

# Thesis

With the view to getting a  
PhD delivered by the *University of Liege*

Submitted and defended by **Frédéric Bair**  
In 2008

Title:

## **Developments of Tools Focused on Production Simulation to Improve Productivity in Shipyards' Workshops**

Board of examiners:

- President of the jury: Professor Jean Marchal
- Thesis Director: Professor Philippe Rigo
- Professor Jacques Rondal
- Professor Yves Crama
- Professor Pierre Duysinx
- Dr. Frank Roland
- Dr. Thierry Van Frachen
- M. Guy Janssen



**Doctoral School:** *Faculté des Sciences appliquées - University of Liege*

**Research Unit:** ANAST – *Naval Architecture*

## Acknowledgments

*I would like to express my gratitude to Professor Philippe RIGO advisor of my work, for his support and encouragement during this study.*

*Many thanks to Professors, colleagues of ANAST ("Architecture Navale et Analyse des Systèmes de Transport"), the University of Liège, friends and family for encouragement and support.*

*Many thanks to my proofreaders for their helpful advices.*

*Finally, I would like to thank the "Fond Social Européen" (FSE) and the "Région Wallonne" for the financial support and Aker Yards France for their availability to provide the required data.*

*Frédéric Bair  
Liege, December 2008*

# Abstract

*The goal of the thesis is to develop tools for improving shipbuilding workshops' productivity. These tools have been tested on workshops of Aker Yards France shipyard, in Saint-Nazaire.*

*The main chapter concerns the modelling of a workshop with the discrete-event simulation methodology. Production simulation is strongly used in some fields – as automobile industry – but is less used in shipbuilding. Indeed, the difficulty is that almost each piece to produce is unique. The objective is to show the importance of production simulation for such workshops. The modelled workshop is linked to a genetic algorithm to improve its productivity by optimizing its production sequence.*

*Secondly a tool of creation and optimization of PERT networks has been created. The tool contains a graphical interface to easily handle networks and an optimization algorithm. This last one can minimize total cost of any project for a given total time. The tool has been linked to the modelled workshop. Consequently we get a PERT diagram of the workshop that can highlight critical activities.*

*Finally, the last chapter explains the development of a tool to solve space allocation problems. Indeed space is often a critical point in many shipyards. Thanks to a user-friendly interface, the developed tool facilitates scheduling of these workshops. Furthermore it contains an optimization algorithm based on a heuristic approach to automate the planning process.*

*In conclusion a set of efficient tools has been carried out with concrete applications on real workshops.*

# Résumé

*L'objectif de la thèse est de développer une série d'outils qui permettent d'optimiser la production d'ateliers de construction navale. Ces outils sont testés sur des ateliers du Chantier Naval Aker Yards France à Saint-Nazaire.*

*Le chapitre principal traite de la modélisation d'un atelier – dit de PréPréFabrication – sur le principe de la simulation de production à événements discrets. La simulation de production est fortement utilisée dans certains domaines – industrie automobile par exemple – mais encore peu souvent en construction navale. En effet, la difficulté réside dans la diversité des pièces à produire. L'objectif est de démontrer l'intérêt de la simulation de production même dans ces ateliers où pratiquement chaque pièce à construire est unique. L'atelier modélisé a été couplé avec un algorithme génétique pour améliorer sa productivité en optimisant la séquence de production.*

*En second lieu, un outil de création et d'optimisation des réseaux PERT a été créé. L'outil contient une interface graphique permettant une manipulation aisée des réseaux et un algorithme d'optimisation. Ce dernier permet de minimiser le coût de réalisation d'un projet pour un temps total fixé. En étant couplé avec le logiciel de simulation développé, une création automatique du diagramme PERT de l'atelier est créée, permettant de mettre clairement en évidence ses activités critiques.*

*Enfin, le dernier chapitre de la thèse porte sur la création d'un logiciel permettant de résoudre les problèmes de gestion de surface et de planification rencontrés dans certains chantiers. En effet, l'espace au sol est un élément critique pour beaucoup d'ateliers de construction navale. Le logiciel développé, grâce à une interface conviviale et pratique, permet de faciliter grandement la planification. De plus, il contient un algorithme d'optimisation basé sur des heuristiques afin d'automatiser le processus de planification.*

*En conclusion, un ensemble d'outils efficaces a été généré, avec des applications concrètes sur des ateliers réels.*

# Table of Contents

<b>Chapter 1 Introduction.....</b>	<b>8</b>
<b>1.1 Context situation.....</b>	<b>8</b>
<b>1.2 Tools developed.....</b>	<b>9</b>
<b>Chapter 2 State-of-Art and Overview of the Industrial Environment.....</b>	<b>12</b>
<b>2.1 Introduction.....</b>	<b>12</b>
<b>2.2 Simulation: general overview and State-of-Art.....</b>	<b>12</b>
2.2.1 Introduction to the simulation.....	12
2.2.2 Object oriented software.....	23
2.2.3 Time-Oriented Simulation and Event-Oriented Simulation.....	26
<b>2.3 Genetic algorithms.....</b>	<b>29</b>
2.3.1 Description.....	29
2.3.2 Disadvantages of GA.....	32
<b>2.4 Presentation of the Aker Yards France shipyard.....</b>	<b>33</b>
2.4.1 Introduction.....	33
2.4.2 Company's workforce.....	34
2.4.3 Industrials means.....	34
2.4.4 The production.....	35
2.4.5 Main Workshops of the Shipyard.....	35
<b>Chapter 3 Development of a Virtual Workshop.....</b>	<b>43</b>
<b>3.1 Introduction.....</b>	<b>43</b>
<b>3.2 Virtual Model of the PrePreFabrication Workshop.....</b>	<b>44</b>
3.2.1 Introduction.....	44
3.2.2 Detailed presentation of the workshop.....	45
3.2.3 The database treatment.....	55
3.2.4 Description of the simulation software.....	67
3.2.5 Development of the PrePreFabrication Model.....	86
3.2.6 Analysis of a realistic problem.....	167
<b>3.3 Optimization of the Model.....</b>	<b>178</b>
3.3.1 Introduction.....	178
3.3.2 Applications to the Workshop.....	180
<b>3.4 Conclusion.....</b>	<b>193</b>

<b><i>Chapter 4 Development of a Specific PERT Tool and Application to the PrePreFabrication Workshop</i></b> .....	<b>195</b>
<b>4.1 PERT: Theoretical background</b> .....	<b>195</b>
4.1.1 Introduction.....	195
4.1.2 The Network Diagram .....	196
4.1.3 Terminology.....	198
4.1.4 Benefits of PERT .....	199
4.1.5 Limitations .....	199
<b>4.2 Tool developed</b> .....	<b>200</b>
4.2.1 Introduction.....	200
4.2.2 Theoretical approach.....	202
4.2.3 Development of the tool.....	210
<b>4.3 Application to the PrePreFabrication Workshop</b> .....	<b>217</b>
4.3.1 Methodology.....	217
4.3.2 Automatic creation of the PERT .....	221
4.3.3 Conclusion .....	226
<b><i>Chapter 5 Development of a tool to solve Space Allocation Problem</i></b> .....	<b>229</b>
<b>5.1 Introduction</b> .....	<b>229</b>
<b>5.2 Development of the tool</b> .....	<b>231</b>
5.2.1 Introduction.....	231
5.2.2 Interface of the software .....	232
5.2.3 Theory developments of the optimisation.....	243
5.2.4 Applications .....	254
<b>5.3 Conclusion</b> .....	<b>257</b>
<b><i>Chapter 6 Conclusion</i></b> .....	<b>259</b>
<b>6.1 Direct results</b> .....	<b>259</b>
<b>6.2 Indirect Results</b> .....	<b>259</b>
<b>6.3 Conclusion</b> .....	<b>260</b>
<b>6.4 Perspectives</b> .....	<b>261</b>
<b><i>Glossary</i></b> .....	<b>263</b>
<b><i>Table of Figures</i></b> .....	<b>265</b>
<b><i>References</i></b> .....	<b>270</b>



# Chapter 1

## Introduction

### 1.1 Context situation

Since the end of the Second World War shipbuilding in Europe has seen the development of competitors springing up in the Asian market. Countries such as Japan, followed notably by South Korea (thanks to a cheap workforce and good work organisation), have become in less than thirty years the primary shipbuilders in the world, forcing a variety of big European shipyards to specialize to avoid a reduction of their activities.

Consequently, only passenger ships or advanced ships are still the assets of Italian, German, French and Scandinavian shipyards. Asian shipyards produce about eighty percent of regular ships, such as tankers or container ships.

Nowadays European shipbuilding is facing a new conjuncture, so working methods and management must evolve to allow the survival of the sector. Indeed on the one hand Japan and South Korea have, since the end of the 1980s, carried out important progress in their modernisation and productivity, and on the other hand China is also becoming more and more competitive in the sector. Consequently European shipbuilders have to reorganize completely their working methods to stay competitive.

To accomplish this crucial task many methods could be considered but they must have all the same objective: reduction of the production cost. Several paths are possible and could be classed in two major methods:

- Optimization of the ship to reduce production cost;
- Optimization of the shipyard to improve its productivity at lower cost.

The first method can for example concern studies of new cheaper material, or optimizing the ship's scantling to minimize its weight and/or its cost – material cost and workforce cost. There are broad improvement possibilities here but less than the second method offers. Indeed, there are numerous possibilities to improve productivity:

- The way ships are cut in blocks has a deep impact in a shipyard's workshops. An optimisation of the block splitting could lead to interesting results;



- General production flow inside the workshop must be controlled and well predicted. Optimisation of this flow is fundamental to exploiting workshop capacity to its maximum. This flow management can be studied and optimised with simulation tools;
- Improvement of scheduling tools. Better scheduling can reduce the production time and consequently reduce cost. Tools can improve the scheduling but also simply facilitate its determination. In the second case the advantage is that different scheduling can be tested very easily and thus lead also to a kind of optimisation;
- Local production optimization in bottlenecks workshop. Many methods exist and one of them is the production simulation;
- Resources optimization: a better use of available resources to avoid waiting time is also a way to reduce costs;
- Etc.

The object of this thesis will be to develop several new tools to improve a shipyard's competitiveness.

## **1.2 Tools developed**

The main tool developed in this thesis focuses on the production simulation. Development of the model and optimization is treated in Chapter 3 which is the core of the thesis. The goal is to simulate a very complex shipbuilding workshop in order to highlight the benefits and advantages that can be obtained. Production simulation is used for some industrial applications – especially in the car industry – but is less frequent in shipyards. One fundamental reason is that shipbuilding production is quite different from other industries: the products to be built are almost every time the only one of its kind. This point makes modelling much more complex and raises difficulties in optimizing it. Nevertheless some shipyards – but still a minority – are nowadays deeply involved in the simulation. The chapter's objective is to show interests to use production simulation even with complex workshops. Rather than working on an academic or theoretical workshop we have worked on a real workshop, at the Aker Yards Shipyard in Saint-Nazaire (France). Its main characteristic is its major complexity, for two reasons:

- A completely unusual production line. Pieces do not have a linear progress crossing from machine to machine;

- Products are small assemblies with various features. Dimensions can change a lot, individual pieces come from many workshops, etc.

We will show that such a complex system could be modelled by simulation and give interesting results to be exploited to improve its productivity. Furthermore one aspect is to see possibilities to link the simulation model to an optimisation tool. Even if the simulation was already used by some shipyards at the beginning of the thesis, the link simulation/optimization was never carried out. Now it seems that some shipyards have understood the interest and are trying to make this link. The chapter shows that even for unusual workshops optimization is possible and can provide not insignificant gains.

In Chapter 4 we have developed a tool that has several interesting aspects. The first characteristic is that the tool can be used in various application fields and is not specific to shipbuilding. The tool allows the user to build easily complex PERT<sup>1</sup> diagrams and to optimize them. In other words we can manage the time/cost activities of a complex project in a very convenient way. Three different uses can be made of it:

- A simple study of the project: bottlenecks activities will be clearly highlighted and information about each activity: the earliest and latest starting dates, margins, etc;
- Optimization of cost of the project (minimization of cost): to carry out a project each activity takes time and of course uses resources and thus produces costs. If we modify the resources allocated to an activity its cost will also change. A deep analysis can show that some activities can take more time without increasing the total time of the project and other ones have an important impact. For the total time of the given project a cost optimization can be done for each activity;
- A parametric study to optimize the total time/cost of the project: here the goal is to see if a reduction of the total time of the project will increase the cost greatly or not, or to see if an increase of the total time will supply benefits. The third point is basically an improvement of the second.

This tool is very useful when we have many parallel activities. In that case a general overview of the process could be very difficult without appropriate tools. As we will see this is exactly the situation of the workshop modelled in Chapter 3. An important task is to link

---

<sup>1</sup> *Program Evaluation and Review Technique*

the simulation model to the PERT tool developed. This link provides a new perspective on results and could help greatly schedulers to have a better planning for the workshop.

Chapter 5 analyses completely different problems: space allocation problems! Shipyards are located where land is limited. Once their developments have been carried out they often face space problems. Consequently these space problems occur in shipyard workshops – mainly on the last steps of the production where assemblies are already huge and occupy significant ground space. A major challenge for shipyards is to manage this phenomenon! Here again a powerful tool has been developed. The goals and methodology used are completely different from the previous chapters. The production will not be improved but the workshop scheduling will be facilitated and optimized! Facilitating the scheduling is indirectly a kind of simulation: indeed if we carry out planning in one day rather than in five days it allows the time to try five alternative planning schedules and to finally choose the best one. Better planning could of course be found if the tool contains an optimizer. Both these characteristics are parts of the developed tool. The software contains a very user-friendly interface to facilitate manual scheduling and has also an optimizer to find the best one. A workshop of the Aker Yards France shipyard was selected for benchmarking. For this tool we have tested three different workshops.

# Chapter 2

## State-of-Art and Overview of the Industrial Environment

### 2.1 Introduction

Firstly the chapter contains a state-of-art of theories and methodologies used: section 2.2 deals with the simulation – more specially production simulation – and section 2.3 gives some reminder about genetic algorithms. Indeed in Chapter 3 a simulation model is developed and its optimization is done thanks to a genetic algorithm.

Tools developed in the framework of this thesis have been tested on real workshops. These workshops are located in the Aker Yards France shipyard at Saint-Nazaire. In this chapter a brief description of the shipyard is given.

### 2.2 Simulation: general overview and State-of-Art

#### 2.2.1 Introduction to the simulation

Simulation is a fundamental point in Chapter 3 – the core of this thesis. First of all, it is important to define the meaning of the term *simulation*. What is simulation? Simulation is a generic term used in completely different fields. In a general way simulation is the emulation of a system, including its dynamic processes, in a model one can experiment with. It aims at achieving results that can be transferred to a real world installation. In addition, simulation defines the preparation, execution and evaluation of carefully directed experiments within a simulation model.

The simulation model could be a physical model or, as is the case in this thesis, a virtual model.

Simulation is the process of designing a model of a real or imagined system and conducting experiments with that model. The purpose of simulation experiments is to understand the behaviour of the system or evaluate strategies for the operation of the system. Assumptions are made about this system and mathematical algorithms and relationships are derived to describe these assumptions – this constitutes a “model” that can reveal how the system works. If the system is simple, the model may be represented and solved analytically.

A single equation such as  $Distance = (Rate * Time)$  is an analytical solution representing the distance travelled by an object at constant rate for a given period of time.

However, problems which are of interest in the real world are usually much more complex than this. In fact, they may be so complex that a simple mathematical model cannot be constructed to represent them. In this case, the behaviour of the system must be estimated with a simulation. Exact representation is rarely possible in a model, constraining us to approximations concerning a degree of accuracy that is acceptable for the purposes of the study. Models have been constructed for almost every imaginable system including factories, communication and computer networks, integrated circuits, highway systems, flight dynamics, national economies, social interactions, and imaginary worlds. In each one of these environments, a model of the system has proved to be more cost-effective, less dangerous, faster, or more practical than experimenting with the real system.

For example, a company may be interested in building a new factory to replace an old one, but how to make sure the increased productivity will justify the investment? In this case, simulation would be used to perform a model of the new factory. The model will describe the floor space required, number of machines, number of employees, placement of equipment, production capacity of each machine, and the waiting time between machines. Simulation runs would then evaluate the system and provide an estimate of the production capacity and the costs of a new factory. This type of information is invaluable in making decisions without having to build the new factory to arrive at an answer [Smith R.D., 1999].

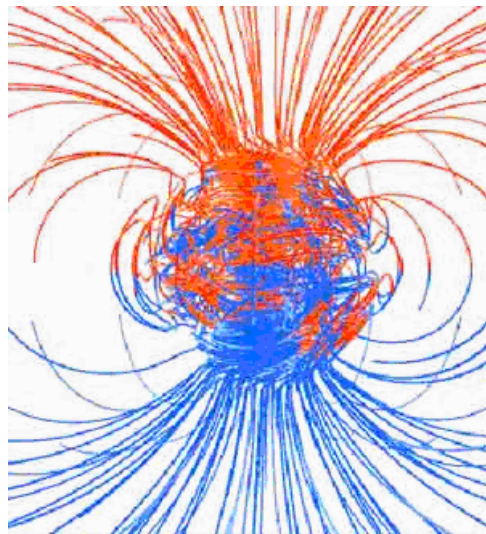
### 2.2.1.1 Areas of application

Simulation can be used in very different fields. For example, we can mention the following fields:

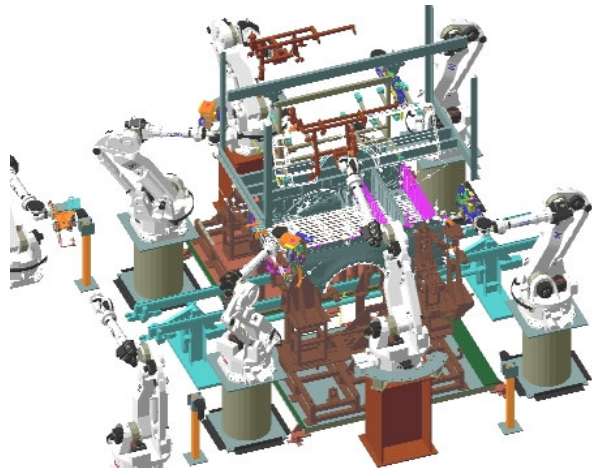
- Manufacturing systems;
- Public systems:
  - Health Care;
  - Military;
  - Natural Resources;
- Transportation systems;
- Construction systems;

- Restaurant and entertainment systems;
- Business process reengineering;
- Food processing;
- Computer system performance;
- ...

