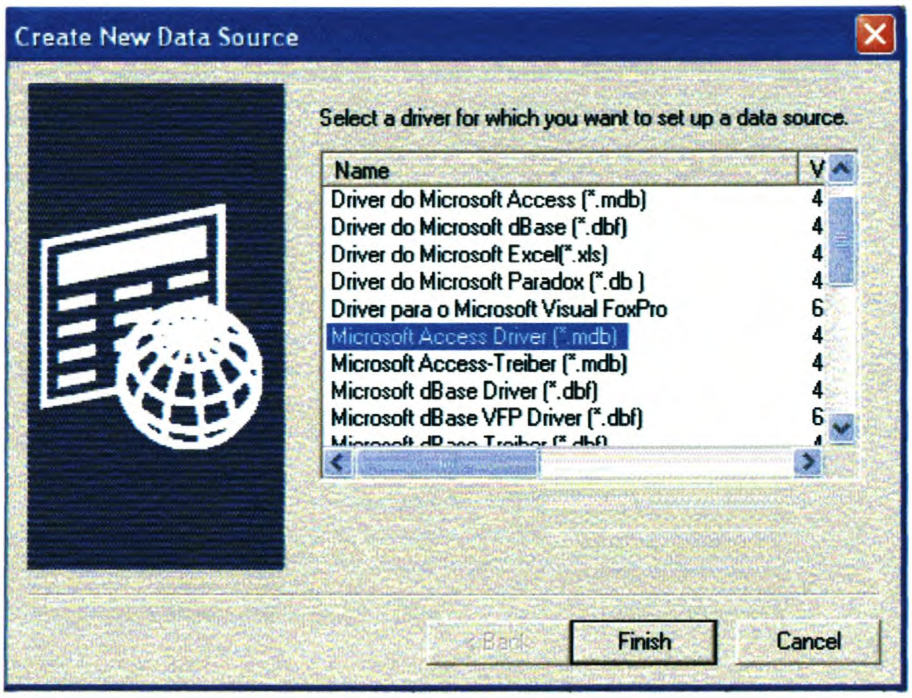
| Name | Driver |
|---|---|
| dBASE Files | Microsoft dBase Driver (*.dbf) |
| Excel Files | Microsoft Excel Driver (*.xls) |
| iScanTestDb | Microsoft Access Driver (*.mdb) |
| MS Access Database | Microsoft Access Driver (*.mdb) |
| SchedulerDb | Microsoft Access Driver (*.mdb) |
| SQL_iScanTestDb | SQL Server |
| Visual FoxPro Database | Microsoft Visual FoxPro Driver |
| Visual FoxPro Tables | Microsoft Visual FoxPro Driver |

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

**Figure A.1:** The ODBC Data Source Administrator window

5. Click on the *Add* button.

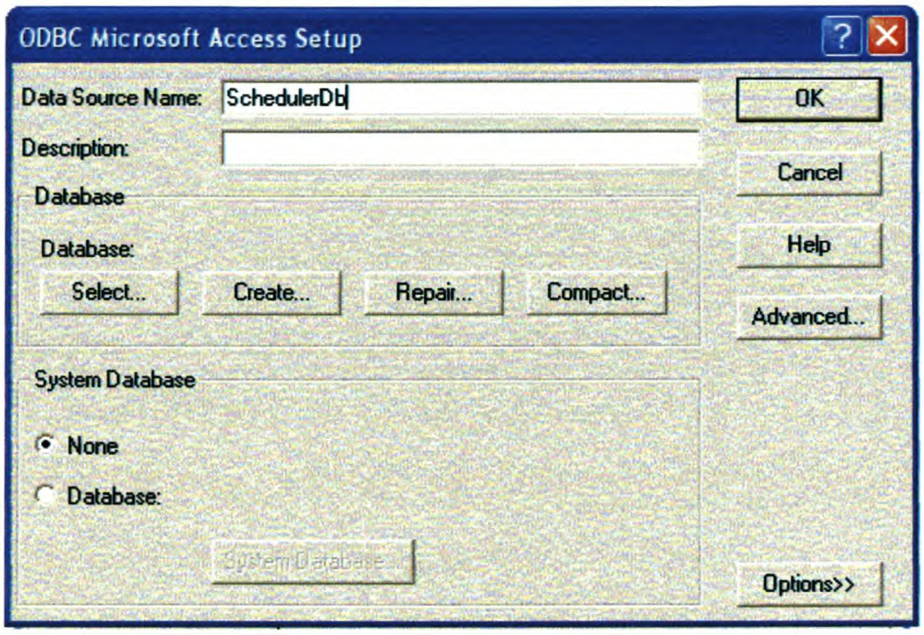
6. The *Create New Data Source* window now opens. Select the *Microsoft Access Driver(*.mdb)* option in the list box and click on the *Finish* button.



| Name | V |
|---|---|
| Driver do Microsoft Access (*.mdb) | 4 |
| Driver do Microsoft dBase (*.dbf) | 4 |
| Driver do Microsoft Excel(*.xls) | 4 |
| Driver do Microsoft Paradox (*.db ) | 4 |
| Driver para o Microsoft Visual FoxPro | 6 |
| Microsoft Access Driver (*.mdb) | 4 |
| Microsoft Access-Treiber (*.mdb) | 4 |
| Microsoft dBase Driver (*.dbf) | 4 |
| Microsoft dBase VFP Driver (*.dbf) | 6 |

**Figure A.2:** The Create New Data Source window

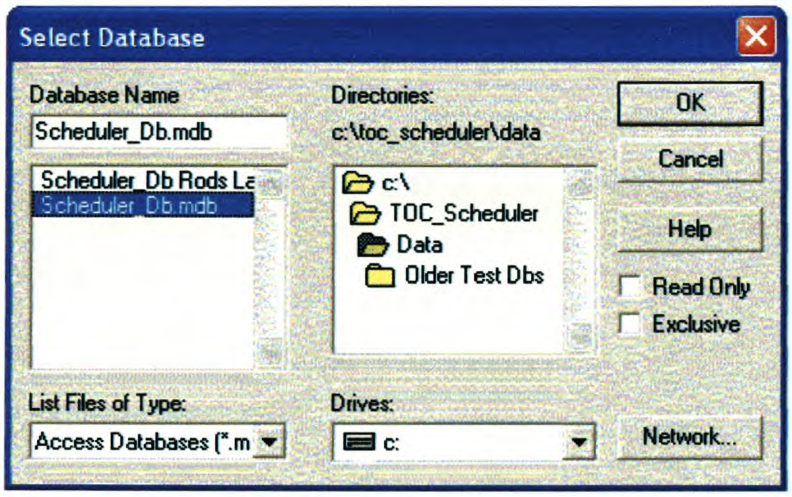
7. This opens the *ODCB Microsoft Access Setup* window. Type in 'SchedulerDb' into the *Data Source Name* edit box.



Data Source Name: SchedulerDb
Description:
Database:
System Database: (● None) Database:

**Figure A.3:** The ODBC Microsoft Access Setup window

8. Click on the *Select Button*. The *Select Database* window opens.



Database Name: Scheduler_Db.mdb
Scheduler_Db Rods La
Scheduler_Db.mdb
Directories: c:\toc_scheduler\data
c:\
TOC_Scheduler
Data
Older Test Dbs
List Files of Type: Access Databases (*.m
Drives: c:

**Figure A.4:** The Select database window

9. Browse to 'C:\TOC_Scheduler\Data and select Scheduler_Db.mdb file. Click on the *OK* button.
10. Click on *OK* twice more.

11. Browse to C:\TOC_Scheduler and click on the TOC_Scheduler.exe file. This will start the application.

## Viewing the Code

To View the programming code do the following:

1. Install Visual Studio 6.0 with Visual C++ on your system.
2. Browse to C:\TOC_Scheduler and click on the TOC_Scheduler.dsw file. This will open the application in Visual C++ and you will be able to browse through the programming code.