In the following diagrams we can see some of the different areas – non exhaustive – simulations can be applied to:



**Fig. 1: Magnetic Field Simulation**



**Fig. 2: Simulation of Automatic Robots**



Fig. 3: Simulation of the descent of Opportunity Rover into the Victoria crater (on Mars)

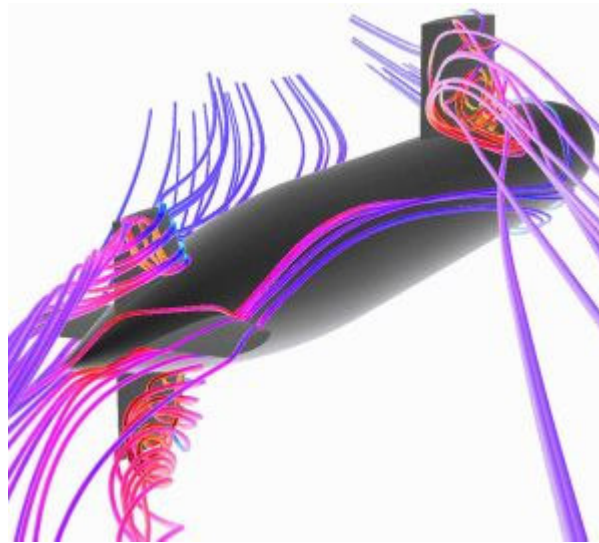


Fig. 4: Simulation of the flow around a submarine by using fluid mechanical theory

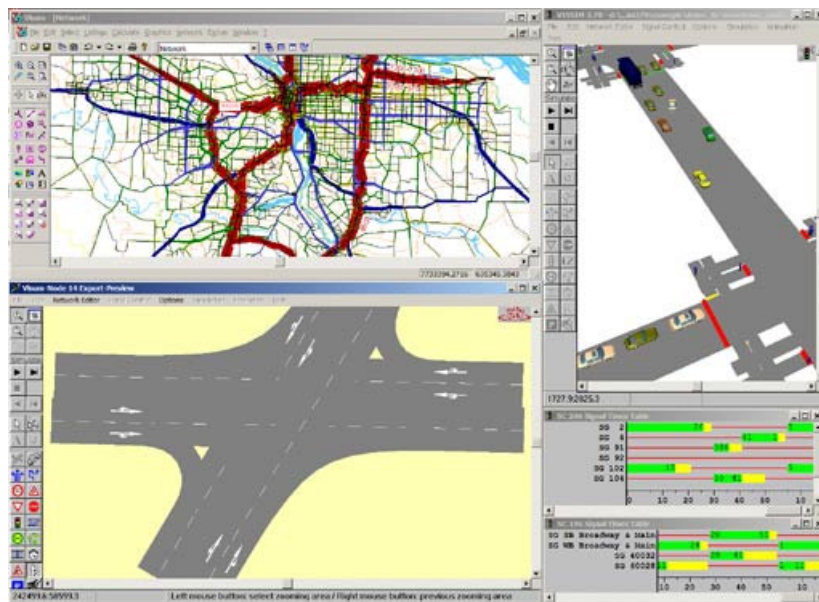


Fig. 5: Microscopic Traffic Simulation

### **2.2.1.2 Systems and system environment**

To model a system, it is necessary to understand the concept of the system and the system boundaries. A system is defined as a group of objects that are joined together in some regular interaction or interdependence with the aim of accomplishing some goal. An example is a production system manufacturing automobiles. The machines, component parts, and workers operate jointly along an assembly line to produce a high-quality vehicle.

A system is often affected by changes occurring outside the system. Such changes are said to occur in the system environment [Gordon, 1978]. In modelling systems, it is necessary to decide on the boundary between the system and its environment. This decision may depend on the purpose on the study.

In the case of the factory system, for example, the factors controlling the arrival of orders may be considered to be outside of the factory's influence and therefore part of the environment. However, if the effect of supply on demand is to be considered, there will be a relationship between factory output and the arrivals of orders and this relationship must be considered as an activity taking place in the system. Similarly, in the case of a bank system, there may be a limit on the maximum interest rate that can be paid. For the study of a single bank, this would be regarded as a constraint imposed by the environment. In a study of the effects of monetary laws on the banking industry, however, the setting of the limit would be an activity taking place in the system.

### **2.2.1.3 Components of a system**

In order to understand and analyze a system, a number of terms need to be defined. An entity is an object of interest in the system. An attribute is a property of an entity. An activity represents a time period of a specified length. If a bank is being studied, customers might be one of the entities, the balance in their checking accounts might be an attribute, and making deposits might be an activity.

The collection of entities that compose a system for one study might only be a subset of the overall system for another study. For example, if the bank mentioned above is being studied to determine the number of tellers needed, the system can be defined as that portion of the bank consisting of the regular tellers and the customers waiting in line.



#### 2.2.1.4 Different kinds of simulations

Numerical simulations can be divided into different categories and it is important to clearly distinguish each one of them.

- *Discrete or continuous*: in a discrete simulation state, the system studied can be modified only in a finite number of moments. It manages events in time; in a continuous simulation, the state of the system can change continuously. For instance the simulation of a stock management system is a discrete simulation. Effectively, the state of the system only changes at isolated moments. In practice, most simulations use both discrete and continuous state variables, but one of these is predominant and drives the classification of the entire simulation.
- *Static (steady state) or dynamic*: most simulations describe the evolution of a system over time, but some of them do not contain that temporal aspect. In numerical analysis the evaluation of integrals often uses simulation techniques.
- *Stochastic or deterministic*: simulation models can include stochastic elements or not. A Deterministic simulation most of the time consists of evaluating the evolution of a system according to different scenarios. Example: a model of a national economy under different taxation systems; a model of the performance of a Formula 1 car with different motor configurations, etc. A stochastic system is a system in which some parameters are not fixed but have variance (distribution).
- *Local or distributed*: distributed models run on a network of interconnected computers, possibly through the Internet. Simulations dispersed across multiple host computers like this are often referred to as “distributed simulations”. There are several standards for distributed simulation, including Aggregate Level Simulation Protocol (ALSP), Distributed Interactive Simulation (DIS), High Level Architecture (HLA) and Test and Training Enabling Architecture (TENA).

#### 2.2.1.5 The history of the simulation

One of the pioneers of simulation concepts was John von Neumann. In the late 1940s he conceived the idea of running multiple repetitions of a model, gathering statistical data, and deriving information on the behaviour of the real system based on these models. This came to be known as the Monte Carlo method because of the use of randomly generated variables to represent behaviours that could not be modelled exactly, but could be characterized

statistically. Von Neumann used this method to study the random actions of neutrons and the effectiveness of aircraft bombing missions. These methods are those that were first used in industry to determine the maximum potential productivity of factories.

Concepts for Discrete Event Simulations (DES) were developed in the late 1950s. The first DES-specific language was developed at General Electric by K.D. Tocher and D.G. Owen. The General Simulation Program (GSP) was created to study manufacturing problems at General Electric and was shared with the rest of the world at the Second International Conference on Operations Research [Smith R.D., 1999].

### **2.2.1.6 Why do simulation?**

We may wonder whether simulation must be used to study dynamic systems. There are many methods of modelling systems which do not involve simulation but which involve finding solutions within a closed-form system (such as a system of linear equations). Simulation is often essential in the following cases:

- 1) The model is very complex with many variables and interacting components;
- 2) The underlying variable relationships are nonlinear;
- 3) The model contains random variates<sup>2</sup>;
- 4) The model output will be visual, as in a 3D computer animation.

The strength of simulation is that – even for easily solvable linear systems – a uniform model execution technique can be used to solve a large variety of systems without resorting to a “bag of tricks” where one must choose special-purpose and sometimes arcane solution methods to avoid simulation. Another important aspect of the simulation technique is that one builds a simulation model to replicate the actual system. When one uses the closed-form approach, the model is sometimes twisted to suit the closed-form nature of the solution method rather than to accurately represent the physical system. A harmonious compromise is to tackle system modelling with a hybrid approach using both closed-form methods and simulation. For example, we might begin to model a system with closed-form analysis and then proceed later with a simulation. This evolutionary procedure is often very effective.

---

<sup>2</sup> *Variate: a variable quantity that is random*

### **2.2.1.7 Advantages**

Simulation approach gives most of the advantages of modelling:

- We can easily vary the parameters of a system and evaluate the impacts of these changes: purchase of a new machine, modification of the profile of a new car, etc.
- We can do errors on the model rather than in reality, etc.

Some advantages are more specific to the simulation [Pegden et al 1995]:

- New policies, operating procedures, decision rules, information flows, organizational procedures, and so on, can be benchmarked and then validated or discarded without the disrupting ongoing operations of the real system;
- New hardware designs, physical layouts, transportation systems, and so on, can be tested without committing resources for their acquisition;
- Hypotheses about how or why certain phenomena occur can be tested for feasibility;
- Time can be compressed or expanded allowing for a speed up or slow down of the phenomena under investigation;
- Insight can be obtained about the interaction of variables;
- Insight can be obtained about the importance of variables on the performance of the system. Bottleneck analysis can be performed indicating where work in process, information, materials, and so on are being excessively delayed;
- A simulation study can help in understanding how the system operates rather than how individuals think the system operates;
- “What if” questions can be answered. This is particularly useful in the design of new systems;
- We can carry out the evaluation of very complex systems where analytic solutions are not known and for which simulation is the only possible approach. Simulation is often used as a last resort;

Simulation models often have a visual interface, sometimes with graphic animations and this fact makes them more reliable to the eyes of managers.

### 2.2.1.8 Disadvantages

When conducting a simulation or contriving a model certain limitations must be acknowledged:

- Each run of a model is usually the result of a random experiment and is thus only an estimation of the studied parameters (mean waiting times in a queue, probability of saturating the system, etc.). This is not the case for analytic models that give exact values of these parameters – of course in the hypothesis of the model, and only if the model can be solved analytically. As a consequence, simulation results may be difficult to interpret. Since most simulation outputs are essentially random variables (they are usually based on random inputs), it may be hard to determine whether an observation is a result of system interrelationships or randomness;
- Model building requires special training. It is an art that is learned over time and through experience. Furthermore, if two models are constructed by two skilled individuals, they may have similarities, but it is highly unlikely that they will be the same.
- The ability to create a model that accurately represents the system to be simulated is not immediately apparent. Real systems are extremely complex and a determination must be made about the details that will be captured in the model. Some details must be omitted and their effects lost or aggregated into other variables that are included in the model. In both cases, an inaccuracy has been introduced and the ramifications of this must be evaluated and accepted by the model developers and the customers [Smith R.D., 1999];
- Another limitation is the availability of data to describe the behaviour of the system. It is common for a model to require input data that is scarce or unavailable. This issue must be addressed prior to the design of the model to minimize its impact once the model is completed [Smith R.D., 1999];
- Development of a simulation model requires time and important resources. Skimping on resources for modelling and analysis may result in a simulation model or analysis that is not sufficient for the task;
- Once a model has been developed, the flexibility of the tool and its acceptance can lead to pernicious effects: for example, an unjustified trust in the accuracy of results

calculated. This fact is reinforced by the visual aspect – animations, etc. – that reduce the distance between the model and the object simulated;

- Simulation is used in some cases when an analytical solution is possible, or even preferable. This is particularly true in the simulation of some waiting lines where closed-form queuing models are available.

In defence of simulation, some disadvantages can be offset, as in the following:

- Marketers of simulation software have been actively developing packages that contain models that only need input data for their operation. Such models have the generic tag “simulators” or “templates”;
- Many simulation software sales people have developed output analysis capabilities within their packages for performing very thorough analyses;
- Simulation can be performed faster today than yesterday, and even faster tomorrow. This is attributable to the advances in hardware that permit rapid running of scenarios. It is also attributable to the advances in many simulation packages. For example, some simulation software contains a library of objects library for modelling material handling such as fork lift trucks, conveyors, and automated guided vehicles;
- Closed-form models are not able to analyse most of the complex systems that are encountered in practice.

### **2.2.1.9 Limitations**

It is also important to specify what simulation can do and what simulation cannot do. Here is a non exhaustive list of possibilities given to the users with a flow simulation:

- To give realistic estimations:
  - of the system’s expected behaviour;
  - of variations inside the system.
- To evaluate the effects of the following actions:
  - to add, move or delete machines;
  - to modify the flow;

- to modify the duration of processes;
- to introduce new products or delete existing products;
- to modify handling equipment;
- to modify scheduling;
- to add or delete workers;
- ...

However, the simulation is not able to solve directly all the problems. Here is a list of tasks that simulation cannot do alone.

- Simulation can not optimize a system's performance. Simulation can only give answers to questions like: "What will happen if ...?" Simulation acts as a black according to a given scenario. It only reproduces the behaviour of the modelled system.
- Simulation cannot give correct results if data is imprecise. An important point in simulation is the data used. It is essential to check the validity of this data if we want to avoid getting results that have no link with the real system. This point is particularly true when the simulated system has a high level of complexity and thus when an analytic verification of results given by the simulation is not possible. The quality of simulation results depends exclusively on the quality of modelling rules and of the data introduced. Simulation tools always give a result but give no information about the validity of this result.
- Simulation cannot describe the characteristics of a system which is not completely modelled. The modelling phase generally has several steps. First we build a global model of the system then we increase step by step the level of details. Each step must be validated. The problem in the modelling stage is knowing which level of details to reach in order to have a model accurate enough to simulate the reality. This is why it is better to refine the model rather than to have a model which is oversimplified and inaccurate.
- Simulation cannot solve problems but only give information and indications about how to solve it. When the model is ready to be used for the simulation we still have to know which information we want to observe during or at the end of the simulation. This information is the variables of the system on which statistical analysis is done. Most software gives standard statistical results that can be shown with visualisation

tools. Usually we can get the quantity and duration relative to the production flow, or occupation rate, breakdown rate, waiting times of machines and resources.

It is only with observations of the model during and after the simulation that we can try to find a solution to the given problem.

## **2.2.2 Object oriented software**

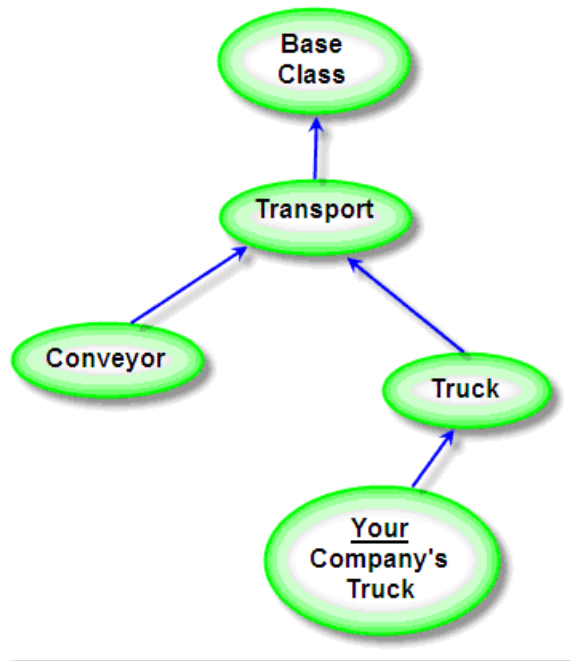
Object-oriented (OO) [Booch, 1991] techniques have received a lot of publicity in recent years and the use of OO techniques is becoming increasingly common. For example, the C++ & Delphi (Pascal) languages along with OO databases and even manufacturing system design methodologies. Interestingly the roots of OO techniques can be traced back to the *SIMULA* simulation programming language developed in the 1960s.

The strength of object oriented techniques lies in the ability to produce ‘modular’ code (known as classes) that can be “easily” modified and reused [Shewchuk & Chang, 1991]. Libraries of the classes can be built up and used to create software such as simulation models. The ability to place software complexity into classes and to be able to realistically represent entities from the real world in software make OO techniques ideally suited to simulation which is inherently complex.

Note that the difference between traditional software design and object-oriented software concerns the way in which the data and mechanisms are structured. In traditional simulation software the data and the event routines are dispersed throughout the software. In the OO approach, anything related to a single entity (both data and event routines) is bundled together to form a class (e.g. a machine). Objects (e.g. machine1, machine2, etc.) of the machine class can then be created.

One of the key advantages of using object oriented software in simulation is the ability to use the OO concepts to extend the functionality of the software. By taking the base functionality (e.g. transport) classes can be inherited to create a more dedicated class (e.g. conveyor). Hence the general functionality such as “moving units” and “processors” can be inherited to provide manufacturing specific functionality. Furthermore the manufacturing functionality can be inherited to company specific functionality – see Fig. 6 .

Therefore the general functionality such as “moving units” and “processors” can be inherited to provide manufacturing specific functionality. Furthermore the manufacturing functionality can be inherited to company specific functionality.



**Fig. 6: Hierarchical view of Object Oriented Concept Example**

There are two important points to note here that illustrate the power of object-oriented simulation software. Firstly, the developed functionality is part of a library, not a model. Therefore the functionality is developed independently and used to build many different models quickly. Secondly, functionality developed as a library can be exchanged with other users in other institutions. E.g. if one institution or company developed a class library for hot rolling mills this could be given to others to enable them to quickly develop their own models of their own particular rolling mills of interest.

Important notions linked to object-oriented software have to be clarified: the notion of classes, subclasses, instances and inheritances.

### 2.2.2.1 About classes

Imagine that, for example, you have to find the optimum type of storage for a production plant. First, you have to come up with different types of storage that might work for the specific installation, such as fully automated high bay warehouse, manually operated shelving system, etc. To be able to compare the different types of storage, you are going to build several simulation models, execute simulation runs and suggest the type of storage with the best cost/gain relation.

Your simulation models are going to be variants of a single basic model, i.e. you are going to use the same production installation and are going to modify the storage system.



When creating the model variants you are going to build the basic model first. In conventional systems you would then copy this basic model until you arrive at the number of variants you need. With object oriented software, on the other hand, you will inherit this basic model, which we call the parent model, and arrive at child models. The main difference between a copied and an inherited model is that a child model recognizes which parent it is derived from, while a copied model knows nothing about its origin.

Now you are going to insert the different types of stores into the model variants you created. If you now find a modelling error while modelling the basic model or if a specification changed, a conventional system requires changing all copies, which is time consuming and error prone. With object-oriented software, on the other hand, you make the change once in the parent model. It then immediately propagates all of your changes to all of its children, provided you did not change the setting in the child model. This saves a considerable amount of time and the hassle of manually updating a number of sub-models. Many programming errors are also avoided.

By changing a property in the class object, you change that property for all the objects you derived from this class. This is much less error-prone than having to change the same property for each and every individual object over and over again.

- A *class* passes all of its properties on to an instance you derive from it;
- A *class* passes those of its properties on to a *subclass* you derive from it, for which you do not deactivate inheritance;
- A *subclass* is an object in the class library which inherits some, but not all, of its properties from another *class*. By deactivating inheritance for certain dialog items, you can define properties that only apply to this subclass. An example of subclasses could be several lines that only differ in their length;
- An *instance* is an object which you insert into your simulation model from the *Class Library*.

The objects use class relations and origin relations to inherit their properties from other objects:

- The object inherits all of its basic properties from its **class** object. This includes settings it has by default, build-in *Methods*, basic functionalities, etc. The *class* is the object in the class library you instantiated the selected object from.

- The object inherits its settings from the **originating** object, provided you did not change them locally within the object. The *origin* of the selected object is the object you derived it from.

Making full use of the potentials of inheritance saves a considerable amount of time and effort during modelling.

### ***Using Inheritance***

Inheritance allows one class or object to incorporate the data or behaviour of another.

Inheritance has a number of applications:

- **Specialization** of existing classes or objects. In specialization, the new class or object has data or behaviour aspects which are not part of the inherited class.
- **Extension** to provide additional data or behaviour features. In contrast to the case of specialization, with extension the new data or behaviours could have been provided in the inherited class because they are generally applicable to all instances of the class.
- **Code re-use** to allow a new class to re-use code, in our case any of the settings you selected, which already exists in another class.

## **2.2.3 Time-Oriented Simulation and Event-Oriented Simulation**

The aim of simulation is to arrive at objective decisions by dynamic analysis, to enable managers to safely plan and, in the end, to reduce cost.

Thus, if real systems and plants are too expensive for conducting experiments and the time to conduct trials is too limited and too expensive, modelling, simulation and animation are excellent tools for analyzing and optimizing time dynamic processes.

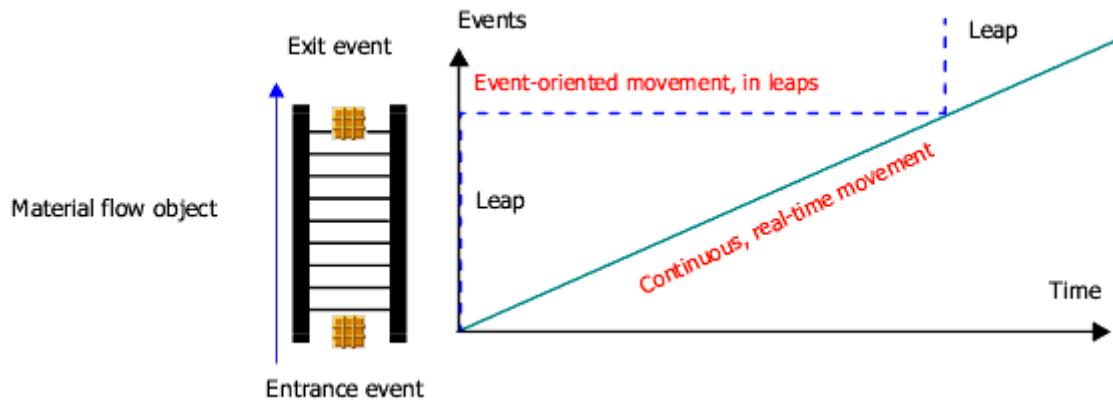
A discrete, event-oriented simulation program only inspects those points in time, at which events take place within the simulation model.

In reality, on the other hand, time passes continually. When watching a part move along a conveyor system, you will detect no leaps in the time. The curve for the distance covered and the time it takes to cover it is continuous, it is a straight line.

A discrete event-oriented simulation program only takes points in time (events) into consideration. Such events may, for example, be a part entering a station or leaving it, or moving on to another machine. Any movements in between are of little interest to the

simulation as such. It is only important that the entrance and the exit events are displayed correctly. When a part enters a material flow object, the software calculates the time until it exits that object and enters an exit event into the list of scheduled events for this point in time.

Thus the simulation time that the software displays leaps from event to event. This happens as soon as an event is processed.



**Fig. 7: Discrete Event Simulation Concept**

Fig. 7 illustrates the entrance and exit of an object on a Material Flow Object. Although the movement of the object is continuous the important events are its entrance and its exit. The simulation software makes a list of all the important events and in the list skips to the next event programmed and executes the event. The total time needed to run a simulation is really faster and is only limited by the computer speed. Of course the simulation can be slowed down to allow the user to see the simulation. Generally it is even possible to run the simulation at the same speed as in reality.

In the small example of Fig. 7, when the object enters into the material flow object the time of the exit is directly calculated as in Fig. 8. The first column indicates the type of event – here the exit of the piece – the second column the event date, the third column the piece that will execute the order and the last one the object that the piece will leave.

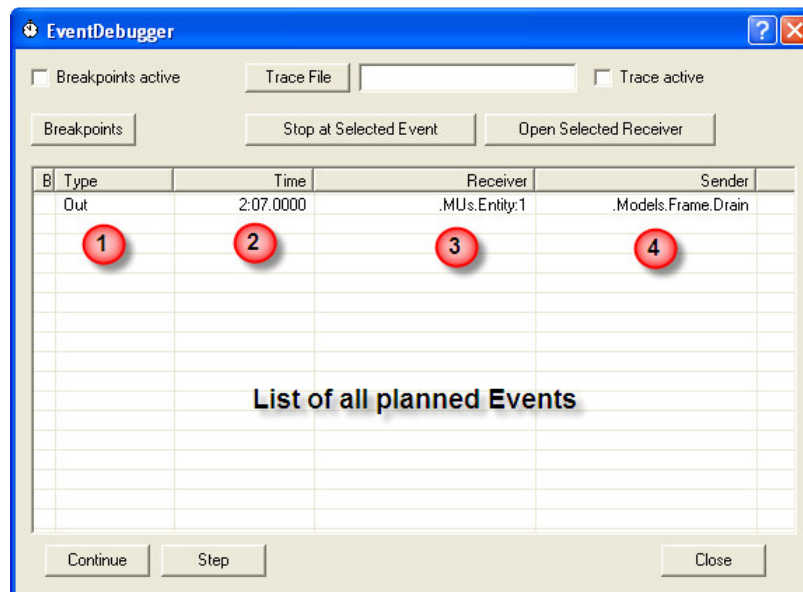


Fig. 8: List of events – simple example

- 1 : Types of events (In, Out, Creation, etc.);
- 2 : Time when the event will be executed;
- 3 : Moving Object that triggers the event;
- 4 : Receiver Object on which moving units has triggered the event.

When we have several tasks to execute, this table will contain more than one event. The simulation software executes the first one (sort by the column Time), then the second one and so on until all events are executed – see Fig. 9. If a new event appears it is automatically inserted at the right position.

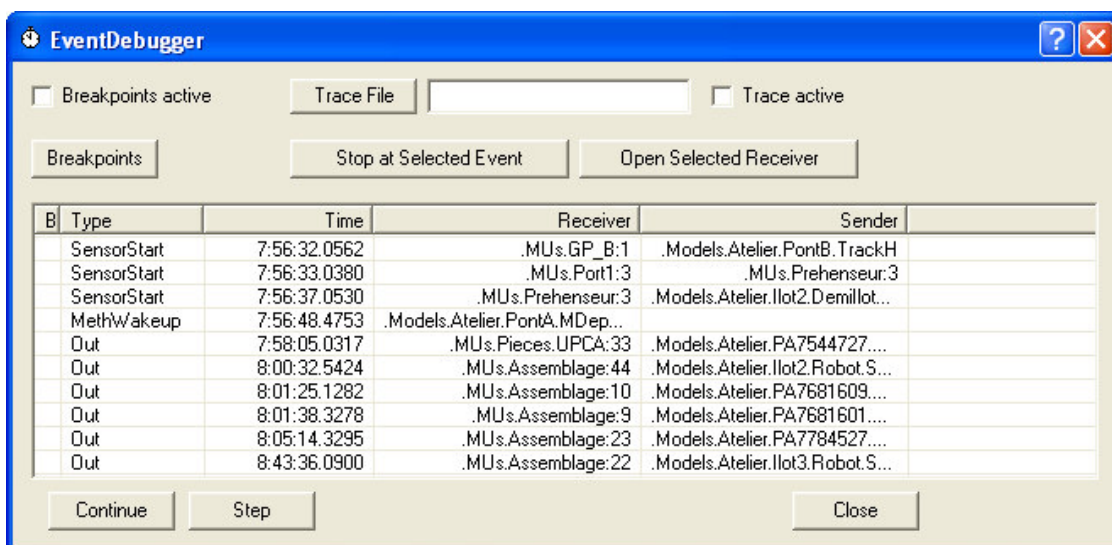


Fig. 9: List of events

Notice that continuous moves can still be represented with this method. Imagine that we have to model the movement of a piece on a conveyor. The movement is simply cut into smaller steps and each step is an event that will be recorded in the table of Fig. 9. Movement is still discrete but if we run the simulation at a reasonable speed we have the feeling that the movement is continuous.

## 2.3 Genetic algorithms

### 2.3.1 Description

In Chapter 3 an optimization of the virtual model is carried out. The algorithm chosen is a genetic algorithm. The goal of this section is not to give a state-of-art or an advanced description of Genetic Algorithms because a lot of publications/articles exist on the subject. A simple reminder will be given to allow for an understanding of the most important things to know.

Genetic Algorithms (GAs) are computerized search procedures based on the principles of natural evolution and heredity. There are many reference textbooks and papers about Gas: *Birk et al. (2003)*, *Coley (1999)*, *Davis (1991)*, *Goldberg (1989)*, *Haupt et al. (1998)*, *Man et al. (1999)*, etc.

The search space of the problem is represented as a collection of “individuals” which are referred to as “chromosomes”. A set of individuals occupying the same time is called a population. Each individual is composed of a set of values, or “genes”, which define the individual’s characteristics. Binary coding (i.e. each gene is a 0 or 1) is usually used for the chromosome. The term “fitness” is used to describe the capability of solving the optimization problem by an individual. An individual’s fitness determines the individual’s likelihood to survive and mate. The fitness is measured by a predefined fitness function. For the calculation of this fitness function the binary chromosome is converted to a real number, and employed in the function equation. The fitness function is also useful for watching how the GAs evolve better chromosomes over time. After initiation, each generation produces new children based on the genetic cross over and mutation operators. It is based on the assumption embedded in the idea of natural selection that, as members of the population mate, they produce offspring (individuals generated by the reproduction process) that have a significant chance of retaining the desirable characteristics of their parents, perhaps even combining the best characteristics of both parents. The population of variants continually evolves in response to selection

pressure described by the fitness function. In this manner, the overall fitness of the population can potentially increase from generation to generation. The same manner can also be employed to discover optimal solutions because one can start with a set of solutions which are not necessarily desirable, but the solutions obtained by combining some of the best characteristics of their parental solutions must have a higher fitness. After the evolution of generations, the optimal solution can finally be obtained. Different parameters have to be chosen to use the algorithm efficiently and it is important to describe them.

### 2.3.1.1 Mutation operator

Two different type of mutation can be used: inversion of a sequence or mutation of two single genes. For an inversion, a random inversion range is chosen and then the sequence of genes within this range is inverted.

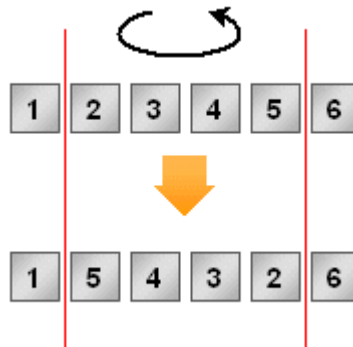


Fig.11: Mutation for an inversion task

Another mutation operator is to exchange two randomly chosen genes. The illustration below demonstrates how the operator works.

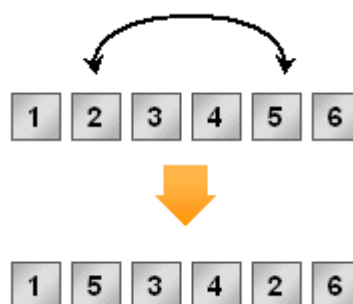


Fig.12: Simple exchange between two random genes

It is also possible to set how the genetic algorithm exchanges the positions by using a probability for exchanging each individual element or by entering a number for the exchanges.

### 2.3.1.2 Cross Over

Contrary to the mutation and inversion operators, the crossover operator is applied to two chromosomes. It exchanges elements between them. Initially two randomly intersecting points are chosen and then the ranges between these two points are exchanged.

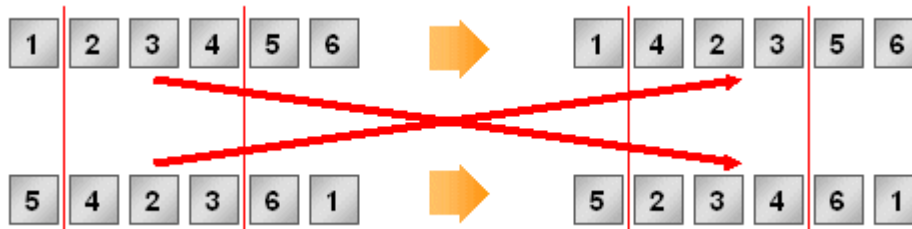


Fig.13: Cross over between two solutions

Using crossover operators will result in better solutions because there is a high probability that short good ranges are preserved and will thus be reproduced in a growing number of solutions. We differentiate between two different types of crossover operator. The operators differ in how they stress the relative and absolute position of the individual elements. While Order Cross Over (OX) preserves the relative position (neighbour relation) of the elements in the solution to each other. Partially Matched Cross Over (PMX) stresses the absolute position of the objects. So, when you use OX, small groups of solution objects are kept together while PMX separates the solution objects to preserve the absolute position of individual elements.

### 2.3.1.3 Parent selection

We can use two different parent selections: Deterministic or Random.

For a deterministic selection, the parents are selected randomly according to their fitness values (roulette wheel selection). For this reason individuals with good fitness values will be used more often as parents for creating the next generation. But individuals with a bad fitness value also have a chance to be used as parents.

For a random selection, the fitness values are not used. All individuals have the same likelihood to be used as parents.

### 2.3.1.4 Offspring selection

This parameter allows us to select family members used in the next generation. We have four different possibilities.

- Only use the two child solutions and select the solution with the better fitness value (1of2);
- Use parent and child solutions and select the best solution (1of4);
- Use parent and child solutions. It employs a stochastic selection. Selection probabilities are proportional to the quality of the solution (Prob);
- Select the individual taken for the new generation regardless of the fitness value (Random).

### 2.3.1.5 Fitness reference

After a new generation has been generated and evaluated, we need to determine the individuals of the parent generation used to create the individuals of the next generation. The fitness value of the individual proposed solutions can be evaluated by two different methods.

- *Absolute*: for this setting the fitness value of the individual solutions is going to be evaluated relative to absolute zero. At the beginning of the optimization runs, it produces good results because of the variance of the individuals in the generation. With this setting the selection pressure for homogenous generations of individuals is extremely low.
- *Relative*: for this setting the fitness value of the individual solutions is going to be evaluated relative to the worst solution of the generation of individuals. For homogenous generations of individuals an even selection pressure is maintained.

### 2.3.2 Disadvantages of GA

To be complete we need to expound here some disadvantages of genetics algorithms:

- Calculation time: compared to other metaheuristics calculations, time levels are high in particular for the evaluation of the fitness;
- They are sometimes difficult to implement: parameters such as the size of the population or mutation rate are difficult to estimate. Now evolution success depends on it and several tests are necessary;
- We are never sure that the solution found is the best one. We just know that we have approached the best solution but there is still uncertainty about how far we are;



- Another important problem is local optima. Indeed when a population evolves it is possible that some individuals that have specific characteristics become the majority. At that moment there is the possibility that the population converges towards these individuals that are maybe not the best ones. We get the heterogeneity of the population. There are some techniques to avoid this phenomenon such as adding to each generation new individuals created randomly.

## 2.4 Presentation of the Aker Yards France shipyard

### 2.4.1 Introduction

During this entire project “*Les Chantiers de l’Atlantique*” has been taken over by different investors, becoming firstly “*Alstom Marine*” before to be finally taken over by Aker. Actually the correct denomination of the shipyards is thus Aker Yard France. General information is directly given in the website of the company – <http://www.akeryards.com>.

Aker Yards ASA is an international shipbuilding group focusing on sophisticated vessels and being one of the world's largest shipbuilders. The group is organised through three business areas; Cruise and Ferries, Merchant Vessels and Offshore and Specialized Vessels. Aker Yards comprises 18 yards in Brazil, Finland, France, Germany, Norway, Romania, Vietnam and Ukraine with some 20,000 employees.

Some key figures:

- 18 shipyards in Brazil, Finland, France, Germany, Norway, Romania, Vietnam and Ukraine;
- Operating revenues 2006: EUR 3 212 million;
- Order backlog 2006: EUR 5 949 million;
- 20 000 employees;
- Listed on the Oslo Stock Exchange (ticker: AKY).

Aker Yards is Europe’s largest shipbuilding group and one of the world’s four largest shipbuilders. Customers worldwide rely on Aker Yards to design and build future-oriented, innovative, market-tailored, and competitively priced vessels. Sophisticated vessels for the demanding markets for cruise ships, ferries, merchant vessels, as well as offshore and

specialized vessels, are the core of the extensive product range built at the Group's 18 shipyards in eight countries.

Aker Yards is organised through three business areas; Cruise and Ferries, Merchant Vessels and Offshore and Specialized Vessels. The Group's acclaimed arctic marine technologies, advanced vessel technology applications, availability of appropriate yard capacity, and modular new building methods all add value. With a highly skilled workforce, modern assembly yards, state-of-the-art design resources, in-depth knowledge of our customers' markets, and a honed ability to listen to customer needs, Aker Yards stands for excellence at sea.

**Aker Yards France** is part of the Cruise & Ferries business area of Aker Yards. Aker Yards SA includes three subsidiaries, Aker Yards Lorient SAS, Aker Yards Cabins and Aker Yards Solutions.