# Appendix B

```
// PlanningHorison.cpp : implementation file
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "PlanningHorison.h"
#include "CalendarDataSet.h"
#include "OrderSet.h"
#include "BufferLength.h"
#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

extern CTime glbStartDate;
extern COrders* pOrder;
extern COrders* OrderEnd;
extern COrders* OrderStart;                                 //Keeps address of last order in l
ist for termination
extern glbSegrLen;
extern float glbConstBuff;
extern CTime glbFirstDate;                                  //To get the first date in the ord
er
extern CTime glbLastDate;                                   //To get the last date in the orde
r

int glbDDateDay;
int glbDDateHrs;
int glbStdDay;                                             //The standard day length in hours

int glbSegr;
float glbShipBuff;
bool Flag;                                                  //To check that an end data has be
en selected
bool OrderFlag;                                             //To check if there are any orders
 in the planning horison

/////////////////////////////////////////////////////////////////////////////
// CPlanningHorison property page

IMPLEMENT_DYNCREATE(CPlanningHorison, CPropertyPage)

CPlanningHorison::CPlanningHorison() : CPropertyPage(CPlanningHorison::IDD)
{
    //{{AFX_DATA_INIT(CPlanningHorison)
    m_NrSchedulingDays = 0;
    m_ctStartDate = 0;
    m_ctEndDate = 0;
    //}}AFX_DATA_INIT
}

CPlanningHorison::~CPlanningHorison()
{
}

void CPlanningHorison::DoDataExchange(CDataExchange* pDX)
{
    CPropertyPage::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CPlanningHorison)
    DDX_Control(pDX, IDC_LIST4, m_wdc);
    DDX_Control(pDX, IDC_SCH_DAYS, m_edtShowHide);
    DDX_Text(pDX, IDC_SCH_DAYS, m_NrSchedulingDays);
    DDX_DateTimeCtrl(pDX, IDC_DTP_START_DATE, m_ctStartDate);
    DDX_DateTimeCtrl(pDX, IDC_DTP_END_DATE, m_ctEndDate);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPlanningHorison, CPropertyPage)
    //{{AFX_MSG_MAP(CPlanningHorison)
    ON_NOTIFY(DTN_CLOSEUP, IDC_DTP_END_DATE, OnCloseupDtpEndDate)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
```

```
// CPlanningHorison message handlers

//This procedure calculates the due date for each order and stores it in a linked list, COrder

void CPlanningHorison::GetWorkDayCount()
{
  CDatabase db;
  CCalendarDataSet* pCalendarDataSet = NULL;
  COrderSet* pOrderSet = NULL;
  COrders* temp = NULL;
  int WorkdayCounter = 1;
  OrderFlag = FALSE;                                    //To get the last date in the order
  bool FirstOrderFlag = FALSE;                              //To get the address of the first order
  CTime TempDate;
  bool FirstDate = FALSE;
  CString Date;



  GetBufferData();

  try{
       if (db.Open("SchedulerDb"))
       {
           pCalendarDataSet = new CCalendarDataSet();                    //created a recordset
so that the calendar table can be accessed
           pCalendarDataSet->Open();

           pOrderSet = new COrderSet();
           pOrderSet->Open();

           pOrder = new COrders();
           COrders* temp = new COrders();
           temp = pOrder;


           pOrder->strOrderID = "Dummy";
           pOrder->WorkdayCounter = 0;

           while (pCalendarDataSet->m_Date <= m_ctEndDate)               //starts at the beginn
ing of the calendar table and moves
           {                                                             //on untill the last d
ate of the order

                //counts all the workdays starting from the earliest order
                if ((pCalendarDataSet->m_Date >= m_ctStartDate) && (pCalendarDataSet->m_IsWorkDa
y == "Yes")){
                      WorkdayCounter++;
                }

                //if the end date of the due date of the order is reached the order detail is sa
ved in the list in memory
                while ((!pOrderSet->IsEOF()) && (pCalendarDataSet->m_Date == pOrderSet->m_DueDat
e)) {
                      char buffer[10] = "";
                      char string[80] ="";

                      temp->next = new COrders();

                      temp->next->strOrderID = pOrderSet->m_OrderID;

                      // to get the adress space of the first order to be used in procedure Inser
tSetupTimes
                      if (FirstOrderFlag == FALSE) {
                          OrderStart = temp->next;
                          FirstOrderFlag = TRUE;
                      }

                      temp->next->WorkdayCounter = WorkdayCounter;
                      temp->next->ctDueDate = pOrderSet->m_DueDate;
                      temp->next->OrderPlaced = pOrderSet->m_OrderPlaced;


                      temp->next->DDateHr = (WorkdayCounter * glbStdDay) - glbShipBuff;         //
determine the hours available to complete the order

                      strcpy( string, pOrderSet->m_OrderID );
                      strcat( string, ":       " );
```

```
                    gcvt(temp->next->DDateHr,3, buffer);
                    strcat( string, buffer );
                    strcat( string, "   Hours or   " );
                    itoa(temp->next->WorkdayCounter, buffer,10);
                    strcat( string, buffer );
                    strcat( string, "   Work Days" );
                    m_wdc.AddString(string);


                //  m_lstTestDD.AddString(buffer);

                    //Check to see if orders were selected
                    if ((WorkdayCounter > 1) && (FirstDate == FALSE)) {
                         glbFirstDate = pOrderSet->m_DueDate;
                         FirstDate = TRUE;                                      //set
s flag for error message
                    }

                    glbLastDate = pOrderSet->m_DueDate;                        //for
use in CIDConstraint
                    pOrderSet->MoveNext();

                    temp = temp->next;

                    OrderFlag = TRUE;                                          //Flag
 for error mesage
                    TempDate = pOrderSet->m_DueDate;
                }

            pCalendarDataSet->MoveNext();
           }

           OrderEnd = temp;                                                   //Keeps
 memory location of last order in list
           temp->next = new COrders();
           temp->next->strOrderID = "EOF";

           pCalendarDataSet->Close();                                        //Closes
 recordset
           pOrderSet->Close();
           delete pOrderSet;
           delete pCalendarDataSet;                                          //delete
s recordset, all info is stored in lists now
           db.Close();

           }
       Flag = TRUE;                  //end date has been selected

       if (TempDate < m_ctStartDate) {
           OrderFlag = FALSE;
       }
    }

   //error handling for database errors
   catch(CDBException* pe)
   {
        AfxMessageBox(pe->m_strError);

        if (pCalendarDataSet)
        {
            if (pCalendarDataSet->IsOpen())
            {
                pCalendarDataSet->Close();
            }
        delete pCalendarDataSet;
        }
        if (db.IsOpen())
        {
            db.Close();
        }
        pe->Delete();
    }

    m_NrSchedulingDays = WorkdayCounter;
    UpdateData(FALSE);
}
```

```
//Check is dates selected are valid
void CPlanningHorison::OnCloseupDtpEndDate(NMHDR* pNMHDR, LRESULT* pResult)
{
    UpdateData(TRUE);

    if (m_ctEndDate <= m_ctStartDate)
        AfxMessageBox("ERROR! The End Date of the scheduling horison must be at a later date th
an the Start Date."
                     " Please select correct dates");
    else {
       GetWorkDayCount();
    }

    *pResult = 0;
}

BOOL CPlanningHorison::OnInitDialog()
{
    CTime CurrentDate;

    CurrentDate = CTime::GetCurrentTime();
    m_ctEndDate = CurrentDate;                  //initialises date picker controls to the curren
t date
    m_ctStartDate = CurrentDate;
    Flag = FALSE;                              //Initialises flag
    UpdateData(FALSE);
    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}


//Retrieves the buffer data from file for use in this class
void CPlanningHorison::GetBufferData()
{
  div_t div_result;
  int numclosed;
  FILE *stream;

    //stream = fopen( "./Data/ParameterData.txt", "r" );
  stream = fopen( "E:/My Documents/M TESIS/Tesis/Working Version/TOC_Scheduler/Data/ParameterDa
ta.txt", "r" );
    if( stream == NULL )
      AfxMessageBox( "The file fscanf.out was not opened" );
    else
    {
      fscanf(stream, "%ld", &glbStdDay);
      fscanf(stream, "%ld", &glbSegr);
      fscanf(stream, "%f", &glbShipBuff);