The shipyard in Saint-Nazaire and Lorient offer a wide range of high-added value vessels ranging from 30 to 300 metres long – or even more. Aker Yards, Saint-Nazaire is capable to build post-panama size vessels, while Aker Yards, Lorient is specialized in smaller-size ships.

## **2.4.2 Company's workforce**

The shipyard in Saint-Nazaire is using 2880 persons. 1730 on 2880 employees have been arrived in the shipyard since 1998. This represents 60% of the personal:

- Engineers and executive: renewed at 65%;
- Technicians and supervisor: renewed at 57%;
- Workers renewed at 60%;

Average age of workers is down to 40 years in 2005.

## **2.4.3 Industrials means**

The shipyard is one of the most important in Europe. Main characteristics are:

- A dry dock of 900 meters long and 70 meters width overcome by a crane bridge with a capacity of 750 tons;
- A Pre-Montage area of 1200 meters long next the dry dock, with a 650 tons crane bridge;

- A dock of 450 meters long and 90 meters width;
- Two fitting out docks of 400 meters long each;
- Robotized and automated workshops (machining, Pre-Pre-Fabrication);
- Pre-Fabrication workshops with 2 panel lines strongly automated.

Total capacity: 60000 tons by year.

## 2.4.4 The production

Aker Yard France is specialized on ship with a high added-value:

- Passenger ships (ferries, cruise liner as panamax and post-panamax). In December 2003, Aker Yards France delivered to Cunard the *Queen Mary 2*, the last transatlantic built by the shipyard after the mythic *France* or *Normandie*;
- Gaz carriers: in particular the biggest of the world – *Provalys* and *Gaselys* – built for “Gaz de France” and delivered respectively in November 2006 and March 2007;
- Military ships: for instance parts of BPC *Mistral* and *Tonnerre* for the National Marine;
- Ships specialized into the research as the “*Pourquoi pas ?*”, oceanic research ship delivered at Ifremer in 2005.

## 2.4.5 Main Workshops of the Shipyard

### 2.4.5.1 Introduction

To build a block the production requires dividing it in different parts. At the beginning parts are very small to become more and more complex and huge. These parts are classified depending on their size: smallest pieces are simple plates or stiffeners, largest ones are blocks. Hierarchy of pieces is given in Fig. 10.

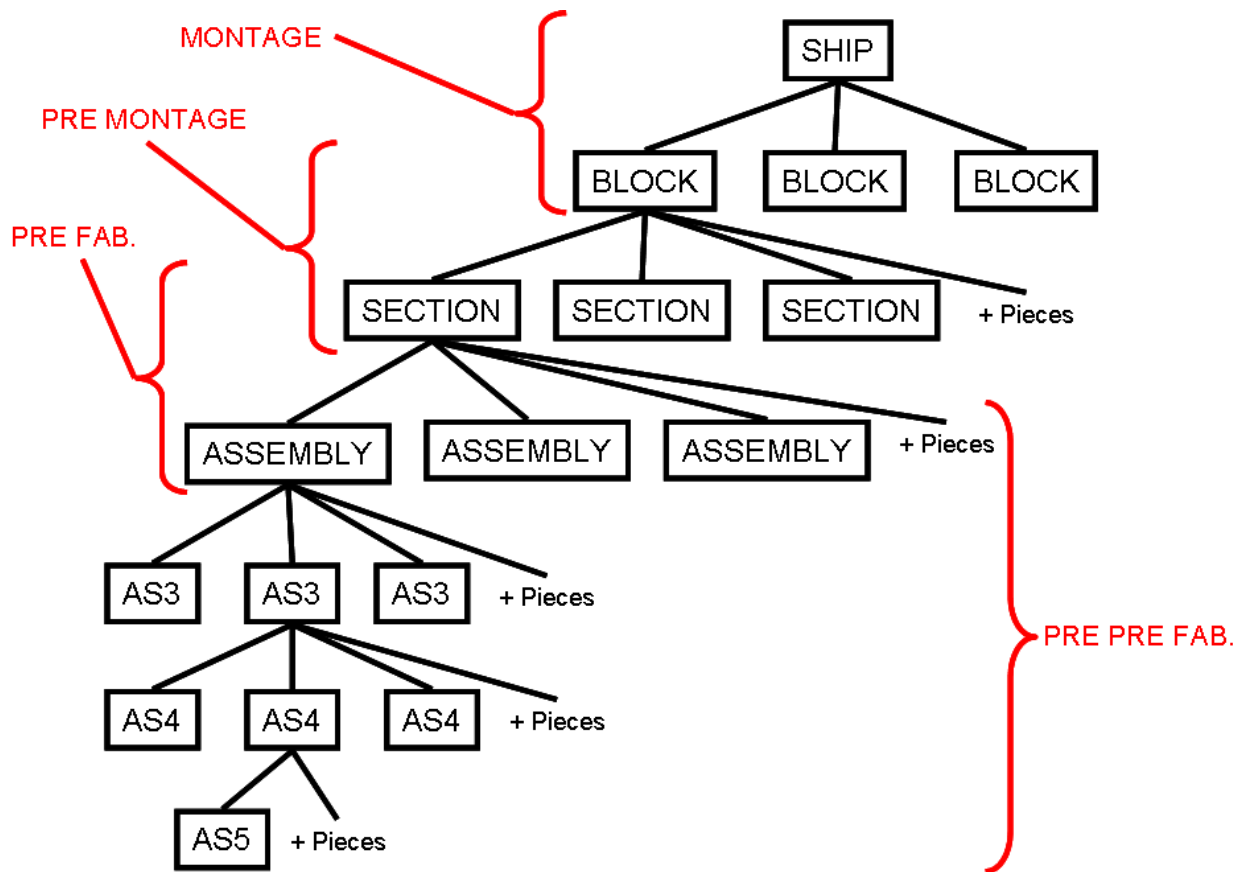


Fig. 10: Assembly sequence of pieces

Main workshops of the shipyard are the following ones:

- Machining of plates (UFU);
- Machining of short profiles (UPC);
- Machining of special profiles (UPS);
- Fabrication of reconstructed welded profiles (PRS);
- Workshop of forming and coiling;
- Workshop of PrePreFabrication (*Pre Pre Fabrication*);
- Panel Lines (*Pre Fabrication*);
- 120 Tons Workshop (*Pre Montage*);
- 180 Tons Workshop (*Pre Montage*);
- Area of Pre Montage (*Pre Montage*);

The final operation is the joint of blocks on the ship (*Montage*). Fig. 10 shows that workshops are classified in four categories: those related to individual pieces, *Pre Pre*

*Fabrication, Pre Fabrication, Pre Montage.* The final operation is called *Montage* – but is not really carried out in a “workshop”.

Next sections will briefly describe workshops of the shipyard. For confidential reasons the descriptions will remain rough. Nevertheless the PrePreFabrication workshop is the workshop chosen for the simulation. A more detailed presentation is given in section 3.2.2.

#### **2.4.5.2 Machining of plates (UFU)**

In this workshop plates are cut with a plasma machine.

Outside of the workshop, the following tasks are achieved: heating → shot blasting machine → painting → drying → identification of plate (marking).

Inside of the workshop: imbrication → identification of pieces (marking) → plasma machine → sorting → finishing.



**Fig. 11: Machining of plates**

#### **2.4.5.3 Machining of short profiles (UPC)**

Outside of the workshop, the following tasks are achieved: storing → drying → identification of profiles.

Inside of the workshop: imbrication → identification of pieces (marking) → Plasma machine → sorting (robot).



**Fig. 12: Machining of short profiles**

#### **2.4.5.4 Machining of special profiles (UPS)**

Inside of the workshop, the following tasks are achieved:

- Punching → cutting → forming → storing;
- Welding machine → straightening machine.



**Fig. 13: Machining of special profiles**

#### **2.4.5.5 Fabrication of reconstructed welded profiles (PRS)**

Inside of the workshop, the following tasks are achieved: cutting of the web (laser) → cutting of the flange → welding machine → turning over → shot blasting machine → painting → drying → slitting → covering.



Fig. 14: Fabrication of reconstructed welded profiles

#### 2.4.5.6 Workshop of forming and coiling

Inside of the workshop we can find the following resources: jib crane, mechanical bender (3000 tons), press and bending machine.

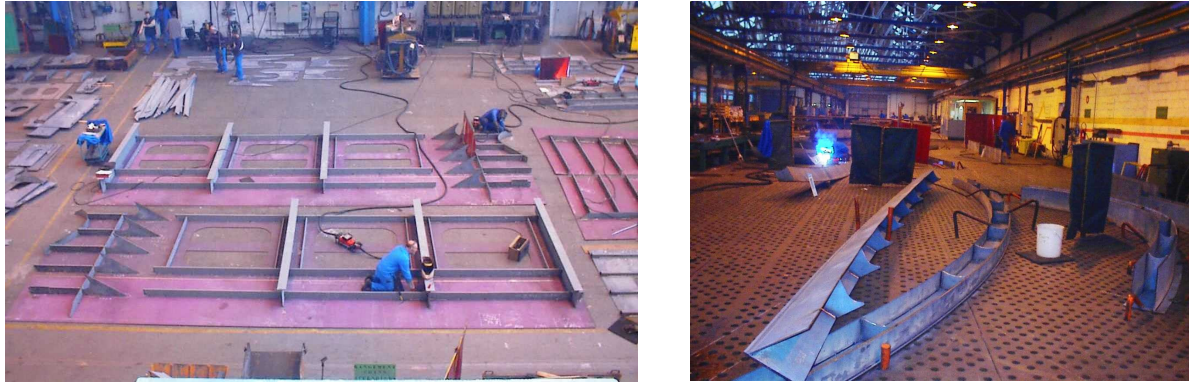


Fig. 15: Workshop of forming and coiling

#### 2.4.5.7 Workshop of PrePreFabrication

This workshop is divided into two workshops: the “old” PrePreFabrication and the new one (strongly automated). The first one is called the “manual” PrePreFabrication workshop and the second one the “automated” PrePreFabrication workshop.

The new workshop will be modelled with simulation tools in the framework of this thesis – see Chapter 3. More details about its functioning in the chapter.



**Fig. 16: Workshop of PrePreFabrication**

#### **2.4.5.8 Panel Lines (Pre Fabrication)**

This workshop is divided into 2 lines (a new and an old one).

On the first line the following tasks are achieved: flame cutting → planing machine → welding machine → turning over → grinder → installation of longitudinal girders.

On the second line: flame cutting → planing machine → welding → grinder → installation of longitudinal girders.



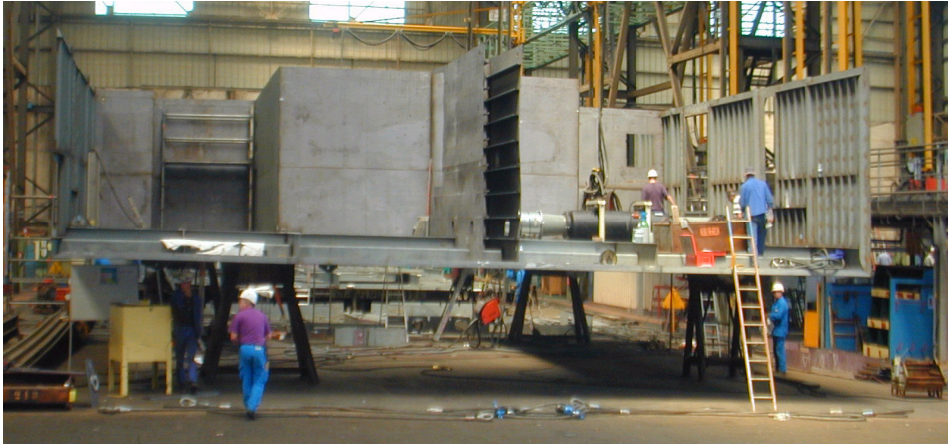
**Fig. 17: Panel Lines**

#### **2.4.5.9 120 Tons Workshop (Pre Montage)**

In this workshop assemblies are welded together to make block. The workshop is divided into 8 areas (naves). In each nave, we have five welding arms (of 2 stations).

The workshop is used in Chapter 5 to test the tool developed to solve space allocation problems.





**Fig. 18: 120 Tons Workshop**

#### **2.4.5.10 180 Tons Workshop (Pre Montage)**

The workshop has the same function that the 120 Tons Workshop but can realize bigger blocks. There are only four naves that contain each twelve welding arms (of 2 welding stations).

The workshop is used in Chapter 5 to test the tool developed to solve space allocation problems.



**Fig. 19: 180 Tons Workshop**

#### **2.4.5.11 Area of Pre Montage**

This is not really a workshop this is an area (just before the dry dock) where blocks are welded together before to be fixed in the dry dock.

Here again the workshop is used in Chapter 5 to test the tool developed to solve space allocation problems.

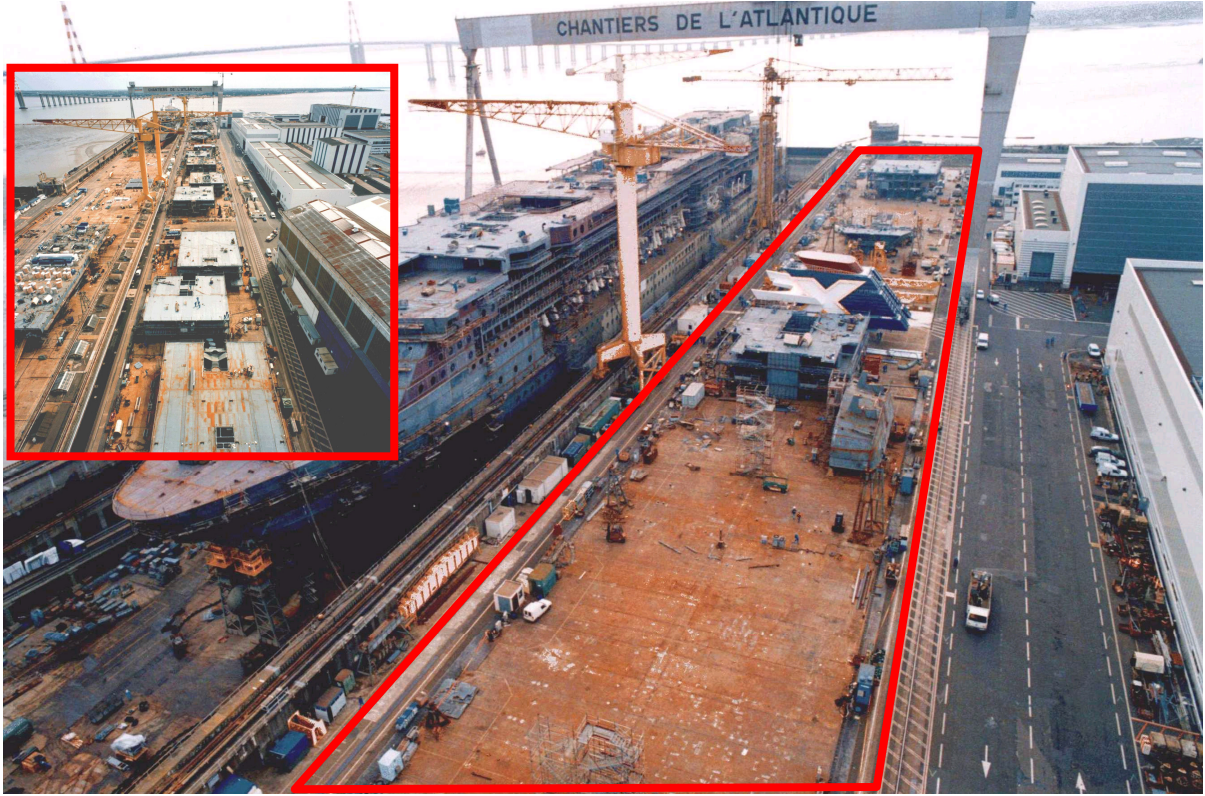


Fig. 20: Area of Pre Montage

# Chapter 3

## Development of a Virtual Workshop

### 3.1 Introduction

The framework of this thesis is to develop tools to optimize production in shipyard workshops. To improve the workshop's productivity, several methods could be used. In this chapter we develop one: the use of production simulation. This complex method is generally used when analytical methods could not be applied.

We will apply the method on a specific workshop of the Saint-Nazaire shipyard: the PrePreFabrication workshop. The objectives of this chapter are:

- To model completely the workshop;
- To show the potential information that a simulation model could offer;
- To improve the production by linking the simulation model to an optimization tool.

The chosen workshop is one of the most complex due to its completely unusual way of functioning.

At the first step the workshop is modelled on a computer with a level of details as precise as wished/needed. Once the model is developed we can perform a deep analysis of the workshop. This analysis underlines first the weakness of the system and after brings solutions! For instance we can have access straightaway to the relative use of resources. Bottlenecks are clearly highlighted. In consequence the planner sees where the problems are and tests solutions on the developed tool. This is probably the main interest of a simulation model: tests and experimentations can be done on the virtual model contrary to the real workshop. Production never needs to be stopped to perform several experiments/benchmarks!

The second step concerns the optimization of the production sequence. This has been done by linking the simulation model to an optimizer – a genetic algorithm.

In Chapter 4 we will develop a tool to create, handle and optimize PERT<sup>3</sup> diagrams. These diagrams give important information about the planning, margin etc. The model developed in this chapter will also be linked to the PERT tool in order to give new important information about the workshop.

The present chapter is divided into five main sections. The first one is a detailed presentation of the workshop studied. The second one is the database treatment. Indeed the simulation requires specific data and this data needs to be treated. The third section is the description of the simulation software and its characteristics/possibilities. The fourth is the development of the PrePreFabrication model. Finally the last one is the analysis of a realistic problem to show potential of the simulation.

## **3.2 Virtual Model of the PrePreFabrication Workshop**

### **3.2.1 Introduction**

The introduction and state-of-the-art summary are given in Chapter 2. As explained in the introduction of this chapter the workshop modelled is named the PrePreFabrication workshop. The workshop is a welding workshop and exit products are small assemblies constituted of a basic plate on which are welded several girders. As we will see, the production line is not a classical one where pieces go from a machine to another one until the end of the product. This characteristic is very important because it will make the modelling and the optimisation much more complex than in a classic workshop. In other words if simulation and optimization are efficient on this workshop we have high hopes that any other workshops could have significant benefits by using the production simulation: the complexity of a workshop must not be a brake to developing a simulation model!