      fscanf(stream, "%f", &glbConstBuff);
    }

     fclose(stream);
    //numclosed = _fcloseall( );


    glbSegrLen = glbStdDay/glbSegr;

    div_result = div(glbShipBuff, glbSegrLen);
    glbDDateDay = div_result.quot;
    glbDDateHrs = div_result.rem;

    UpdateData(FALSE);
}


//Displays error messages if invalid dates have been selected
void CPlanningHorison::OnOK()
{
    if (Flag == FALSE) {
       AfxMessageBox("ERROR! Please Select an End Date");
       CMainFrame().OnScheduleParameters();

     }
    else if(OrderFlag == FALSE){
        AfxMessageBox("ERROR! No Orders in Planning Horison");
       CMainFrame().OnScheduleParameters();
       }
```

```
// BufferLength.cpp : implementation file
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "BufferLength.h"
#include "io.h"
#include "Parameters1.h"
#include "NetDataSet1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CBufferLength property page

IMPLEMENT_DYNCREATE(CBufferLength, CPropertyPage)

CBufferLength::CBufferLength() : CPropertyPage(CBufferLength::IDD)
{
	//{{AFX_DATA_INIT(CBufferLength)
	m_edtAssemblyDef = 0;
	m_ConstraintDef = 0;
	m_edtShippingDef = 0;
	m_edtAssembly = 0;
	m_edtShipping = 0;
	m_edtConstraint = 0;
	m_intSegrDef = 0;
	m_intStdDay = 0;
	m_intSegr = 0;
	m_intStdDayDef = 0;
	//}}AFX_DATA_INIT
}

CBufferLength::~CBufferLength()
{
}

void CBufferLength::DoDataExchange(CDataExchange* pDX)
{
	CPropertyPage::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CBufferLength)
	DDX_Control(pDX, IDC_APPLY, m_btnApply);
	DDX_Text(pDX, IDC_ASS_DEF, m_edtAssemblyDef);
	DDX_Text(pDX, IDC_CON_DEF, m_ConstraintDef);
	DDX_Text(pDX, IDC_SHIP_DEF, m_edtShippingDef);
	DDX_Text(pDX, IDC_ASSEMBLY, m_edtAssembly);
	DDX_Text(pDX, IDC_SHIPPING, m_edtShipping);
	DDX_Text(pDX, IDC_CONSTRAINT, m_edtConstraint);
	DDX_Text(pDX, IDC_SEGR_DEF, m_intSegrDef);
	DDX_Text(pDX, IDC_STDDAY, m_intStdDay);
	DDX_Text(pDX, IDC_SEGR, m_intSegr);
	DDX_Text(pDX, IDC_STDDAY_DEF, m_intStdDayDef);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CBufferLength, CPropertyPage)
	//{{AFX_MSG_MAP(CBufferLength)
	ON_BN_CLICKED(IDC_APPLY, OnApplyShip)
	ON_EN_UPDATE(IDC_SHIPPING, OnUpdateBuffer)
	ON_EN_UPDATE(IDC_CONSTRAINT, OnUpdateBuffer)
	ON_EN_UPDATE(IDC_ASSEMBLY, OnUpdateBuffer)
	ON_EN_UPDATE(IDC_SEGR, OnUpdateBuffer)
	ON_EN_UPDATE(IDC_STDDAY, OnUpdateBuffer)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CBufferLength message handlers


//this procedute save the new input variables
void CBufferLength::SaveBuffers()
{
    if (m_edtShipping == 0) {
         m_edtShipping = m_edtShippingDef;
    }
    if (m_edtConstraint == 0){
         m_edtConstraint = m_ConstraintDef;
    }
    if (m_edtAssembly == 0){
        m_edtAssembly = m_edtAssemblyDef;
    }
    if (m_intSegr == 0){
        m_intSegr = m_intSegrDef;
    }
    if (m_intStdDay == 0){
        m_intStdDay = m_intStdDayDef;
    }


    FILE *stream;
    stream = fopen( "E:/My Documents/M TESIS/Tesis/Working Version/TOC_Scheduler/Data/Parameter
Data.txt", "w");
    if( stream == NULL )
      AfxMessageBox( "The file fscanf.out was not opened\n" );
    else
    {
      fprintf( stream, "%ld %ld %f %f %f", m_intStdDay, m_intSegr, m_edtShipping,
               m_edtConstraint, m_edtAssembly);
    }
    fclose( stream );

    UpdateData(FALSE);
    CBufferLength::OnInitDialog();

}

//this procedure verifies that the new input is correct and if so calls the save procedure
void CBufferLength::OnApplyShip()
{
    UpdateData(TRUE);
    div_t div_result;
    bool Flag = FALSE;

    //the smallest segregation is 1 hour, this statement scheck whether the segregations is an
integer value
    if ((m_intStdDay != 0) && (m_intSegr != 0)){
        div_result = div( m_intStdDay, m_intSegr);
        Flag = TRUE;
    }

    //various error messages for incorrect input
    if (((m_intStdDay == 0) && (m_intSegr != 0)) || ((m_intStdDay != 0) && (m_intSegr == 0))){
        AfxMessageBox("ERROR! Please enter a value for Day Length and Nr. of Segregations");
    }
    else if ((div_result.rem  != 0) && (Flag == TRUE)) {
        AfxMessageBox("ERROR! Please change Day Length or Nr. of Segregation values so "
        " that the Segregation length is to the nearest hour");
    }
    else if ((m_edtAssembly != 0) || (m_edtConstraint != 0) || (m_edtShipping != 0) || (m_intSt
dDay !=0) || (m_intSegr != 0))
    {
        SaveBuffers();
    }

    m_btnApply.EnableWindow(FALSE);
}

void CBufferLength::OnUpdateBuffer()
{
    m_btnApply.EnableWindow(TRUE);

}

//Retrieves the input variables stored in file and displays them in buffer length dialog
BOOL CBufferLength::OnInitDialog()
{
    CPropertyPage::OnInitDialog();

    FILE *stream;
    stream = fopen( "./Data/ParameterData.txt", "r" );                        //openes the file for r
eading
    if( stream == NULL )
      printf( "The file fscanf.out was not opened\n" );
    else
    {
      fscanf( stream, "%ld", &m_intStdDayDef);
      fscanf( stream, "%ld", &m_intSegrDef);
      fscanf( stream, "%f", &m_edtShippingDef);
      fscanf( stream, "%f", &m_ConstraintDef);
      fscanf( stream, "%f", &m_edtAssemblyDef);

    }
    fclose( stream );

    m_edtShipping = 0;
    m_edtAssembly = 0;
    m_edtConstraint = 0;
    m_intStdDay = 0;
    m_intSegr = 0;
    UpdateData(FALSE);


    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}
```

```
// NetDataSet1.cpp : implementation file
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "NetDataSet1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CNetDataSet

IMPLEMENT_DYNAMIC(CNetDataSet, CRecordset)

CNetDataSet::CNetDataSet(CDatabase* pdb)
    : CRecordset(pdb)
{
    //{{AFX_FIELD_INIT(CNetDataSet)
    m_From_Part_Op = _T("");
    m_Qty = 0;
    m_ResourceID = _T("");
    m_RunTime = 0.0f;
    m_SetupTime = 0.0f;
    m_To_Part_Op = _T("");
    m_nFields = 7;
    //}}AFX_FIELD_INIT
    m_nDefaultType = snapshot;
}


CString CNetDataSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=SchedulerDb");
}

CString CNetDataSet::GetDefaultSQL()
{
    return _T("[NetData]");
}

void CNetDataSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(CNetDataSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[From_Part/Op]"), m_From_Part_Op);
    RFX_Int(pFX, _T("[Qty]"), m_Qty);
    RFX_Date(pFX, _T("[Due_Date]"), m_DueDate);
    RFX_Text(pFX, _T("[ResourceID]"), m_ResourceID);
    RFX_Single(pFX, _T("[RunTime]"), m_RunTime);
    RFX_Single(pFX, _T("[SetupTime]"), m_SetupTime);
    RFX_Text(pFX, _T("[To_Part/Op]"), m_To_Part_Op);
    //}}AFX_FIELD_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CNetDataSet diagnostics

#ifdef _DEBUG
void CNetDataSet::AssertValid() const
{
    CRecordset::AssertValid();
}

void CNetDataSet::Dump(CDumpContext& dc) const
{
    CRecordset::Dump(dc);
}
#endif //_DEBUG
```

```
// IDConstraintDlg.cpp : implementation file
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "IDConstraintDlg.h"
#include "NetDataSet1.h"
#include "ResourceSet.h"
#include "SchedulerDoc.h"
#include "SchedulerView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


COrders* pOrder;
COrders* OrderEnd;
COrders* OrderStart;                //Global variable from class CPlanningHorison
CResourceData* pResourceData;
int glbSegrLen;                   //Global variable from class CPlanningHorison
float glbConstBuff;                 //Global variable from class CPlanningHorison
int glbNrOfResources = 0;
CTime glbFirstDate;
CTime glbLastDate;
CString CCR;                       //Capacity constrained resource

/////////////////////////////////////////////////////////////////////////////
// CIDConstraintDlg dialog


CIDConstraintDlg::CIDConstraintDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CIDConstraintDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CIDConstraintDlg)
    m_CCR = _T("");
    //}}AFX_DATA_INIT
}


void CIDConstraintDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CIDConstraintDlg)
    DDX_Control(pDX, IDC_LSTCTRL, m_lstCtrl);
    DDX_Text(pDX, IDC_CCR, m_CCR);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CIDConstraintDlg, CDialog)
    //{{AFX_MSG_MAP(CIDConstraintDlg)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CIDConstraintDlg message handlers

void CIDConstraintDlg::RetrieveNetData()
{
  CNet* End = NULL;
  CNet* Back = NULL;
  COrders* OrderEnd = NULL;
  COrders* OrderBack = NULL;


  CNetDataSet* pNetDataSet = NULL;
  CDatabase db;

  CString tempToPartOp;
  CString FirstOrder;
  CTime OrderListDueDate;
  int count=0;
  try
     {
       if (db.Open("SchedulerDb"))
```

```
        {
            pNetDataSet = new CNetDataSet();
            pNetDataSet->Open();

            CNet* pNet = new CNet();
            CNet* NetList = new CNet();
            NetList = pNet;

            COrders* OrderList; // = new COrders();
            OrderList = pOrder->next;

             //Needed to terminate while in procedure IdentifyFDLPeaks
            FirstOrder = pOrder->next->strOrderID;


            while ((!pNetDataSet->IsEOF()) && (pNetDataSet->m_DueDate <= glbLastDate))
            {

               //Enter Table NetData into linked list in memory
                //if ((pNetDataSet->m_DueDate != OrderList->ctDueDate) && (OrderList!= OrderEnd)
){

               pNetDataSet->MovePrev();
               tempToPartOp = pNetDataSet->m_To_Part_Op;       //Placeholder
               pNetDataSet->MoveNext();

               //statement to make sure orderlist moves forward in the correct place
               if ((tempToPartOp == OrderList->strOrderID) && (OrderList!= OrderEnd)){
                    OrderBack = OrderList;                          //Stores pointers so that b
ackward pass is possible
                    OrderList = OrderList->next;
                    OrderList->prev = OrderBack;
                    //End = NetList;

              //    AfxMessageBox("hello");
                  }

                 if ((pNetDataSet->m_DueDate >= glbFirstDate) && (pNetDataSet->m_DueDate <= glbLa
stDate)) {
                    NetList->next = new CNet();
//                  NetList->next->strAltResource = pNetDataSet->m_Alt_Resource;
//                  NetList->next->fltAltSetupTime = pNetDataSet->m_Alt_RunTime;
//                  NetList->next->fltSetupTime = pNetDataSet->m_Alt_SetupTime;
                    NetList->next->strFromPartOp = pNetDataSet->m_From_Part_Op;
                    NetList->next->intQty = pNetDataSet->m_Qty;
                    NetList->next->strResourceID = pNetDataSet->m_ResourceID;
//                  NetList->next->fltRework = pNetDataSet->m_Rework;
//                  NetList->next->strReworkPartOp = pNetDataSet->m_Rework_Part_Op;
                    NetList->next->fltRunTime = pNetDataSet->m_RunTime;
//                  NetList->next->fltScrap = pNetDataSet->m_Scrap;
                    NetList->next->fltSetupTime = pNetDataSet->m_SetupTime;
                    NetList->next->strToPartOp = pNetDataSet->m_To_Part_Op;
                    NetList->next->ctDueDate = pNetDataSet->m_DueDate;

                    //Dislays the due date of each order and retrieves end date
                    NetList->next->EndTime = OrderList->DDateHr;

                    //Calculations to determine the total time for each operation and the start
time of each operation

                    //Only  includes the setup time for the first occurence of the same orders o
nce
                    //This is only for the purpose of the identification of the CCR
                    if (OrderList->OrderPlaced == TRUE) {
                      //  AfxMessageBox("True");
                        NetList->next->TotalTime = (pNetDataSet->m_Qty * pNetDataSet->m_RunTime)
 + pNetDataSet->m_SetupTime;
                        NetList->next->StartTime = OrderList->DDateHr - NetList->next->TotalTime
;
                    }
                    else {
                     //   AfxMessageBox("False");
                        NetList->next->TotalTime = (pNetDataSet->m_Qty * pNetDataSet->m_RunTime)
;

                        NetList->next->StartTime = OrderList->DDateHr - NetList->next->TotalTime
;
                    }
                    Back = NetList;                                     //Stores pointers so that backward
pass is possible
                    NetList = NetList->next;
                    NetList->prev = Back;
                    End = NetList;
                }
                pNetDataSet->MoveNext();
                count++;
            }
          //  Posintion of end of list
          // End = NetList;
           pNetDataSet->Close();
           delete pNetDataSet;
           db.Close();

           NrOfResources();                                    //retrieves Resourcse data and calcu
lates nr of resources
           IdentifyFDLPeaks(pNet, End, FirstOrder);            //Identifies the FDL peaks and the c
onstrained resource
      //    ScheduleBackwards(pNet, End);
      //    UpdateTableSchedule(pNet, End);

           DisplayNetData(pNet, End);


  }
}
    catch(CDBException* pe)
    {
        AfxMessageBox(pe->m_strError);

        if (pNetDataSet)
        {
            if (pNetDataSet->IsOpen())
            {
                pNetDataSet->Close();
            }
        delete pNetDataSet;
        }
        if (db.IsOpen())
        {
            db.Close();
        }
        pe->Delete();
    }
    UpdateData(FALSE);
}

BOOL CIDConstraintDlg::OnInitDialog()
{

    CDialog::OnInitDialog();
    RetrieveNetData();                                              //Retrieves Net Data from Data
base
    UpdateData(FALSE);

    return TRUE;  // return TRUE unless you set the focus to a control
                  // EXCEPTION: OCX Property Pages should return FALSE
}

//This precedure identify the CCR by leveling the loads on each resource and
//checking which resource must perform the most work in the past
void CIDConstraintDlg::IdentifyFDLPeaks(CNet* pNet, CNet* End, CString FirstOrder)
{
  CNet* NetList;
  CResourceData* ResourceList;
  ResourceList = pResourceData;
  NetList = End;                                                          //Starts at end of lis
t and works backward
  float EndTimeTracker = NetList->EndTime;
  float tempEndTime = NetList->EndTime;

  for (int i = 1; i <= glbNrOfResources; i++) {                           //Loops once through t
he list for each resource
      NetList = End;
      tempEndTime = EndTimeTracker;
      ResourceList = ResourceList->next;                                  //Starts with the firs
```

```
t resource in the list                                                          //and compares with th
e resrouce names in the net

      while (pNet != NetList) {
    //    NetList = NetList->prev;
          if (ResourceList->strResourceID == NetList->strResourceID) {  //loops thorugh the net
 backward and levels the load
              if (tempEndTime > NetList->EndTime) {
                  tempEndTime = NetList->EndTime;
              }
               else {
                  NetList->EndTime = tempEndTime;
              }

            NetList->StartTime = tempEndTime - NetList->TotalTime;
            tempEndTime = NetList->StartTime;
          }
        NetList = NetList->prev;
    }
  }

   NetList = pNet;
   float temp = 1000000;


//Loop to determine FDL peak by checking which resource has the earliest negative start time
   //while (NetList->strToPartOp != FirstOrder) {
   while (End != NetList) {
      NetList = NetList->next;
      if (NetList->StartTime < temp) {
          temp = NetList->StartTime;
          CCR = NetList->strResourceID;
      }
    }

   if (temp > 0) {
       CCR = "No CCR";
   }
   InsertSetupTimes(pNet,End);                                   //Procedure that inserts the
setup times that were left out
                                                                     //in the ID of the CCR

   if (temp < 0){
      m_CCR = CCR;

      InsertRods(pNet, End);
     // float NegativeTime = GetCCREarliestTime(pNet, End, CCR);
      MessageBox("The Primary Constraint is: " + CCR, "Order Placed", MB_OK);

   }
   else {
       MessageBox("There is no Primary Constraint! The Orders will be backwards Scheduled.", "O
rder Placed", MB_OK);
       m_CCR = "No CCR";
       ScheduleBackwards(pNet, End);
   }

   glbNrOfResources = 0;                                              //Reinitialises variable
 if new planning horison is chosen
   UpdateData(FALSE);


}

void CIDConstraintDlg::NrOfResources()
{
    CDatabase db;
    CResourceSet* pResourceSet = NULL;                           //Set from database
    pResourceData = new CResourceData();
    CResourceData*  ResourceList = NULL;                         //list in memory
    ResourceList = pResourceData;                                //stores first pointer for re
ference
    try {

       if (db.Open("SchedulerDb")) {
          pResourceSet = new CResourceSet();
          pResourceSet->Open();
          while (!pResourceSet->IsEOF()) {
               ResourceList->next = new CResourceData();
               ResourceList->next->strResourceID = pResourceSet->m_ResourceID;
               CString temp = ResourceList->next->strResourceID;
               ResourceList->next->intResourceQty = pResourceSet->m_Resource_Qty;