The first section 3.2.2 describes in detail the workshop studied and its characteristics and the second section 3.2.3 concerns the database treatment. Indeed the simulations required lots of different information. This information is contained in the shipyard's huge database but not necessarily in a convenient way. The goal is to extract the required data from a different database and then to treat it to be directly used for the simulation. Most of the database treatment is done in Access by SQL<sup>4</sup> requests and VBA<sup>5</sup> macros. The third section

---

<sup>3</sup> *Program (or Project) Evaluation and Review Technique*

<sup>4</sup> *Structured Query Language*

3.2.3 is related to the description of the software which is used. In our case we have used *Plant-Simulation* software. Other simulation software exists but they all have the same methodology. Differences are located in the functionalities offered: possibilities of visualisation, object libraries, etc. The fourth section 3.2.5 is the development of the virtual workshop itself. Finally we finish by analysing a realistic application in section 3.2.6.

An important step is to link the simulation model to an optimization tool. The second part of the chapter – section 3.3 – treats this aspect by linking the model with a genetic algorithm.

## 3.2.2 Detailed presentation of the workshop

### 3.2.2.1 Products and equipments

The workshop studied in this chapter is the Automated PrePreFabrication Workshop<sup>6</sup>. It has been briefly described in Chapter 2 and we will continue the description further in this section.

Products of the workshop are small assemblies constituted of a plate on which stiffeners are welded – see Fig. 21. For the Saint-Nazaire shipyard there are two workshops that build these small assemblies. The first one is completely manual and the second one is automated – for the welding. Automation of welding is only possible on bigger assemblies with a lot of girders and a high welding length. On Fig. 22 we can observe small assemblies that are not built in the Automated PrePreFabrication Workshop due to their complexity or due to their dimensions. In this work we have only modelled the *Automated* workshop and not the *Manual* one. Note that modelling manual operations is possible – we will do this for some operations in the automated workshop – but it is a very delicate and sensitive task due to the high degree of randomness of these operations! Results are obviously much more reliable when they are related to an automated task.

---

<sup>5</sup> *Visual Basic for Applications*

<sup>6</sup> *For convenient reasons, the term Automated will be skipped in the entire chapter*



Fig. 21: Typical assemblies

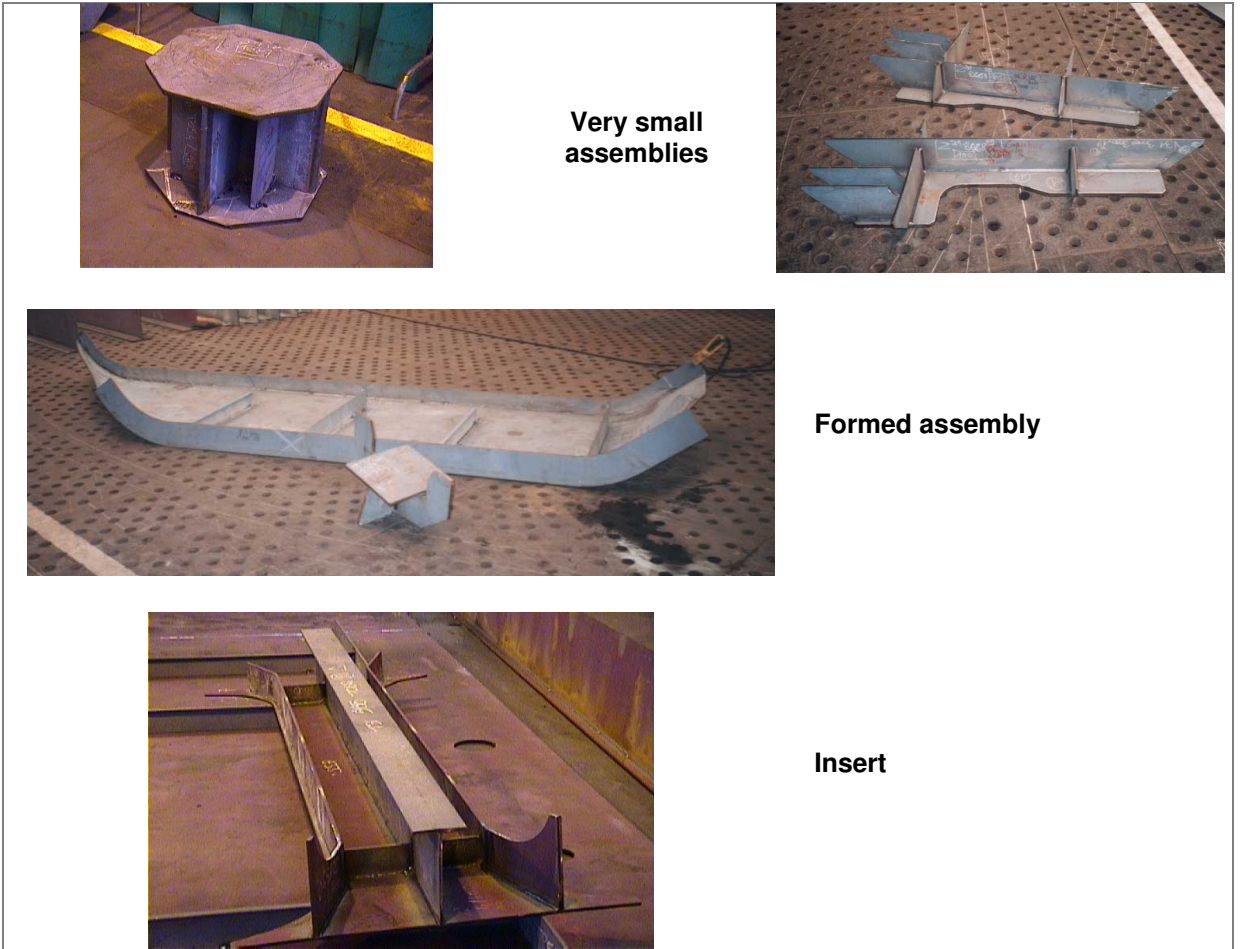


Fig. 22: Non automatable assemblies

The main characteristic of the workshop is that each assembly is slightly different – by their dimensions, numbers of stiffeners, etc. The constituents are ranked in two parts depending on their weight: the heavier parts can only be handled by a crane bridge. There are also different kinds of girders to build an assembly:

- UPS: *abbrev.* of “*Usinés profilés spéciaux*” meaning “Special stiffeners”. These parts are divided into long UPS (dim > 3 meters) and short UPS;
- UPC (A and B): *abbrev.* of “*Usinés profilés courts*” meaning “Short stiffeners”. These parts are divided into long UPC (dim > 2 meters and called UPCA) and short UPC (called UPCB). Notice that each UPC (even UPCA) can be handled without the crane bridge. This part is not classed in the heaviest parts;
- PRS: *abbrev.* of “*Profilés reconstitués soudés*” meaning “Welded reconstituted stiffeners”.

Plates are also divided into two categories: large (GTole – *abbrev.* of “*Grandes Tôles*” meaning “Big plates”) and small plates (PTole – *abbrev.* of “*Palanqué*”). Note that there are also very small pieces such as brackets etc.

An important characteristic of the workshop is its symmetry. We can almost consider that we have two workshops with their own entries and exits. This separation is very important because it has a deep impact on the production sequence. Indeed at the exit the finished assemblies are stored on containers. They are not stored randomly: an exit container is booked for a type of assembly. Indeed assemblies belong to bigger assemblies – that are in fact part of panel elements – and assemblies produced in this workshop are stored and evacuated according to these bigger assemblies. By convention in this thesis we will call these bigger assemblies sub-panels. Assemblies stored on exit containers all belong to the same sub-panel! This indirectly implies another constraint in the sequence of production: kits of the same sub-panel must be built in the same side of the workshop!

The workshop’s resources are: two cranes bridges (one for each side), workers, eight mechanized grippers (four for each side) and four completely automated welding robots. Views of the workshop are given at the Fig. 23.

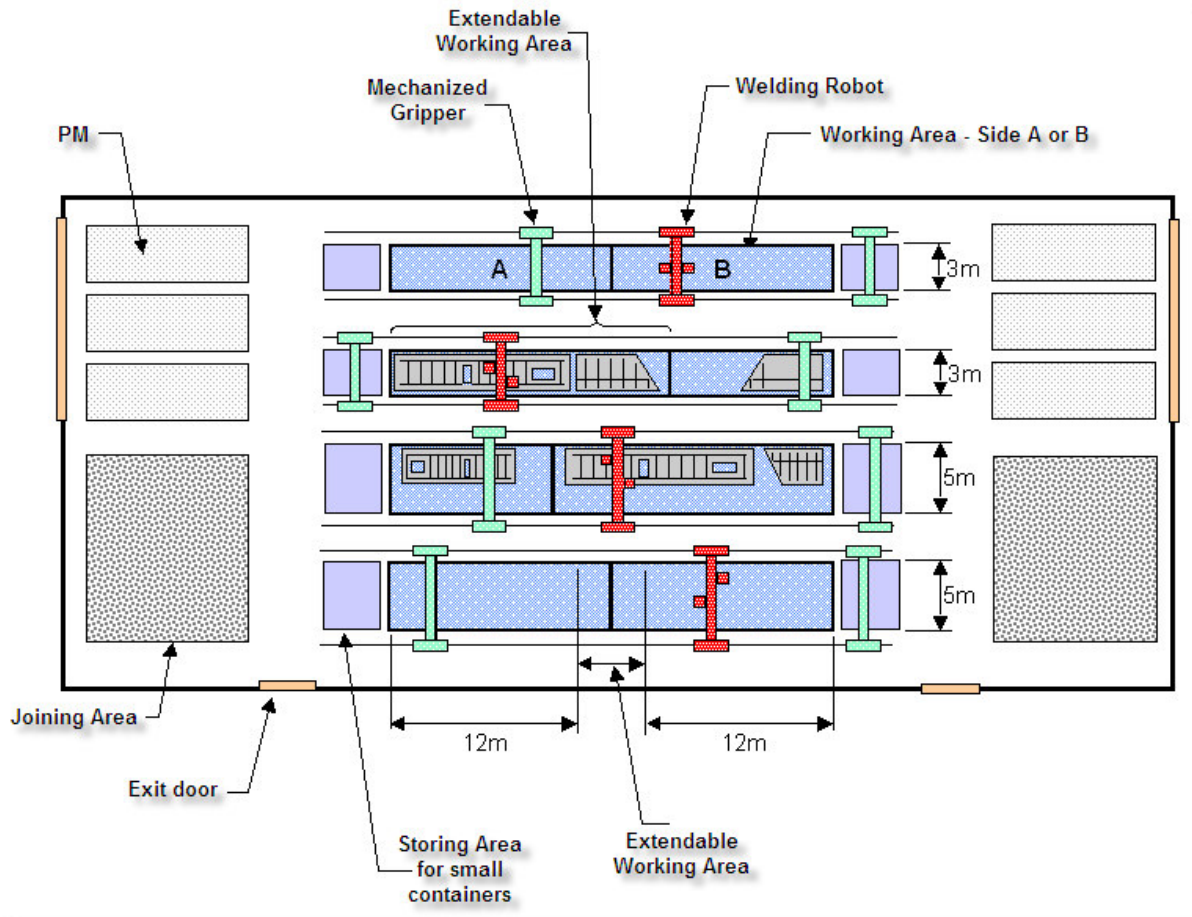


**Fig. 23: Views of the PrePreFabrication Workshop**

The workshop contains four working areas (that we will call *cells*) and each of them is divided into two parts – one for each side of the workshop. On these parts (that are named *half-cells*) the following production sequence is carried out: tacking – welding – finishing. A schematic view of the workshop is given in Fig. 24. The two bottom cells are wider than the two upper ones. This point will be important for the sequence optimisation because some assemblies could only be done on these larger cells. The constraint is quite important.

The workshop also contains two joining areas. These areas are used to join basic plates together before to be put them on a working area. These welding operations are done manually by workers.



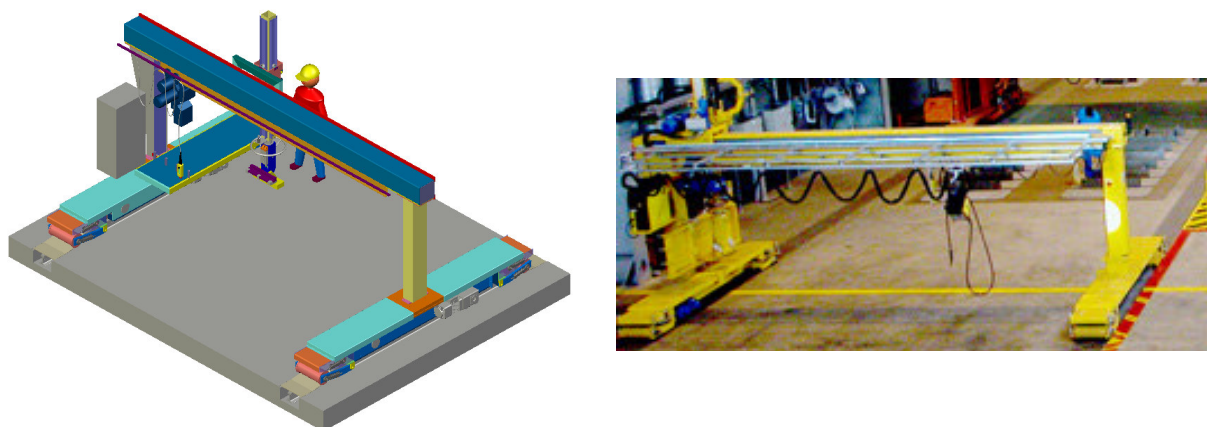


**Fig. 24: Schematic view of the workshop**

The main resources are described in the following sections.

a) The mechanized gripper

The tacking operation is done either manually or with a mechanized gripper – Fig. 25.



**Fig. 25: Mechanized gripper**

The mechanized gripper is a gantry crane constituted of one magnetic gripper arm used to position stiffeners (300 kg) and of a polyvalent hoist (2.5 t) to handle small plates or containers. Two kinds of tools can be fixed to the hook of the hoist: a sling system with

Chicago grips or an electromagnet system more powerful than those used on the gripper arm. If the weight of the piece is less than 15 kg, a worker can handle it manually without the mechanized gripper.

In addition mechanized grippers are equipped with a welding torch to facilitate the tacking work. Only one worker can use it at the same time.

#### b) The crane bridge

Two crane bridges, one on each side, bring input pieces from the entry zone located in the extremity of the workshop to the work area of the corresponding cell. For security reasons, the crane bridge cannot bring pieces above workers, so when it takes a piece from the entry zone, it has to go to the far end of the workshop, to do its vertical movement and then to do its horizontal movement to the work area. Crane bridge speed and accelerations are set in the model like in the real workshop. Sometimes the pieces needed are not on the top of the entry container, so workers have to put an upper piece in another area (at the bottom of the workshop, to the extreme right or left depending on the side we are) to create a new pile of pieces.

The crane bridge is also used to clear out finished assemblies and to evacuate them on one container of the exit (two containers – called PM – are used for the evacuation for each side). Assemblies can be classified in different groups depending on the larger assembly to which they belong (that we will call *panel*). So at the exit, only the same assemblies are stored on the exit containers. We have two different exit containers, so we can have two different groups of assembly at the same time in the workshop. Unfortunately, it will not always be the case so when we have an assembly to clear away and if there is no free container, we store it in a temporary area (just before the exit containers).

Depending on the kind of pieces to bring, the crane bridge uses either a magnet system for flat pieces, or slings for more complex pieces. Thus for the simulation a set-up time has to be modelled for each change of manufacturing tool, and this set-up depends on the number of slings to be put on the crane bridge.

#### c) Workers

The workshop is not completely automated and workers are still very important. There are mainly four pools of workers. The first and the second ones are working respectively on the first two and on the last two cells. The third and the second one are working in the joining areas.

Workers on working areas are responsible for two operations. The first operation is the tacking operation. First pieces are brought – either manually for small pieces or with the help of the mechanised gripper for heavier ones – on the cell and then tacking is done. The second operation is the finishing. As we will see, welding on working areas is done by welding robots but these automated robots cannot make all welds! As a consequence workers have to come after the automatic welding to finish manually the missing welds. Some mending operations also have to be done.

In the joining area workers are used only for the welding of plates: the goal being to build the basic plate to put on a working area. These plates are welded on the two sides. It means that once a side is welded we have to wait for the crane bridge to turn the plate over. The welding can then be done on the opposite side.

Note that there are some other workers in the workshop. Of course the crane bridge is not automatic and we need an operator to control it. To start the welding robot a worker is needed. Its role is to program the robot and to check if everything is going right – and eventually to act if problems arise.

#### d) Welding Robot

This is the only tool which is shared between the two sides of the workshop. It is a completely automatic tool, which is used to weld pieces that have been tacked before either manually or by the mechanized gripper. Practically, the total time for the welding is calculated by special software which knows the constraints and limitations of the welding robot, and the geometric characteristics of pieces to be welded. The robot has two separate arms that can work simultaneously to reduce the welding time.

### **3.2.2.2 The notion of Kit**

For the input, there is a list of assemblies which have to be realized. In fact, there are two different lists, one for each side. These assemblies are grouped and constitute a kit. A kit can contain only one assembly, but it may also be four or five small assemblies. All the assemblies of one kit will be done on the same half-cell at the same time. Our goal is to find the best sequence of all these kits to minimize the total time of production. Currently these kits have to be defined before by the user, and they cannot be changed. It could be interesting in further research to try to create these kits by an optimization tool.

For the simulation, the sequences of these kits are inputs but using optimization it would be possible to find the best sequences which give – on average – the shortest production time.

There are two different ways to use the simulation and to define the sequence: it could be either a “half-workshop sequence”, or a “half-cell sequence”.

In the half-workshop sequence, at the beginning two sequences of kit are selected (one for each side) which correspond to the order of the production of the kits. As soon as a half-cell is free, the next kit in the list is allocated to this free half-cell. Thus it is not known at the start of the simulation where kits will be done. Plates are stacked at the entry on containers (PM) according to the order of this sequence. The advantage of this type of sequence is thus avoiding using unnecessarily the crane bridge to stack up plates from PM to a temporary area because the plates we need will always be on the top of the stack. Unfortunately the major problem is balancing each side of a cell in order to saturate the welding robot. Indeed this operation is the longest so it is really important to avoid down time as far as this tool is concerned. With this sequence type it is very difficult to reach a good balance.