               glbNrOfResources = glbNrOfResources + pResourceSet->m_Resource_Qty;

               pResourceSet->MoveNext();
               ResourceList = ResourceList->next;
           }

           pResourceSet->Close();
           delete pResourceSet;
           db.Close();
       }
     }
    catch(CDBException* pe)
    {
        AfxMessageBox(pe->m_strError);

        if (pResourceSet)
        {
            if (pResourceSet->IsOpen())
            {
                pResourceSet->Close();
            }
        delete pResourceSet;
        }
        if (db.IsOpen())
        {
            db.Close();
        }
        pe->Delete();
    }
    UpdateData(FALSE);

}

void CIDConstraintDlg::DisplayNetData(CNet* pNet, CNet* End)
{
  CNet* NetList;
  NetList = pNet;
  char buffer[10] = "";
  char string[15] = "";
  CString Date;
  int count = 0;
  int iIndex = 0;
  char underline[20] = "__________";
  InitListControl();


  while (End != NetList) {
      NetList = NetList->next;

      if (NetList->strResourceID == CCR) {

       Date.Format(_T("%02d/%02d/%2d"),NetList->ctDueDate.GetMonth(),NetList->ctDueDate.GetDay(
),NetList->ctDueDate.GetYear());
       strcpy( string, "| " );
       strcat( string, Date );
       strcat( string, " |" );
       m_lstCtrl.InsertItem(iIndex, string, 0);

       strcpy( string, "| " );
       strcat( string, NetList->strToPartOp );
       strcat( string, " |" );
       m_lstCtrl.SetItemText(iIndex, 1, string);

       strcpy( string, "| " );
       strcat( string, NetList->strFromPartOp );
       strcat( string, " |" );
       m_lstCtrl.SetItemText(iIndex, 2, string);

       char buffer[10] = "";
       char string[15] = "";
       itoa( NetList->intQty, buffer, 10 );
       strcpy( string, "| " );
       strcat( string, buffer );
```

```
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 3, string );

        strcpy( string, "| " );
        strcat( string, NetList->strResourceID );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 4, string );

        gcvt(NetList->fltSetupTime,6, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 5, string );

        gcvt(NetList->fltRunTime,5, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 6, string );

        gcvt((NetList->TotalTime),6, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 7, string );

        gcvt(NetList->StartTime, 6, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 8, string );

        gcvt( NetList->EndTime, 6, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 9, string );
                                                                                  //Converts time
s into Time Bucket format
        int tempST = floor(NetList->StartTime);                                   //Uses integer
declaration to drop the fraction
        div_t div_result;

        div_result = div( tempST, glbSegrLen );                                  //Calculated the
 nr of time buckets
        float StartHr = (NetList->StartTime - tempST) + div_result.rem;          //Calculates tim
e bucket start hour
        int StartTimeBucket = div_result.quot;

        gcvt( StartTimeBucket, 6, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 10, string );

        gcvt( StartHr, 6, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 11, string );

        int tempET = floor(NetList->EndTime);

        div_result = div( tempET, glbSegrLen );                                  //Calculated the
 nr of time buckets
        float EndHr = (NetList->EndTime - tempET) + div_result.rem;              //Calculates tim
e bucket end hour
        int EndTimeBucket = div_result.quot;

        div_result = div( tempET, glbSegrLen );                                  //Calculates end
 time in time bucket format
        gcvt( div_result.quot, 6, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 12, string );

        gcvt( div_result.rem, 6, buffer);
        strcpy( string, "| " );
        strcat( string, buffer );
        strcat( string, " |" );
        m_lstCtrl.SetItemText(iIndex, 13, string );
        iIndex++;
       }
       else {

        Date.Format(_T("%02d/%02d/%2d"),NetList->ctDueDate.GetMonth(),NetList->ctDueDate.GetDay(
),NetList->ctDueDate.GetYear());
        m_lstCtrl.InsertItem(iIndex, Date, 0);
        m_lstCtrl.SetItemText(iIndex, 1, NetList->strToPartOp);

        m_lstCtrl.SetItemText(iIndex, 2, NetList->strFromPartOp);

        char buffer[10] = "";
        itoa( NetList->intQty, buffer, 10 );
        m_lstCtrl.SetItemText(iIndex, 3, buffer );

        m_lstCtrl.SetItemText(iIndex, 4, NetList->strResourceID );

        gcvt(NetList->fltSetupTime,6, buffer);
        m_lstCtrl.SetItemText(iIndex, 5, buffer );

        gcvt(NetList->fltRunTime,5, buffer);
        m_lstCtrl.SetItemText(iIndex, 6, buffer );

        gcvt((NetList->TotalTime),6, buffer);
        m_lstCtrl.SetItemText(iIndex, 7, buffer );

        gcvt(NetList->StartTime, 6, buffer);
        m_lstCtrl.SetItemText(iIndex, 8, buffer );

        gcvt( NetList->EndTime, 6, buffer);
        m_lstCtrl.SetItemText(iIndex, 9, buffer );
                                                                                    //Converts time
s into Time Bucket format
        int tempST = floor(NetList->StartTime);                                     //Uses integer
declaration to drop the fraction
        div_t div_result;

        div_result = div( tempST, glbSegrLen );                                    //Calculated the
 nr of time buckets
        float StartHr = (NetList->StartTime - tempST) + div_result.rem;            //Calculates tim
e bucket start hour
        int StartTimeBucket = div_result.quot;

        gcvt( StartTimeBucket, 6, buffer);
        m_lstCtrl.SetItemText(iIndex, 10, buffer );

        gcvt( StartHr, 6, buffer);
        m_lstCtrl.SetItemText(iIndex, 11, buffer );

        int tempET = floor(NetList->EndTime);

        div_result = div( tempET, glbSegrLen );                                    //Calculated the
 nr of time buckets
        float EndHr = (NetList->EndTime - tempET) + div_result.rem;                //Calculates tim
e bucket end hour
        int EndTimeBucket = div_result.quot;

        div_result = div( tempET, glbSegrLen );                                    //Calculates end
 time in time bucket format
        gcvt( div_result.quot, 6, buffer);
        m_lstCtrl.SetItemText(iIndex, 12, buffer );

        gcvt( div_result.rem, 6, buffer);
        m_lstCtrl.SetItemText(iIndex, 13, buffer );

        iIndex++;
       }

    }
    CCR = "";
    UpdateData(FALSE);
}
```

```
 void CIDConstraintDlg::ScheduleBackwards(CNet *pNet, CNet *End)
 {
   CNet* NetList;
   NetList = End;                                                //Starts at end of list and wo
rks backward
   CNet* temp;
   float maxTime = NetList->prev->EndTime;

   while (pNet != NetList) {
      if ((NetList->strFromPartOp == NetList->prev->strToPartOp) && (NetList->prev->EndTime > N
etList->StartTime)) {
              float testtime = NetList->StartTime;
              CString ToPart = NetList->strToPartOp;
              NetList->prev->EndTime = NetList->StartTime;
              NetList->prev->StartTime = NetList->prev->EndTime - NetList->prev->TotalTime;

              temp = NetList->prev->prev;
               float tempEndTime = NetList->prev->StartTime;
              while (pNet != temp) {
                 if ((NetList->prev->strResourceID == temp->strResourceID) && (temp->EndTime >
tempEndTime)){
                      temp->EndTime = tempEndTime;
                      temp->StartTime = temp->EndTime - temp->TotalTime;
                      tempEndTime = temp->StartTime;
                 }
                 temp = temp->prev;
              }
          }
      else {
          NetList = NetList->prev;
      }
     }
   float minTime = pNet->next->StartTime;


 }


//Procedure to updata the database with the schedule was used for testing but not called at the
 moment
void CIDConstraintDlg::UpdateTableSchedule(CNet *pNet, CNet *End)
{
 BOOL bSuccess = FALSE;
 CDatabase db;
 CNet* NetList;
 NetList = pNet;
 char TTime[10] = "";
 char STime[10] = "";
 char dummy[5] = "XXX";

 /*try
 {
  if (db.Open("SchedulerDb"))
  {
    while (End != NetList) {
       NetList = NetList->next;
        CString strSQL = CString("DELETE FROM Schedule"
        "WHERE ");
         strSQL += CString("ResourceID != '") +
         dummy + CString("'");

         bSuccess = TRUE;
    }
  }
 }

 catch(CDBException* pe)
 {
  AfxMessageBox(pe->m_strError);

  if (db.IsOpen())
  {
   db.Close();
  }
 }

 NetList = pNet;*/
 try
 {
  if (db.Open("SchedulerDb"))
  {
    while (End != NetList) {
       NetList = NetList->next;
       gcvt(NetList->TotalTime, 6, TTime);

       gcvt(NetList->StartTime, 6, STime);


       CString strSQL = CString("INSERT INTO Schedule (ResourceID, Start_Time, Total_Time)");
       strSQL += CString("VALUES ('") + NetList->strResourceID + ("', '") + STime + ("', '") +
TTime + ("') ");

       db.ExecuteSQL(strSQL);

       bSuccess = TRUE;

    }
  }
 }
 catch(CDBException* pe)
 {
  AfxMessageBox(pe->m_strError);

  if (db.IsOpen())
  {
   db.Close();
  }

  pe->Delete();
 }
  MessageBox("The Order was Successfully Placed!", "Order Placed", MB_OK);

//return bSuccess;
}

//This formula inserts rods between the constraints that feeds themselves
void CIDConstraintDlg::InsertRods(CNet *pNet, CNet *End)
{
   CNet* NetList;
   NetList = End;                                              //Starts at end of list and wo
rks backward
   float tempStartTime = 0;                                    //Temporary start time for com
parison
   float tempRunTime;                                          //Temporary run time for compa
rison
   float tempTotalTime;                                        //Temporary total time for com
parison
   float tempSetupTime;                                        //Temporary setup time for com
parison
   float rodGap;                                               //This is the actual delay bet
ween the first operation and the second
   bool Flag = FALSE;                                          //Terminating condition

   while (pNet != NetList) {
       //Checks if it the first occurence of the CCR in a leg
       // If it is it initialises the temp variables
       if (NetList->strResourceID == CCR) {
           tempStartTime = NetList->StartTime;
           tempRunTime = NetList->fltRunTime;
           tempSetupTime = NetList->fltSetupTime;
           tempTotalTime = NetList->TotalTime;
           NetList = NetList->prev;
           Flag = TRUE;
           break;
       }
      NetList = NetList->prev;
   }

   //This loop start at the end of the list and move forward
   while (pNet != NetList) {
         //Checks whether temp variables as been set and the loops has stepped to the second oc
curence of the CCR
         if ((Flag == TRUE) && (NetList->strResourceID == CCR)) { // if 1
           //run time of current job > runtime of already placed job
```

```
            if (tempRunTime >= NetList->fltRunTime) {
                //current time between jobs
                CString fp = NetList->strFromPartOp;
                float et = NetList->EndTime;

                //calculates what the time gap between jobs must be
                rodGap = 0.5*glbConstBuff - (NetList->TotalTime - NetList->fltSetupTime - NetLis
t->fltRunTime) - tempSetupTime;
               //   rodGap = 1;
                //if rodGap is > 0 then the rod needs to be placed
                if (rodGap > 0) {
                    NetList->EndTime = tempStartTime - rodGap;
                    float ett = NetList->EndTime;
                    NetList->StartTime = NetList->EndTime - NetList->TotalTime;
                    float dummyEndTime = NetList->StartTime;

                    //Re-initalises values
                    tempStartTime = NetList->StartTime;
                    tempRunTime = NetList->fltRunTime;
                    tempSetupTime = NetList->fltSetupTime;
                    tempTotalTime = NetList->TotalTime;
                  }
                if (rodGap <= 0) {
                    NetList->EndTime = tempStartTime;
                    NetList->StartTime = NetList->EndTime - NetList->TotalTime;

                    tempStartTime = NetList->StartTime;
                    tempRunTime = NetList->fltRunTime;
                    tempSetupTime = NetList->fltSetupTime;
                    tempTotalTime = NetList->TotalTime;
                }
            }
            //run time of current job < runtime of already placed job
            else {
                //calculates what the time gap between jobs must be
                rodGap = 0.5*glbConstBuff - (tempTotalTime - tempRunTime);
               // rodGap = 1;
                //if rodGap is > 0 then the rod needs to be placed
                if (rodGap > 0) {
                    NetList->EndTime = tempStartTime - rodGap;
                    float ett = NetList->EndTime;
                    NetList->StartTime = NetList->EndTime - NetList->TotalTime;
                    float dummyEndTime = NetList->StartTime;

                    //Re-initalises values
                    tempStartTime = NetList->StartTime;
                    tempRunTime = NetList->fltRunTime;
                    tempSetupTime = NetList->fltSetupTime;
                    tempTotalTime = NetList->TotalTime;
                }
                if (rodGap <= 0) {
                    NetList->EndTime = tempStartTime;
                    NetList->StartTime = NetList->EndTime - NetList->TotalTime;


                    tempStartTime = NetList->StartTime;
                    tempRunTime = NetList->fltRunTime;
                    tempSetupTime = NetList->fltSetupTime;
                    tempTotalTime = NetList->TotalTime;
                }
            }
        } //end if 1

          //This re-initialises the temp variables when you move on to a new production leg in t
he net
          if ((Flag == FALSE) && (NetList->strResourceID == CCR)) {
              if (tempStartTime > NetList->EndTime) {
//                   tempEndTime = NetList->EndTime;
                     tempStartTime = NetList->StartTime;
                     tempRunTime = NetList->fltRunTime;
                     tempSetupTime = NetList->fltSetupTime;
                     tempTotalTime = NetList->TotalTime;
              }
                else {
                    NetList->EndTime = tempStartTime;
                    NetList->StartTime = NetList->EndTime - NetList->TotalTime;

                    tempStartTime = NetList->StartTime;
                tempRunTime = NetList->fltRunTime;
                tempSetupTime = NetList->fltSetupTime;
                tempTotalTime = NetList->TotalTime;
            }

    /*      NetList->EndTime = tempStartTime;
            NetList->StartTime = NetList->EndTime - NetList->TotalTime;

            tempStartTime = NetList->StartTime;
            tempRunTime = NetList->fltRunTime;
            tempSetupTime = NetList->fltSetupTime;
            tempTotalTime = NetList->TotalTime;*/

            Flag = TRUE;
        }
        //Sets flag when you move on to a new production leg
        if (NetList->strResourceID == "") {
            Flag = FALSE;
        }
    NetList = NetList->prev;
  }//end while

}

float CIDConstraintDlg::GetCCREarliestTime(CNet *pNet, CNet *End)
{
  CNet* NetList;
  NetList = pNet;
  float NegativeTime;

  while (NetList != End) {
      NetList = NetList->next;
    if (NetList->strResourceID == CCR) {
       NegativeTime = NetList->StartTime;
       break;
       }
  }

  return NegativeTime;


}

int CIDConstraintDlg::GetColumnCount()
{
  int iNbrOfColumns = -1;

    CHeaderCtrl* pHeader = (CHeaderCtrl*)m_lstCtrl.GetDlgItem(0);
    ASSERT(pHeader);
    if (pHeader)
    {
        iNbrOfColumns = pHeader->GetItemCount();
    }

    return iNbrOfColumns;
}

void CIDConstraintDlg::InitListControl()
{
    static struct
    {
        LPTSTR szColText;
        UINT uiFormat;
    } columns[] = {
        _T("  Due Date  "), LVCFMT_LEFT,
        _T("To Part/Op"), LVCFMT_LEFT,
        _T("From Part/Op"), LVCFMT_LEFT,
        _T("Quantity"), LVCFMT_LEFT,
        _T("Resource"), LVCFMT_LEFT,
        _T("Setup Time"), LVCFMT_LEFT,
        _T("Run Time"), LVCFMT_LEFT,
        _T("Total Time"), LVCFMT_LEFT,
        _T("Start Time"), LVCFMT_LEFT,
        _T("End Time"), LVCFMT_LEFT,
        _T("Start Time Bucket"), LVCFMT_LEFT,
        _T("Start Hour"), LVCFMT_LEFT,
        _T("End Time Bucket"), LVCFMT_LEFT,
        _T("End Hour"), LVCFMT_LEFT
    };

    for (int i = 0; i < sizeof columns / sizeof columns [0]; i++)
    {
        m_lstCtrl.InsertColumn(i, columns[i].szColText, columns[i].uiFormat);
    }


  CHeaderCtrl* pHeader =
  (CHeaderCtrl*)m_lstCtrl.GetDlgItem(0);
  ASSERT(pHeader);

  if (pHeader) {
    // turn off redraw until the columns have all
    // been resized
    m_lstCtrl.SetRedraw(FALSE);

    // get the nbr of columns
    int iNbrOfColumns = pHeader->GetItemCount();
    int iNbrOfCols = GetColumnCount();

    for (int iCurrCol = 0; iCurrCol < iNbrOfCols; iCurrCol++) {
        m_lstCtrl.SetColumnWidth(iCurrCol, LVSCW_AUTOSIZE);
        int nCurrWidth = m_lstCtrl.GetColumnWidth(iCurrCol);

        m_lstCtrl.SetColumnWidth(iCurrCol, LVSCW_AUTOSIZE_USEHEADER);