In the half-cell sequence, each kit receives its destination (half-cell) at the start of the simulation. So we can try to put kit on the left and on the right of a cell which are very complementary to saturate the welding robot. The problem of this method is managing breakdown. If there is a problem or a delay in a cell, it could decrease strongly the productivity if it is expected to keep absolutely the sequence predicted.

Obviously each type of sequence has its own advantages and disadvantages, and simulation can predict which one is the best for a given scenario.

By a cell there are two workers (not including the worker managing the welding robot, and the one moving the crane bridge). They order smalls containers (which come in the STK area) and prepare the tacking (i.e. read plans,...).The first plates to bring are called basic plates (one for each assembly of the kit), and they generally come from the entry PM. When they are in place, the crane bridge brings others plates and girders, pieces which are directly tacked with the mechanized gripper. After that, workers bring pieces from the small containers to tack them on the assembly (one using the mechanized gripper, the other tacking manually smallest pieces). Possibly the crane bridge can bring sub-assemblies (already produced in the workshop) which are a part of another assembly. When all the pieces are tacked, we have to wait for the welding robot to finish welding the other half-cell. When the robot is free, it comes and starts welding. The details of the robot movements are not

modelled, and in the simulation we just see it staying on a sub-assembly during a time calculated by a simplified formula. Unfortunately, the welding robot cannot weld everything so workers have to come back again to finish the missing welds. This finishing is – in a first approximation – calculated as a fraction of the robot welding time. Finally, the finished assemblies are cleared away with the crane bridge on the exit containers.

However in practice it seems that is not possible to work with a Half-cell sequence. The reason is that the workshop has too many unforeseen events to use it efficiently. Thus we have a list of kits to build and when a working area is free we put the next kits in this place – except if we are on a small cell with a wide assembly.

The sequence of the process is thus: Preparation (1) – Tacking (2) – Welding (3) – Finishing (4) – Clearing away (5). Workers have to alternate work on each side of the cell (tacking in one side followed by finishing on the other one). So to saturate the welding robot, we have to satisfy the following scheme – where each number corresponds to the time to execute the corresponding operation:

Left Side		Right Side
1 - 2		
3	≥	1 - 2
4 - 5 - 1 - 2	≤	3
3	≥	4 - 5 - 1 - 2
4 - 5 - 1 - 2	≤	3
3	≥	4 - 5

— : Kit 1

— : Kit 2

— : Kit 3

— : Kit 4

— : Kit 5

Fig. 26: Condition to saturate the welding robot

The sequence described here above was the theoretical approach. But in practice workers do not respect it exactly: when a kit is on a cell – for welding – if workers have nothing else to do, they start the tacking of the following kit next to the current kit! At the end of the cell we generally have enough space to start the tacking. So when the “central” kit of a half-cell is finished the crane bridge comes to clear it away. Immediately afterwards the crane bridge comes again to transfer “external” kit near the centre of the cell. Then this kit could be

welded once the welding robot is free. Another kit can be tacked on the “external” area! In other words to the five described operations we can add a new one: the transfer operation. Note that this operation could be skipped. Indeed if a kit requires a lot of space it is not always possible to start the tacking of another kit at the extremity. In that case we have to wait for the kit to be cleared away before starting the next kit as usual. It should also be noticed that the “central” kit is never cleared away if the “external” kit is not completely tacked. This is to avoid calling on the crane bridge at two different moments.

The model developed is very detailed as we can see in the list below (non exhaustive) that shows the parameters introduced in the simulation:

- supply and evacuation times (different for each container);
- set-up time, speed, acceleration of the crane bridge;
- set-up time, speed, acceleration of the mechanized gripper;
- tacking time by meter (different if done manually or with the mechanized gripper);
- take down and up time (for crane bridge and robotic gripper);
- maximum capacity of each container;
- ...

Each time introduced in the model is an average value (with a distribution, variance and bounds associated) and can be changed easily.

Each piece that has to be welded in the workshop has been precisely modelled with all its characteristics (weight, length, wide, etc.). The attributes of each piece can easily be observed by clicking on the piece. All these characteristics come directly from an Access database which has been modified to facilitate data exchange with Plant-Simulation software. The next section explains the information required from the database and the required pre-processing.

### 3.2.3 The database treatment

#### 3.2.3.1 Introduction

To run the simulation model a lot of detailed information about the assemblies is required. All the information needed is available in the shipyard's database but filters have to be done to work only with the needed data. This section explains this database treatment.

What kind of database are we using? To build a ship the amount of data to be treated is obviously colossal. All these data could be divided into different categories and treated by different kinds of software:

- CAD – Computer Aided Design: This kind of software is used to facilitate the design hull and the 3D structure modelling;
- PDM – Product Data Management: It can help to treat all geometrical data, maps etc. Parts of the ship are classed into a hierarchy containing different detail levels;
- CAM – Computer Aided Manufacturing: Software used to facilitate the manufacturing – for example nesting software, machining software, welding software, etc;
- ERP – Enterprise Resource Planning: They allow to manage planning and scheduling.

These categories are most of the time strongly linked and various information is transferred between software.

In our case what kinds of information are required for the simulation? Here again two kinds are required. First of all information about the workshop! We must know its functioning and have access to the main production parameters: mean time for tacking, speed of the crane bridge, number of workers, etc. These data are related to the production. They do not enter into the categories cited above but some information can nevertheless come from the CAM. Indeed if we have automated machines they need to be programmed by CAM tools. The software can sometimes give interesting results concerning the estimation of production time! These data are vital for the simulation. In the workshop to be modelled, there are some automated welding robots. Programming of these robots is done with CAM tools and is called PHL – “*Programmation Hors Ligne*” in French. In particular results given are welding time for the assembly. This information is very important and is used in the simulation. We

consequently have to import data from the CAM tool into our simulation model. This data importation is direct and does not require a lot of database treatment. This topic will not be discussed in this chapter but in section 3.2.5.18.

Secondly information about products is needed. Once the workshop is modelled we have to simulate a production sequence of pieces. Geometrical information is required! Unfortunately we cannot do a simple extraction of data and use it directly in the simulation. Each piece/assembly data must be treated to give more than just geometrical dimensions. For instance we need to know the welding length of girders. Or the kind of container that will bring the piece! That is an important point because pieces do not come together by the same way into the workshop. And in addition there is no direct information about this in the database. However we can guess it by analysing some parameters: for instance the workshop where the piece has been built, or we also know that long pieces cannot arrive in small containers. With a treatment of the database we can find missing data. This operation is not trivial and will be explained in this chapter.

Data coming from ERP software can also be useful for the simulation. Indeed we have to model a fabrication sequence and this sequence is part of a global process: the building of the ship. We consequently have time constraints – bottom decks must be built before top decks. To take into account these time constraints we need information that comes from the ERP software. For the modelled workshop, this information is taken from a data base called BDRO<sup>7</sup>. As for the PHL this extraction is not very complicated and does not require a lot of developments. Consequently the importation will not be described in this chapter but in section 3.2.5.18.

The object of this chapter is to explain in detail which data must be furnished to make the simulation and how they are modified by Access requests.

The data that will be treated are geometrical data. In the Aker Yards French shipyard all geometrical information is stored in a huge database called MARS. Most workshop schedulers have access to it and can carry out extractions to get information relative to their workshop. For the PrePreFabrication workshop information about assemblies is needed. Of course we need the list of assemblies to be built but also all the pieces that constitute them! The MARS database contains this information but also information that we do not need.

---

<sup>7</sup> ERP database used by the Aker Yards Shipyard



Consequently we need to extract from MARS only the required information. Notice that the simulation requires specific information that is not clearly given in the database. For instance we need to know in what kind of containers pieces arrive. This information is given nowhere. On the other hand each piece has an attribute that indicates its workshop fabrication. With this information and its geometrical dimensions the relevant container can be defined! Detailed explanations will be given later.

From the shipyard we receive different tables from the MARS database. Two main tables are given: the *STEELM* table and the *STEPRT* table. These two tables are extractions of two homonym tables but with only few assemblies – not all the assemblies of the ship! They give information about individual pieces that constitute assemblies and that must be produced in the workshop. The list of assemblies is an entry data of the simulation – eventually grouped in kits. That list is placed into a table named *Kit*. Other tables must be given by the shipyard and will be explained in detail in the next sections.

The main result of the treatment is the table *YL\_KIT\_TABLE*. Linked to the table *Kit* this table contains all the information on pieces of kits. Requests to get this table are numerous and only the main ones will be explained in this chapter.

### 3.2.3.2 The Kit Table

The table contains the list of assemblies to be built in the workshop. The main objective of the simulation is to estimate the time needed to build them and in a second step to find the best sequence to minimize the total production time. The table below gives an example of a *KIT* table with some assemblies:

HNR_ID	NO_KIT	ELM_NO	NOPAN	IND	AS3	AS4	X_LOCAL	Y_LOCAL	SOUR	GROUPE	NBR_RAB	KIT_SCM
A33	495	7520269	2210	04	201		0	0	S1	4	2	1TM
A33	499	7520329	2210	04	304		0	0	S1	4	2	1TQ
A33	690	7544389	2212	04	203		0	0	S1	4	2	2ES
A33	870	7311683	6304	01	309		0	0	S1	4	0	2RG
A33	870	7311685	6304	01	313		0	0	S1	4	0	2RG
A33	870	7311686	6304	01	315		0	0	S1	4	0	2RG
A33	870	7311687	6304	01	317		0	0	S1	4	0	2RG
A33	870	7311688	6304	01	319		0	0	S1	4	0	2RG
A33	871	7311684	6304	01	311		0	0	S1	4	0	2RH
A33	871	7311689	6304	01	321		0	0	S1	4	0	2RH
A33	871	7311690	6304	01	323		0	0	S1	4	0	2RH
A33	871	7311691	6304	01	325		0	0	S1	4	0	2RH
A33	871	7311692	6304	01	327		0	0	S1	4	0	2RH
A33	880	7474838	2404	15	307		0	0	S1	4	0	2SR
A33	880	7474890	2404	15	324		0	0	S1	4	0	2SR
A33	880	7474893	2404	15	326		0	0	S1	4	0	2SR
A33	885	7567902	2264	01	300		0	0	S1	4	2	2TF
A33	909	7507752	5358	03	332		0	0	S1	4	0	2WK
A33	923	7507876	5359	01	219		0	0	S1	4	0	2YM
A33	925	7507877	5359	01	317		0	0	S1	4	0	2YO
A33	926	7507916	5359	03	333		0	0	S1	4	0	2YP
A33	926	7653082	5359	03	113		0	0	S1	4	0	2YP
A33	938	7602562	2606	15	304		0	0	S1	4	0	3AF
A33	938	7602569	2606	15	305		0	0	S1	4	0	3AF
A33	938	7602573	2606	15	306		0	0	S1	4	0	3AF
A33	938	7602577	2606	15	400		0	0	S1	4	0	3AF
A33	985	7560861	2519	01	211		0	0	S1	4	0	3HQ
A33	986	7636371	2320	01	300		0	0	S1	4	0	3JU
A33	987	7636380	2320	01	304		0	0	S1	4	2	3IK
A33	987	7636388	2320	01	316		0	0	S1	4	2	3IK
A33	994	7641850	2321	01	307		0	0	S1	4	0	3KN
A33	1003	7646420	2421	16	210		0	0	S1	4	0	3MC
A33	1003	7646423	2421	16	211		0	0	S1	4	0	3MC
A33	1003	7646427	2421	16	214		0	0	S1	4	0	3MC

Fig. 27: Example of a KIT table

The table's fields are the following:

- *HNR\_ID*: identification number of the ship;
- *NO\_Kit*: identification number of the kit;
- *ELM\_NO*: identification number of the assembly. An assembly could be identified either by the *ELM\_NO* or by the set of fields *HNR\_ID*, *NOPAN*, *IND*, *AS3*, *AS4*. The number is not given by the user but is deducted from these five fields;
- *HNR\_ID*: identification number of the ship;
- *NOPAN*: identification number of the panel that contains the assembly;
- *IND*: identification number of an upper hierarchical element that contains the assembly;
- *AS3*: identification number of the assembly. However this number is not complete enough to identify the assembly. We need the *ELM\_NO* field;
- *AS4*: assembly built in the workshop can be an element of another assembly. They have the denomination *AS4* and have also a specific identification number;
- *X\_LOCAL* and *Y\_LOCAL*: Position of the assembly in the working area. If no information is given these coordinates will be calculated automatically;
- *SOUR*: define the kind of welding done on the assembly. Depending on this parameter the welding time will be calculated differently. If we get information of the *PHL* this field is not used.

- *GROUPE*: see explanations below;
- *NBR\_RAB*: For the kit number of plates that must be welded in the joining area;
- *KIT\_SCM*: identification number of the kit but in a string format. This column is redundant with the column *NO\_Kit*.

An important notion to highlight is the field *GROUPE*. This field is used to generate the production sequence of kits. This production sequence could be given by the user but can also be found automatically. The main goal of the optimisation of the workshop will be to find the best sequence. However kits have to respect some constraints: pieces that constitute it come from previous workshops and we have to depend on these workshops. In other words the first kit can maybe be built in second, third, or fourth position but probably not in the last one. To take into account this problem we divide the sequence of kits into groups. The order of groups is fixed and cannot be changed. So a group contains several kits and in the group we do not have any constraint on the sequence. The field *GROUPE* indicates the number of the kits groups.

The table is the main entry table of the simulation. This table will be directly imported in the simulation tool. Notice that for the simulation the most important information is assemblies and not kits because the notion of kit can be changed in the simulation – in order to improve the productivity.

### 3.2.3.3 Treatment of STEELM and STEPRT tables

These tables contain all the information about the studied assemblies: list of individual pieces, geometrical dimensions, weight, etc. Globally the first table *STEPRT* contains more geometrical information and the second table *STEELM* important descriptions – identification numbers of IND, AS3, AS4 etc. Nevertheless these tables contain more information that necessary and some files have to be modified – for example extraction of part of the string field.

When these tables are treated we obtain at the end a table: *YL\_KIT\_TABLE*. This table will be imported in eM-Plant and will be the main reference table. It also contains a list of each piece of kits – defined in the initial *KIT* table – and their geometrical characteristics.

By treatment of the tables we mean use of SQL<sup>8</sup> requests and VBA<sup>9</sup> developments. For instance one difficulty is finding out in which container each piece will arrive. The followed methodology is explained in Fig. 28.

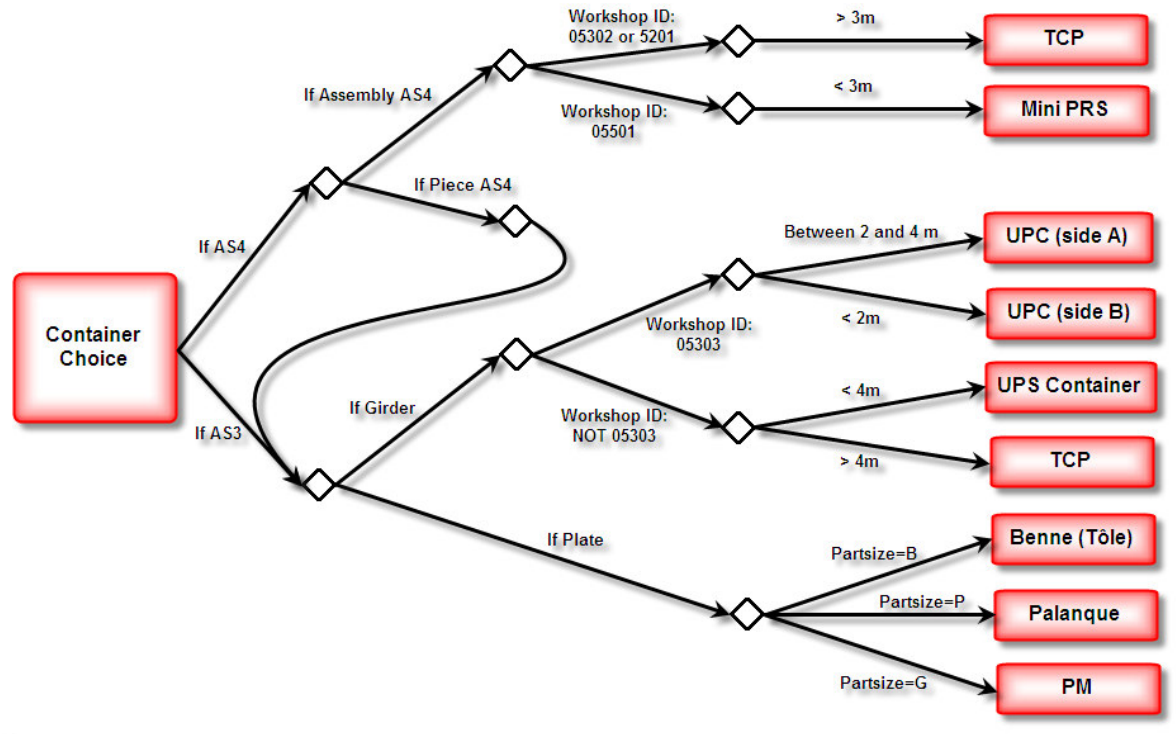


Fig. 28: Scheme of containers determination

Each arrow is an attribute of the piece that we can find in one of the two main tables. With SQL and VBA operations we can thus program this scheme and finally get for each piece a container type.

Specific requests and operations are explained in the next sections.