        int nColHdrWidth = m_lstCtrl.GetColumnWidth(iCurrCol);
        m_lstCtrl.SetColumnWidth(iCurrCol, max(nCurrWidth, nColHdrWidth));
    }

    // now that col sizing is finished,
    // invalidate so that the control is repainted
    m_lstCtrl.SetRedraw(TRUE);
    m_lstCtrl.Invalidate();
  }
  m_lstCtrl.SetRedraw(FALSE);

  int iNbrOfCols = GetColumnCount();
  for (int iCurrCol = 0; iCurrCol < iNbrOfCols; iCurrCol++) {
        m_lstCtrl.SetColumnWidth(iCurrCol, LVSCW_AUTOSIZE);
        int nCurrWidth = m_lstCtrl.GetColumnWidth(iCurrCol);

        m_lstCtrl.SetColumnWidth(iCurrCol, LVSCW_AUTOSIZE_USEHEADER);
        int nColHdrWidth = m_lstCtrl.GetColumnWidth(iCurrCol);

        m_lstCtrl.SetColumnWidth(iCurrCol, max(nCurrWidth, nColHdrWidth));
    }
    m_lstCtrl.SetRedraw(TRUE);
    m_lstCtrl.Invalidate();


}



//This procedure brings the rest of the setup times into calculations agter ther ccr has been i
dentified

void CIDConstraintDlg::InsertSetupTimes(CNet *pNet, CNet *End)
{
  CNet* NetList;
  NetList = End;
  COrders* OrderList;
  OrderList = OrderEnd;
  OrderList = OrderList;

  //This loop inserts the additional setup times
  while (pNet != NetList) {

      if (OrderList->OrderPlaced == FALSE) {
          NetList->TotalTime = NetList->TotalTime + NetList->fltSetupTime;
          NetList->StartTime = NetList->EndTime - NetList->TotalTime;
      }

      NetList = NetList->prev;

      if ((OrderList->prev->strOrderID == NetList->strToPartOp) && (OrderList->prev != OrderSta
rt)) {
            OrderList = OrderList->prev;
        }

        if (OrderStart->strOrderID == NetList->strToPartOp) {
            break;
        }
    }

    //This section levels the load again after the insertion of the setup times
    CResourceData* ResourceList;
    ResourceList = pResourceData;
    NetList = End;                                                          //Starts at end of lis
t and works backward
    float EndTimeTracker = NetList->EndTime;
    float tempEndTime = NetList->EndTime;

    for (int i = 1; i <= glbNrOfResources; i++) {                           //Loops once through t
he list for each resource
        NetList = End;
        tempEndTime = EndTimeTracker;
        ResourceList = ResourceList->next;                                  //Starts with the firs
t resource in the list
                                                                            //and compares with th
e resrouce names in the net

        while (pNet != NetList) {
    //      NetList = NetList->prev;
            if (ResourceList->strResourceID == NetList->strResourceID) {  //loops thorugh the net
 backward and levels the load
                if (tempEndTime > NetList->EndTime) {
                    tempEndTime = NetList->EndTime;
                }
                 else {
                    NetList->EndTime = tempEndTime;
                }

              NetList->StartTime = tempEndTime - NetList->TotalTime;
              tempEndTime = NetList->StartTime;
            }
          NetList = NetList->prev;
      }
    }
}
```

```
// Parameters1.cpp : implementation file
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "Parameters1.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CParameters

IMPLEMENT_DYNAMIC(CParameters, CPropertySheet)

CParameters::CParameters(UINT nIDCaption, CWnd* pParentWnd, UINT iSelectPage)
    :CPropertySheet(nIDCaption, pParentWnd, iSelectPage)
{
}

CParameters::CParameters(LPCTSTR pszCaption, CWnd* pParentWnd, UINT iSelectPage)
    :CPropertySheet(pszCaption, pParentWnd, iSelectPage)
{
}

CParameters::~CParameters()
{
}


BEGIN_MESSAGE_MAP(CParameters, CPropertySheet)
    //{{AFX_MSG_MAP(CParameters)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CParameters message handlers

BOOL CParameters::OnInitDialog()
{
    BOOL bResult = CPropertySheet::OnInitDialog();

    int ids[] = {ID_APPLY_NOW, IDHELP, IDCANCEL};
    for ( int i = 0; i  < sizeof ids / sizeof ids[0]; i++)
    {
        CWnd* pWnd = GetDlgItem(ids[i]);
            if (pWnd)
            {
                pWnd->ShowWindow(FALSE);
            }
    }

    CWnd* pbtnOk = GetDlgItem(IDOK);
    ASSERT(pbtnOk);

    CRect rectSheet;
    GetWindowRect(rectSheet);

    // get size of ok button
    CRect rectOkBtn;
    pbtnOk->GetWindowRect(rectOkBtn);

    // get border space between btn bottom and sheet bottom
    int iBorder = rectSheet.bottom - rectOkBtn.bottom;

    // find first page
    CPropertyPage* pPage = GetPage(0);
    ASSERT(pPage);
    CRect rectPage;
    pPage->GetWindowRect(rectPage);

    // save width and height
    int cxOk = rectOkBtn.Width();
    int cyOk = rectOkBtn.Height();
    // move ok button
    rectOkBtn.top = rectPage.bottom + iBorder;
    rectOkBtn.bottom = rectOkBtn.top + cyOk;
    rectOkBtn.left = rectSheet.right -
        (cxOk + iBorder + .5*(iBorder));
    rectOkBtn.right = rectOkBtn.left + cxOk;
    ScreenToClient(rectOkBtn);
    pbtnOk->MoveWindow(rectOkBtn);

    // Change the text of the OK button to "Close"
    CWnd* pWnd = GetDlgItem(IDOK);
    pWnd->SetWindowText(_T("Cl&ose"));

    return bResult;
}
```

```
// ProductDataSet.cpp : implementation file
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "ProductDataSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CProductDataSet

IMPLEMENT_DYNAMIC(CProductDataSet, CRecordset)

CProductDataSet::CProductDataSet(CDatabase* pdb)
    : CRecordset(pdb)
{
    //{{AFX_FIELD_INIT(CProductDataSet)
    m_ProductID = _T("");
    m_Product_Description = _T("");
    m_nFields = 2;
    //}}AFX_FIELD_INIT
    m_nDefaultType = snapshot;
}


CString CProductDataSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=SchedulerDb");
}

CString CProductDataSet::GetDefaultSQL()
{
    return _T("[Product_Data]");
}

void CProductDataSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(CProductDataSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[ProductID]"), m_ProductID);
    RFX_Text(pFX, _T("[Product_Description]"), m_Product_Description);
    //}}AFX_FIELD_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CProductDataSet diagnostics

#ifdef _DEBUG
void CProductDataSet::AssertValid() const
{
    CRecordset::AssertValid();
}

void CProductDataSet::Dump(CDumpContext& dc) const
{
    CRecordset::Dump(dc);
}
#endif //_DEBUG
```

```
// CalendarDataSet.cpp : implementation file
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "CalendarDataSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CCalendarDataSet

IMPLEMENT_DYNAMIC(CCalendarDataSet, CRecordset)

CCalendarDataSet::CCalendarDataSet(CDatabase* pdb)
    : CRecordset(pdb)
{
    //{{AFX_FIELD_INIT(CCalendarDataSet)
    m_ID = 0;
    m_DayNr = 0.0;
    m_IsWorkDay = _T("");
    m_nFields = 4;
    //}}AFX_FIELD_INIT
    m_nDefaultType = snapshot;
}


CString CCalendarDataSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=SchedulerDb");
}

CString CCalendarDataSet::GetDefaultSQL()
{
    return _T("[Calendar]");
}

void CCalendarDataSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(CCalendarDataSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Long(pFX, _T("[ID]"), m_ID);
    RFX_Double(pFX, _T("[DayNr]"), m_DayNr);
    RFX_Text(pFX, _T("[IsWorkDay]"), m_IsWorkDay);
    RFX_Date(pFX, _T("[Date]"), m_Date);
    //}}AFX_FIELD_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CCalendarDataSet diagnostics

#ifdef _DEBUG
void CCalendarDataSet::AssertValid() const
{
    CRecordset::AssertValid();
}

void CCalendarDataSet::Dump(CDumpContext& dc) const
{
    CRecordset::Dump(dc);
}
#endif //_DEBUG
```

```
#if !defined(AFX_BUFFERLENGTH_H__845BE306_0E2F_4185_A329_691BB68FA1F0__INCLUDED_)
#define AFX_BUFFERLENGTH_H__845BE306_0E2F_4185_A329_691BB68FA1F0__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// BufferLength.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CBufferLength dialog

class CBufferLength : public CPropertyPage
{
    DECLARE_DYNCREATE(CBufferLength)

// Construction
public:
    void SaveBuffers();
    CBufferLength();
    ~CBufferLength();

// Dialog Data
    //{{AFX_DATA(CBufferLength)
    enum { IDD = IDP_BUFFER_LENGTH };
    CButton m_btnApply;
    float       m_edtAssemblyDef;
    float       m_ConstraintDef;
    float       m_edtShippingDef;
    float       m_edtAssembly;
    float       m_edtShipping;
    float       m_edtConstraint;
    int     m_intSegrDef;
    int     m_intStdDay;
    int     m_intSegr;
    int     m_intStdDayDef;
    //}}AFX_DATA


// Overrides
    // ClassWizard generate virtual function overrides
    //{{AFX_VIRTUAL(CBufferLength)
    public:
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    // Generated message map functions