### 3.2.3.4 The YL\_KIT\_CHILD request

The main request to obtain the YL\_KIT\_TABLE is called YL\_KIT\_CHILD. The request is quite complicated and requires the following tables and requests:

<sup>8</sup> Structured Query Language

<sup>9</sup> Visual Basic for Applications

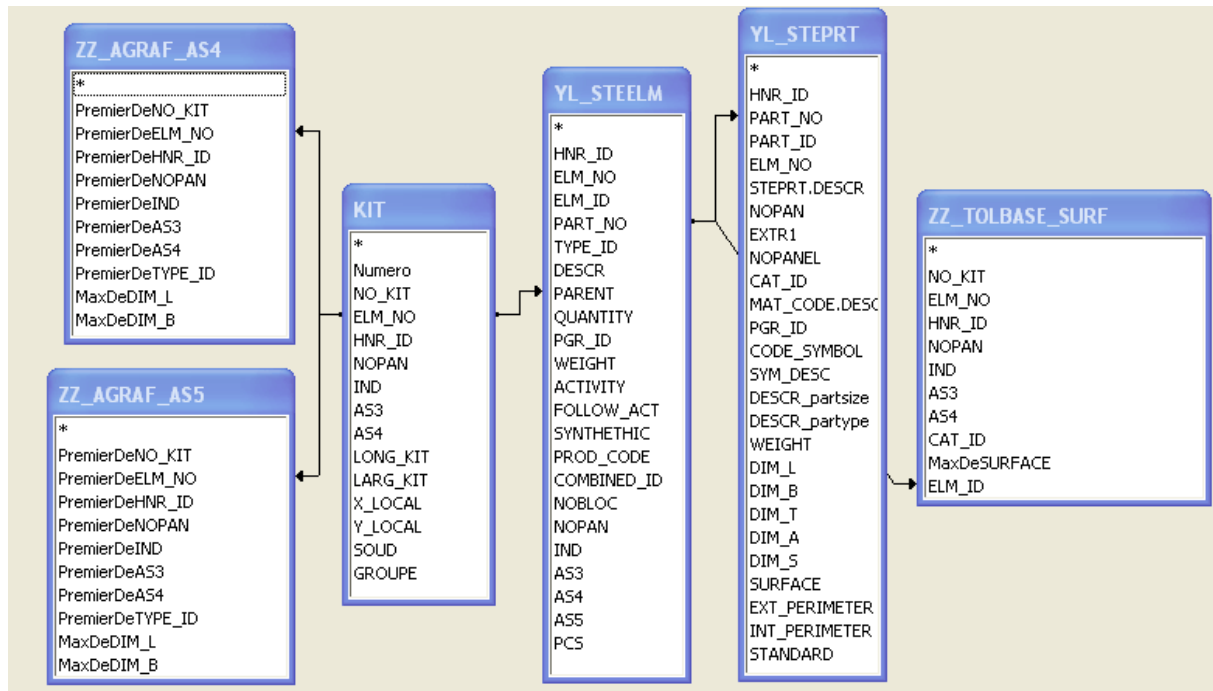


Fig. 29: Scheme of the YL\_KIT\_CHILD request

- Requests : *YL\_STEPRT*, *YL\_STEELM*, *ZZ\_AGRAF\_AS4*, *ZZ\_AGRAF\_AS5*, *ZZ\_TOLBASE\_SURF*
- Table : *KIT*

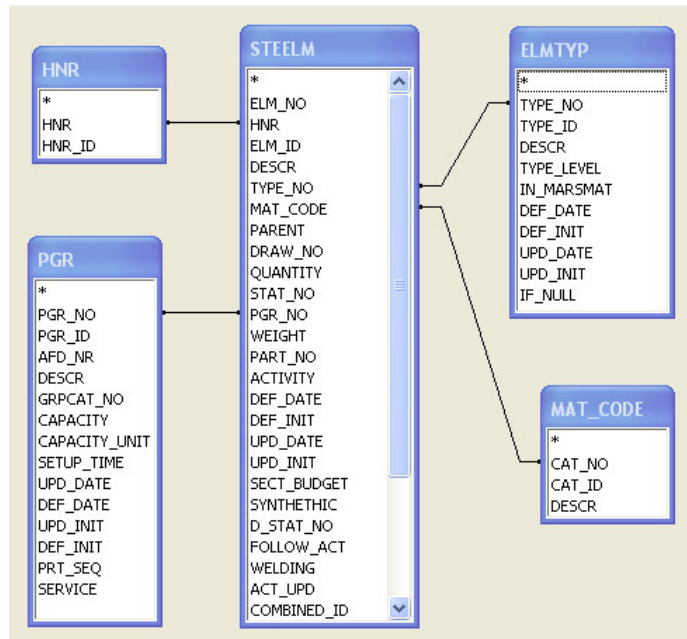
Request *YL\_STEPRT* and *YL\_STEELM* are extractions of respectively *STEPRT* and *STEELM*. Requests *ZZ\_AGRAF\_AS4* and *ZZ\_AGRAF\_AS5* are used to find maximal dimensions of AS4 and AS5. They are also used to find their welding length. The request *ZZ\_TOLBASE\_SURF* is used to find if the piece is a basic plate or not.

Fields of the request are almost identical to those of the *YL\_KIT\_TABLE*.

Two requests are important and should be explained in more detail: *YL\_STEPRT* and *YL\_STEELM*. Here again some fields are calculated with VBA functions but we will not explain it in details.

### 3.2.3.5 The YL\_STEELM request

This is an extraction of the *STEELM* table. Links with others tables are indicated in the following figure:



**Fig. 30: Scheme of the YL\_STEELM request**

This is the first request to execute once we integrate a new STEELM table into the database.

### 3.2.3.6 The YL\_STEPRT request

This is an extraction of the STEPRT table. Links with other tables are indicated in the following figure:

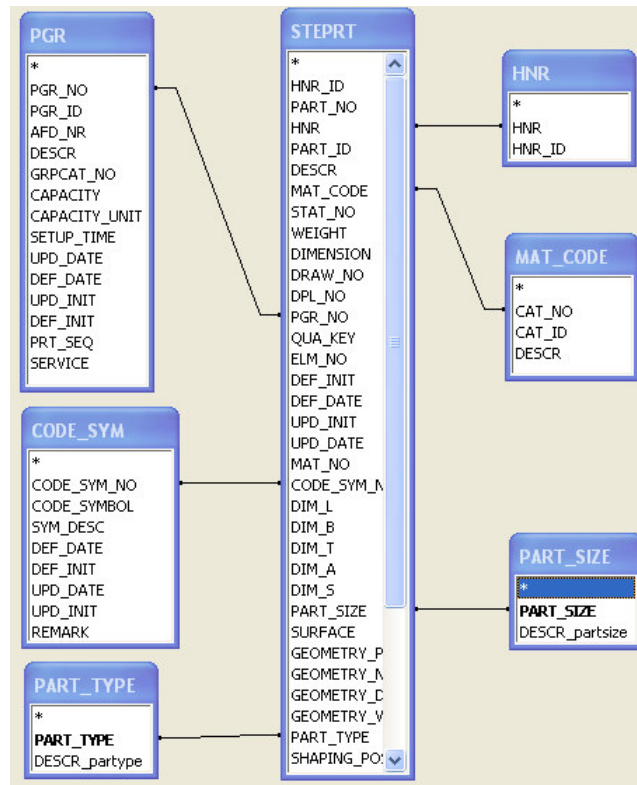


Fig. 31: Scheme of the YL\_STEPRT request

This is the first request to execute once we integrate a new STEPRT table into the database.

### 3.2.3.7 The YL\_KIT\_TIME request

That request is used to obtain the total welding time for each assembly of each kit:

	NO_KIT	ELM_NO	X_LOCAL	Y_LOCAL	SOUDTIME	SOUD
	1	5494555	3	1,5	75952,527473	S1
	1	5494571	10	1,5	27081,318681	S2
	2	5494447	7	2,5	36237,362637	S1
	2	5494482	1,5	1,5	35338,901099	S1
	2	5494498	3	4	30938,901099	S2
	2	5494581	11	1,5	27081,318681	S2
	3	5494524	5	1,5	116169,67033	S2
	4	5494437	11	1,5	36295,384615	S1
	4	5494463	3	1,5	75761,758242	S2
	5	5438282	11	1,5	67959,56044	S2
	5	5439494	5	1,5	43447,912088	S1

Fig. 32: Fields of the YL\_KIT\_TIME

Calculation of the welding time needs a parameter in the table KIT – the SOUD field. to Equation 1 gives an estimation of the welding length for an assembly.. In Table 1 we give the detailed parameters of the formula where:

- DIM\_L is the length of the piece (Girder, Plate or Sub Assembly);
- DIM\_B is the width of the piece;

- The parameter  $k$  represents the proportion of vertical welding compared to the horizontal welding.

Nevertheless this formula is still an approximation.

$$m\acute{e}tr\acute{e}soud\acute{e} = ms = \sum (PS1 + PS2) + \sum (TS1 + TS2) + \sum (AS1 + AS2)$$

**Equation 1: Total Welding length (= "Métré soudé")**

**Table 1: Analytic evaluation of the welding length**

Symbol	Type	Welding	Formula
PS1	Girder	S1	$2DIM\_L + k \cdot 4DIM\_B$
PS2	Girder	S2	$DIM\_L + 4DIM\_B \cdot (1 + k)$
TS1	Plate	S1	$2DIM\_L$
TS2	Plate	S2	$2DIM\_L$
AS1	Sub Ass.	S1	$2Max(DIM\_L)$
AS2	Sub Ass.	S2	$2Max(DIM\_L)$

Evaluation of the welding time must be specified. This time is estimated by different equations explained below. The Arc Time is defined by Equation 1. The robot is not always welding and has to position correctly its arms. We estimate this time – Run Time – by Equation 3. Finally, the total process time – Equation 4 – is still greater due to failure, hitches, etc. Different parameters  $k_2$ ,  $k_3$ ,  $k_4$  and  $k_5$  are defined in Table 2.

$$T_{Soudage} = Arc_{Time} = ms \times \frac{1}{V_s} \times \frac{1}{k_2}$$

**Equation 2: Welding Time (ArcTime)**

$$T_{Portique} = T_{Soudage} \times \frac{1}{k_3}$$

**Equation 3: Run Time**



$$T_{Process} = T_{Portique} \times \frac{1}{(k_4 + k_5)}$$

**Equation 4: Total Process Time****Table 2: Welding coefficient**

	Symbol	Value
Productivity gain due to the 2 arms	k2	1.3
Arc Time / Run Time	k3	0.75
Lack of workforce	k4	0.04
Diverse stops	k5	0.1

The welding speed  $V_s$  is 50 (cm/min). However in practice  $k_4$  and  $k_5$  coefficients raise too much welding time. Finally we will not take into account these factors.

### 3.2.3.8 The YL\_KIT\_TABLE

With the table KIT this table is the most important for the simulation. This is the result table arising from all the treatment. The table contains all information about all pieces that will be modelled in the simulation. The number of lines corresponds thus to the number of pieces that are needed to build kits given in the KIT table. Fields in the table are numerous and not all of them will be discussed but they can be classified in different categories:

- Fields relative to the piece identification. Indications are also given to identify parent assembly;
- Fields relative to the description of the piece. In particular fields that indicate container's type of the piece and equipment required to handle it – with the mechanised gripper, manually, or with a crane bridge, with slings, with magnet, etc. Container field is important because the way the piece enters depends directly from this field;
- Fields relative to the geometry. All dimensions are given, as well as the weight. Dimensions are important to represent exactly the assembly's size;
- Fields relative to operations times. In fact there is only one field: the assessment of the tacking time. This time is simply function of the type of

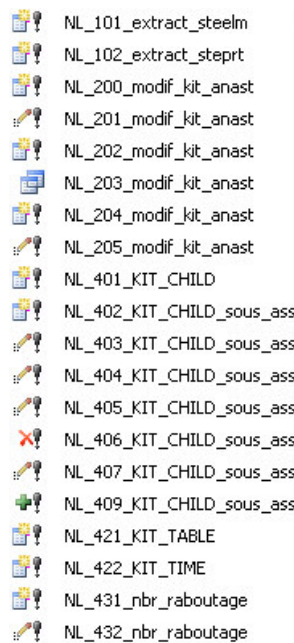
piece, its length and thickness. Of course the times given are only means and the simulation will use a distribution when modelling the tacking operation;

- Fields relative to special operations. For instance a flag indicates if the plate must be joined before to be brought on a working area. Another one indicates if the plate is the basic plate or not.

This table and the KIT table are directly imported from Access into the simulation software. Once the importation is done we have no exchange with the database! This is important because these exchanges slow down strongly the CPU-time of the simulation. If we want to simulate a new list of kits we have to treat the database again.

### 3.2.3.9 Actions to run to execute the treatment

In practice we do not have to run each request one after another when we have new data. In fact, requests to execute are sorted and have special ID numbers to know the execution order – see Fig. 33. These requests are called main requests, described in the previous workshop.



**Fig. 33: Requests to execute for a new treatment**

The execution of these methods creates the table YL\_KIT\_TABLE depending on the KIT table given in entry. In the file Access containing the database we have created two buttons – see Fig. 34.



**Fig. 34: Simple interface to manage the database treatment**

If we want to simulate a new list of kits we just have to import into the Access file the table and to click on the “New KIT” button. If assemblies in the table are new and information about them is not in the table STEELM or in the table STEPRT we first have to import these tables! Indeed we cannot do the treatment if data is not available. Once tables are imported we can click on the “New STEELM or STEPRT” button and thereafter on the “New KIT” button. All required requests are automatically run! If there are incoherencies in the database – lack of information, only one plate to join, etc – detailed messages will be given to the user when he will do the importation of the database in the simulation model.

### 3.2.4 Description of the simulation software

Many simulation software are available on the market. Each of them has its specific options/characteristics and differences. The following main options can be found in most of the simulation package:

- Graphical model construction (icon or drag-and-drop);
- Model building using programming/access to programmed modules;
- Runtime debug;
- Input Distribution Fitting;
- Output Analysis Support;
- Batch run or experimental design;
- Optimization;
- Code reuse (e.g., objects, templates);
- Model Packaging (e.g., can completed model be shared with others who might lack the software to develop their own model);
- Tools to support packaging;

- Mixed Discrete/Continuous Modelling (Levels, Flows, etc.);
- Animation;
- Real-time viewing;
- Export animation (e.g., MPEG version that can run independent of simulation for presentation);
- Compatible animation software;
- 3D Animation;
- Import CAD drawings;

Here is a non exhaustive list of Simulation package available: @RISK, AgenaRisk, Analytica, AnyLogic, Arena, AutoMod, AutoSched AD, Bulesss Simulation Software, Crew Station Design Tool (CSDT), Crystal Ball Professional, CSIM 19, Emergency Department Simulator, Enterprise Dynamics, ExpertFit, ExtendSim, Flexsim, ForeTell-DSS, GoldSim, GPSS/H, Integrated Performance Modelling Environment (IPME), L-SIM, Lean-Modeler, MAST, MedModel Optimization Suite, Micro Saint Sharp, mystrategy, NAG C Library, NeuralTools Plant Simulation (previously named eM-Plant or even SiMPLE++), Portfolio Simulator, Process Simulator, ProcessModel Professional, Project Simulator, ProModel Optimization Suite, Proof Animation, Quantitative Methods Software (QMS), Renque, SAIL, ServiceModel Optimization Suite, ShowFlow 2, Sigma, SimCad Pro, SIMPROCESS, SIMUL8 Professional, SLIM, SLX Stat::Fit, The DecisionTools Suite, Vanguard System, VISIO Simulation Solution, WebGPSS, WITNESS Simulation, XLSim(R), etc. The application fields are different and all the software is simulation oriented but not necessary designed for the specific production simulation.

The chosen software was eM-Plant© from Tecnomatix but the company is now part of Siemens Group and the new name of the software is Plant Simulation©. The tool contains almost all the options cited before. Nevertheless the main reason for the choice is mainly because this tool is used in the Flensburger Shipyard and it assures that the tool is good enough and adequate to be used to model shipbuilding production.

The tool is a completely object oriented software and uses the discrete-event simulation method – see section 2.2.1.4 for further information. A list of predefined objects is part of the simulation package. The main difference between the simulation software and classical object oriented software – like Java for instance – is the way a model is built. In Java we start

directly by typing lines of programming. In eM-Plant, we start by working with icons – that represents completely developed object – and then we can add programming lines to manage specific behaviours. It is much more convenient to first work on the graphical interface than programming lines straightaway. One advantage is that for simple models we do not need a lot of programming and we just have to fill the parameters of objects that are already available in the software library. Of course when the model becomes more complex the user has to modify basic objects by coding new objects or by improving existing objects. The programming language used is specific to the eM-Plant software. To understand the main philosophy of the software, we will describe here the main objects used to develop our model. This description is important to have a good overview of the possibilities and ways of building simulation models!

Objects are grouped in different categories:

- Material Flow;
- Resources;
- Information Flow;
- User Interface;
- MUs<sup>10</sup>;
- Tools

The list described here is not exhaustive.

### 3.2.4.1 Material Flow Objects

These objects are the main ones of the simulation. They allow the user to represent the production flow and all the workshop's manufacturing machines. Produced objects are represented by the MUs objects.

- The *Frame* Object

This is the main object to which we will add all other objects. One object *Frame* can be inserted into another one and that allows the user to easily organize the model into a hierarchy.

---

<sup>10</sup> Movable Units

To model a system we at least need one object *Frame* that will be the root object. Management of the simulation time is done from this root *Frame*.

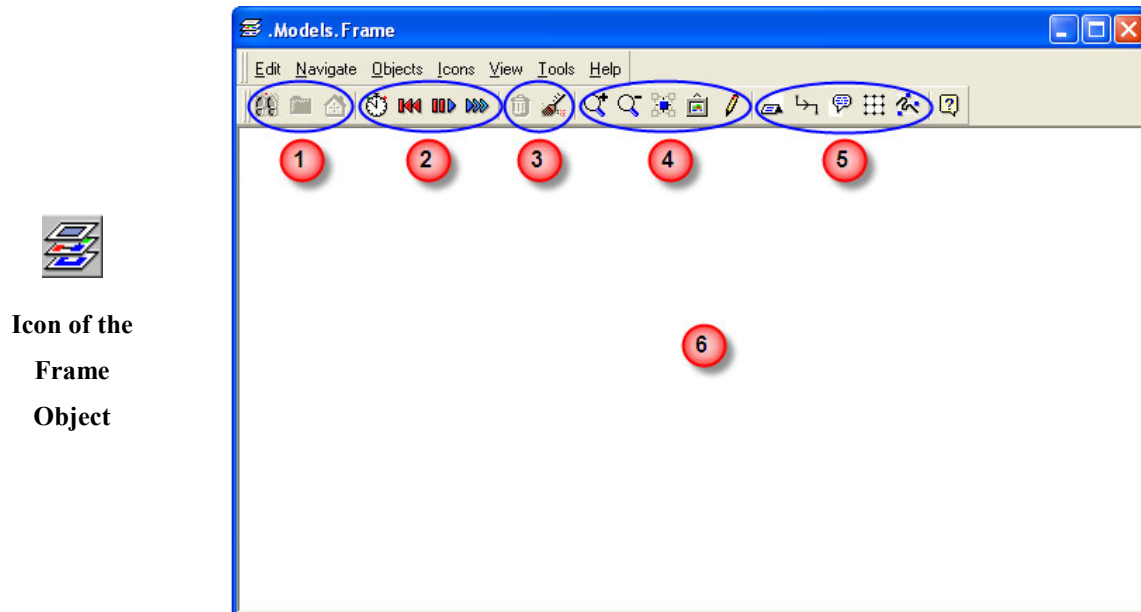


Fig. 35: The Frame Object

- ① : Shortcut to move into the hierarchy of the model;
- ② : Buttons to manage the time of the simulation;
- ③ : Buttons to delete Movable Units;
- ④ : View navigation buttons;
- ⑤ : Buttons to show/hide comments, objects name, information, etc.;
- ⑥ : Main area where we can add objects to model our system.