    //{{AFX_MSG(CBufferLength)
    afx_msg void OnApplyShip();
    afx_msg void OnUpdateBuffer();
    virtual BOOL OnInitDialog();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()

};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous lin
e.

#endif // !defined(AFX_BUFFERLENGTH_H__845BE306_0E2F_4185_A329_691BB68FA1F0__INCLUDED_)
```

```
// ResourceSet.cpp : implementation file
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "ResourceSet.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CResourceSet

IMPLEMENT_DYNAMIC(CResourceSet, CRecordset)

CResourceSet::CResourceSet(CDatabase* pdb)
    : CRecordset(pdb)
{
    //{{AFX_FIELD_INIT(CResourceSet)
    m_ResourceID = _T("");
    m_Resource_Qty = 0;
    m_nFields = 2;
    //}}AFX_FIELD_INIT
    m_nDefaultType = snapshot;
}


CString CResourceSet::GetDefaultConnect()
{
    return _T("ODBC;DSN=SchedulerDb");
}

CString CResourceSet::GetDefaultSQL()
{
    return _T("[Resource_Data]");
}

void CResourceSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(CResourceSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Text(pFX, _T("[ResourceID]"), m_ResourceID);
    RFX_Int(pFX, _T("[Resource_Qty]"), m_Resource_Qty);
    //}}AFX_FIELD_MAP
}

/////////////////////////////////////////////////////////////////////////////
// CResourceSet diagnostics

#ifdef _DEBUG
void CResourceSet::AssertValid() const
{
    CRecordset::AssertValid();
}

void CResourceSet::Dump(CDumpContext& dc) const
{
    CRecordset::Dump(dc);
}
#endif //_DEBUG
```

```
// SchedulerView.cpp : implementation of the CSchedulerView class
//

#include "stdafx.h"
#include "TOC_Scheduler.h"

#include "SchedulerDoc.h"
#include "SchedulerView.h"
#include "IDConstraintDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

bool thisFlag = FALSE;
/////////////////////////////////////////////////////////////////////////////
// CSchedulerView

IMPLEMENT_DYNCREATE(CSchedulerView, CView)

BEGIN_MESSAGE_MAP(CSchedulerView, CView)
    //{{AFX_MSG_MAP(CSchedulerView)
    ON_WM_PAINT()
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSchedulerView construction/destruction

CSchedulerView::CSchedulerView()
{
    // TODO: add construction code here

}

CSchedulerView::~CSchedulerView()
{
}

BOOL CSchedulerView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CSchedulerView drawing

void CSchedulerView::OnDraw(CDC* pDC)
{
    CSchedulerDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
//  pDC->TextOut(20,20, "Hello");

}

/////////////////////////////////////////////////////////////////////////////
// CSchedulerView printing

BOOL CSchedulerView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CSchedulerView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add extra initialization before printing
}
```

```
void CSchedulerView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CSchedulerView diagnostics

#ifdef _DEBUG
void CSchedulerView::AssertValid() const
{
    CView::AssertValid();
}

void CSchedulerView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CSchedulerDoc* CSchedulerView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CSchedulerDoc)));
    return (CSchedulerDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSchedulerView message handlers




void CSchedulerView::myFunction(bool Flag)
{
        //Invalidate(TRUE);
        //UpdateWindow();



/*  thisFlag = Flag;
    if (thisFlag == TRUE) {
        CClientDC ClientDC( this );
        CString strText = ("Hello World!");
        CSize size = ClientDC.GetTextExtent(strText, strText.GetLength());

        ClientDC.SetTextColor(RGB( 255, 0, 0));
        ClientDC.SetBkColor(RGB( 0, 0, 0));
        ClientDC.TextOut( 50, 50,   strText, strText.GetLength() );


    }*/


}



void CSchedulerView::OnPaint()
{
    CPaintDC dc(this); // device context for painting
       // Invalidate(TRUE);
        UpdateWindow();

    // TODO: Add your message handler code here

    // Do not call CView::OnPaint() for painting messages
    int x = 50;

        CClientDC ClientDC( this );
        CString strText = ("Hello World!");
        CSize size = ClientDC.GetTextExtent(strText, strText.GetLength());

        ClientDC.SetTextColor(RGB( 255, 0, 0));
        ClientDC.SetBkColor(RGB( 0, 0, 0));
    //  ClientDC.TextOut( x, 50,    strText, strText.GetLength() );
        x = x + 100;
```

```
// SchedulerDoc.cpp : implementation of the CSchedulerDoc class
//

#include "stdafx.h"
#include "TOC_Scheduler.h"

#include "SchedulerDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSchedulerDoc

IMPLEMENT_DYNCREATE(CSchedulerDoc, CDocument)

BEGIN_MESSAGE_MAP(CSchedulerDoc, CDocument)
    //{{AFX_MSG_MAP(CSchedulerDoc)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSchedulerDoc construction/destruction

CSchedulerDoc::CSchedulerDoc()
{
    // TODO: add one-time construction code here

}

CSchedulerDoc::~CSchedulerDoc()
{
}

BOOL CSchedulerDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}



/////////////////////////////////////////////////////////////////////////////
// CSchedulerDoc serialization

void CSchedulerDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        // TODO: add loading code here
    }
}

/////////////////////////////////////////////////////////////////////////////
// CSchedulerDoc diagnostics

#ifdef _DEBUG
void CSchedulerDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CSchedulerDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSchedulerDoc commands
// TOC_Scheduler.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "TOC_Scheduler.h"

#include "MainFrm.h"
#include "SchedulerDoc.h"
#include "SchedulerView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSchedulerApp

BEGIN_MESSAGE_MAP(CSchedulerApp, CWinApp)
    //{{AFX_MSG_MAP(CSchedulerApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
    // Standard print setup command
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSchedulerApp construction

CSchedulerApp::CSchedulerApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CSchedulerApp object

CSchedulerApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CSchedulerApp initialization

BOOL CSchedulerApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

    // Change the registry key under which our settings are stored.
    // TODO: You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings();  // Load standard INI file options (including MRU)

    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CSchedulerDoc),
        RUNTIME_CLASS(CMainFrame),       // main SDI frame window
        RUNTIME_CLASS(CSchedulerView));
    AddDocTemplate(pDocTemplate);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CSchedulerApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CSchedulerApp message handlers
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "TOC_Scheduler.h"
#include "MainFrm.h"
#include "Parameters1.h"

#include "BufferLength.h"
#include "PlanningHorison.h"
#include "IDConstraintDlg.h"

#include "SchedulerDoc.h"
#include "SchedulerView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
    ON_WM_CREATE()
    ON_COMMAND(ID_SCHEDULING_IDCONST, OnSchedulingIDConstraint)
    ON_COMMAND(ID_SCHEDULING_DISPLAY, OnSchedulingDisplay)
    ON_WM_INITMENU()
    ON_COMMAND(ID_SCHEDULING_SCHEDULEPARAMETERS, OnScheduleParameters)
    ON_COMMAND(ID_SCHEDULING_IDCONSTRAINT, OnSchedulingIDConstraint)
    ON_UPDATE_COMMAND_UI(ID_SCHEDULING_IDCONSTRAINT, OnUpdateSchedulingIdconstraint)
    //}}AFX_MSG_MAP
    // Global help commands
    ON_COMMAND(ID_HELP_FINDER, CFrameWnd::OnHelpFinder)
    ON_COMMAND(ID_HELP, CFrameWnd::OnHelp)
    ON_COMMAND(ID_CONTEXT_HELP, CFrameWnd::OnContextHelp)
    ON_COMMAND(ID_DEFAULT_HELP, CFrameWnd::OnHelpFinder)
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers


void CMainFrame::OnScheduleParameters()
{
  CParameters sheet(IDS_PARAMETERS);
  CBufferLength pageBufferLength;
  CPlanningHorison pagePlanningHorison;

  sheet.AddPage(&pagePlanningHorison);
  sheet.AddPage(&pageBufferLength);
  sheet.DoModal();
}


void CMainFrame::OnSchedulingIDConstraint()
{
  CIDConstraintDlg().DoModal();
}

void CMainFrame::OnSchedulingDisplay()
{

    CSchedulerView().myFunction(TRUE);

}

void CMainFrame::OnInitMenu(CMenu* pMenu)
{
    CFrameWnd::OnInitMenu(pMenu);

    //CMenu::EnableMenuItem(ID_SCHEDULING_IDCONSTRAINT, MF_GRAYED);
    //ModifyMenu(3, MF_POPUP, ID_SCHEDULING_IDCONSTRAINT, MF_GRAYED);

    //MessageBox("hello","",MB_OK);
```

```
}

void CMainFrame::OnUpdateSchedulingIdconstraint(CCmdUI* pCmdUI)
{
	//MessageBox("hello","",MB_OK);
//	EnableMenuItem(ID_SCHEDULING_IDCONSTRAINT, MF_GRAYED);

}
```