- The *EventController* Object

This object coordinates and synchronizes different events occurring during the simulation. This object must be situated in the model's root Frame. It allows the user to run the simulation and to stop it, to count the total time of the process. The speed of the simulation can also be controlled. The list of all future events is also indicated in this object which is very important to more easily debug the program. It is also possible to introduce the time when the simulation must stop and when statistics must be recorded. This object can also run initialisation and reset methods to "prepare" the model and all the objects for the simulation.

Syntax of time is <dd>:<hh>:<mm>:<ss.ssss> for days, hours, minutes and seconds (with a precision of 1/10000 seconds) although the syntax can be changed and chosen by the user.



**Icon of the  
EventController  
Object**



**Fig. 36: The EventController Object**

- The *SingleProc* Object

This is the main “machine” object. It can represent any machine but can only treat one piece at the same time. This object can be configured in encoding parameters: process time, set-up time, repair time, frequency of failures, etc. Each of these times can be linked to a probable distribution: normal, uniform, logarithmic, Poisson distribution, Gamma, Beta, etc. Furthermore it is possible to link the machine to a kind of resource. For instance, if a piece must be welded we need to have a free welding man. This situation is the same for the failure to repair, the operation can be done only if the right resource is available. When a resource is not free the piece has to wait on the machine.

The object automatically records different statistics such as the number of pieces in and out, percentage of working time, waiting time, blocking time (if the piece can be moved away because its destination is full), etc. This characteristic is the same for almost all the objects in the library.

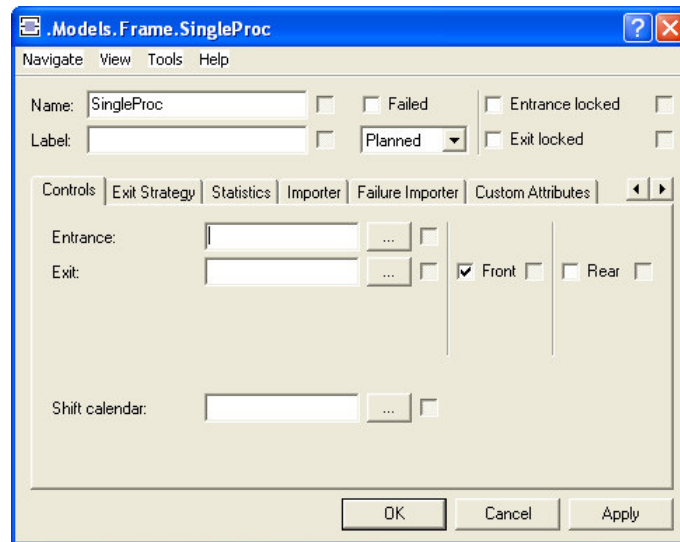
Furthermore, as is the case for each object, the user can add new attributes to the machine. That could be a string variable to indicate the state of the machine or still a characteristic dimension. This property gives a very high flexibility for the modelling.

As is the case for each object that can receive movable units (called MUs) the exit or entrance of one of these pieces can trigger a *Method* (see further) that runs programming

codes to carry out the desired operation – for instance changing the colour of the piece at the end of a painting hall.



**Icon of the  
SingleProc  
Object**



**Fig. 37: The SingleProc Object**

Different exit strategies are available. If the object is linked to several other objects, the MUs can choose the successor depending on the chosen parameters: cyclic, random, least recent demand, most recent demand, max content, min content, etc. By default the choice is huge and the user can also define his own rules thanks to an exit method.

Other parameters are not described here.

In the same categories of object, we have also:

- The *ParallelProc* Object: The *ParallelProc* has several stations for processing moving units (MUs) in parallel at the same time. The built-in properties of the *ParallelProc* are the same as those of a *SingleProc* with several processing stations. When you do not enter a special entrance control, the *ParallelProc* places an incoming MU onto a random station. Set-up time always applies when an MU has a different name than the MU it processed before, i.e. its predecessor.
- The *Assembly* Object: The *Assembly* station adds mounting parts to a main part, for example doors and fenders to a car body. It moves the mounting parts either to the main MU or it deletes them. When the assembly process requires services, we can assign the order in which the *Assembly* station requests mounting parts and services.
- The *DismantleStation* Object: The *DismantleStation* removes mounting parts from the main MU or it creates new MUs.

The last object will not be used for the modelling of the PrePreFabrication workshop.

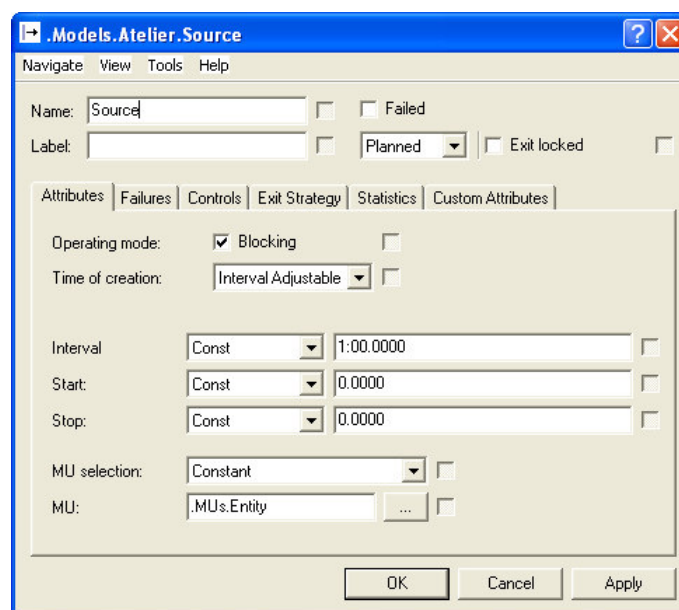


- The *Source* Object

The Source produces MUs in a single station. It has a capacity of one and no processing time. It produces different types of MUs one after the other or in a mixed sequence. We can set a procedure to determine the generation times as well as a procedure to determine the types of MUs to be produced. As an active material flow object, the Source attempts to move the MUs it produced to objects it is connected to. We also define how the Source is to proceed when it cannot move an MU to its succeeding object. We have to use the Source to create parts and work pieces that move through our installation to be processed by the different stations.



**Icon of the  
Source  
Object**



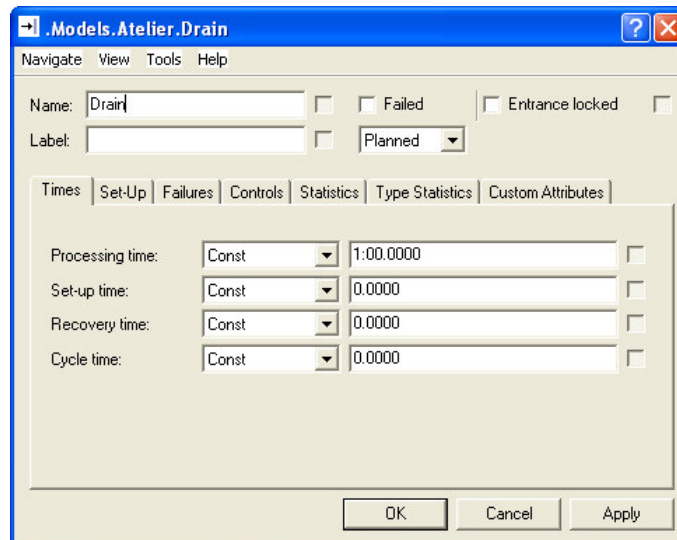
**Fig. 38: The Source Object**

- The *Drain* Object

To remove the parts and work pieces from the simulation models, after they have been processed, for example to model the shipping department, we use the object Drain.

The *Drain* has a single processing station. It removes the MU from the installation after setting up for it and after processing it. The built-in properties of the *Drain* are the same as those of the SingleProc. The *Drain* destroys the MU after it processed it, instead of moving it on to a succeeding object in the flow of materials and it collects statistics about the MU.

→ |  
**Icon of the  
 Drain  
 Object**

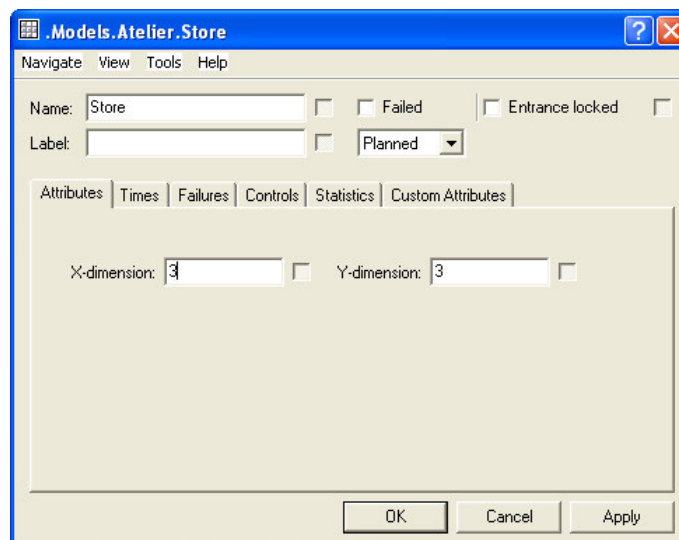


**Fig. 39: The Drain Object**

- The *Store* Object

The *Store* stores any number of MUs we define. They remain in the *Store* until we remove them, for example by using a *Method*. The *Store* receives MUs as long as storage locations are available. The MU triggers a sensor when it enters the *Store*. The sensor then calls an Entrance control, i.e. a *Method* object that determines the storage location where the *Store* places the MU. The entrance control can update the inventory list or execute any other action we define. If we do not define an Entrance control, the *Store* places the MU onto the first unoccupied location in the net of coordinates.

  
**Icon of the  
 Store  
 Object**



**Fig. 40: The Store Object**

- The *PlaceBuffer* Object

The *PlaceBuffer* has a number of stations, arranged in a row, one behind the other. The MUs it processes have to advance from station to station and can only leave the *PlaceBuffer* after they passed the last station. This way, we can call each and every station individually.

An MU can move directly to an idle station in the *PlaceBuffer*. When we create an MU on an idle station, it may pass on to the next station or the next object.

In case the first MU cannot leave the *PlaceBuffer*, you can decide what happens next:

- To make the MUs accumulate on the *PlaceBuffer*, select the check box Accumulating. This allows the MUs to move front to end to each other, when the exit of the *PlaceBuffer* is blocked.
- To make the MUs retain their distance to each other, i.e. make all succeeding MUs stop moving when the preceding MU cannot exit, clear the check box to deactivate Accumulating.

We can only define the processing time for the *PlaceBuffer* as whole, not for each individual station. It divides the processing time equally among its stations, i.e. when we set a processing time of 60 seconds for 3 stations each station processes the MU for 20 seconds.

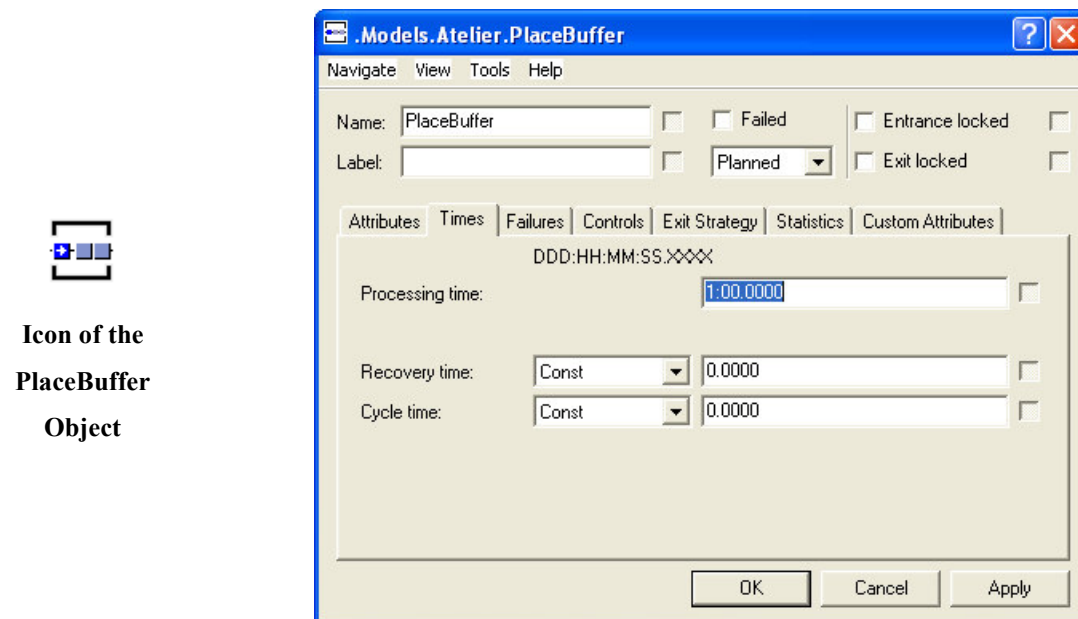


Fig. 41: The PlaceBuffer Object

- The *Connector* Object

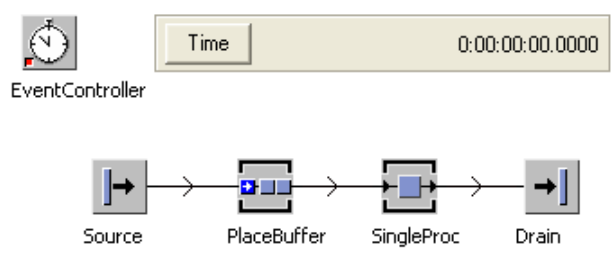
The *Connector* establishes material flow connections between two objects in the same *Frame* or connects an object with an exit or entrance – modelled with the Interface – of a

*Frame*. The *Connector* shows the direction of the connection with an arrow head in the middle of the connecting line.



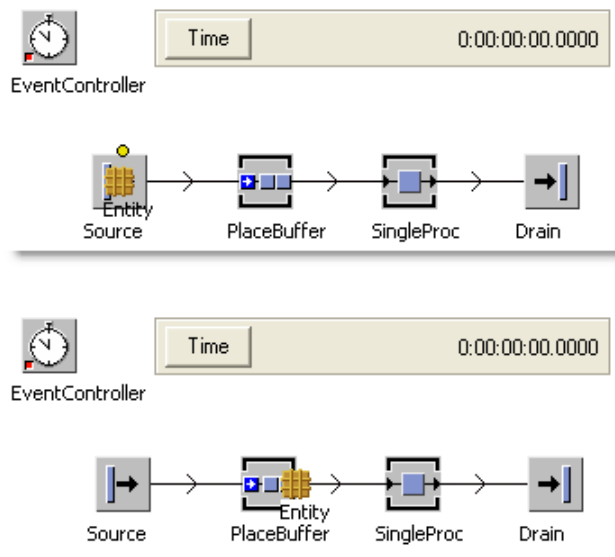
**Fig. 42: Icon of the Connector Object**

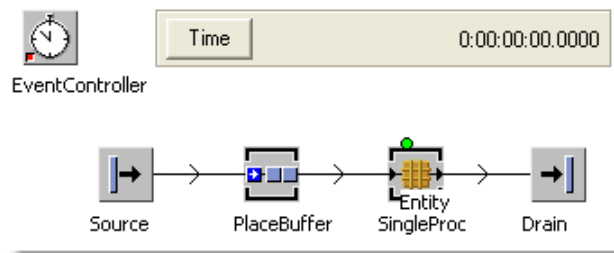
On Fig. 43 we can see a very simple model: a *Source* connected to *PlaceBuffer*, itself connected to a *SingleProc*, itself connected to a drain. The Source takes one minute to create a new MU, the PlaceBuffer has no process time. The SingleProc operates for two minutes and the drain takes one minute to evacuate the MU.



**Fig. 43: Example of a simple system modelled**

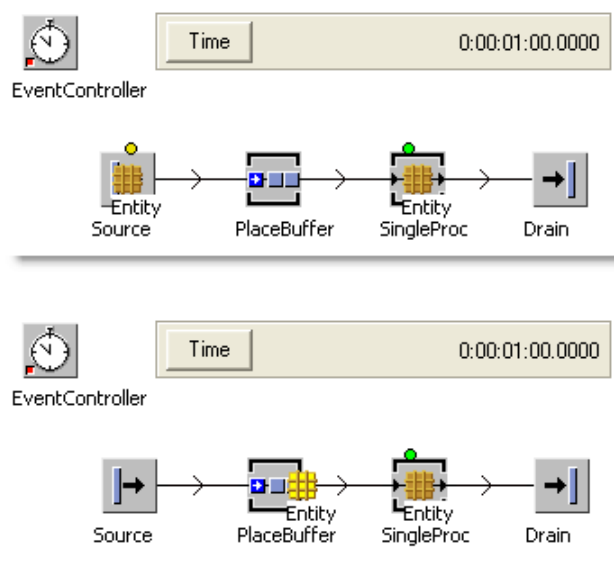
As a first step an entity MU is created onto the *Source* object. Then the MU is directly sent onto the PlaceBuffer. It remains there for zero seconds because we have defined no process time for this object. The MU will then go onto the *SingleProc* object.





**Fig. 44: First step of the process**

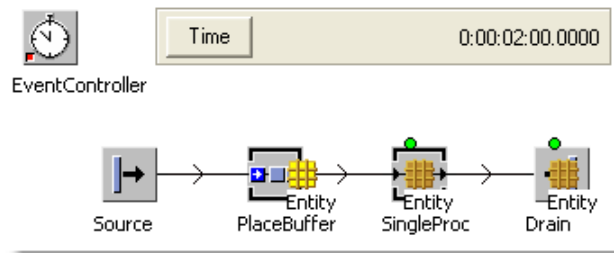
For the second step, after one minute the *Source* object creates a new entity MU. Once again it is sent on the *Place Buffer*. Because the *SingleProc* can treat only one MU at the same time the second entity MU must be blocked on the *PlaceBuffer*.



**Fig. 45: Second step of the process – creation of a new MU**

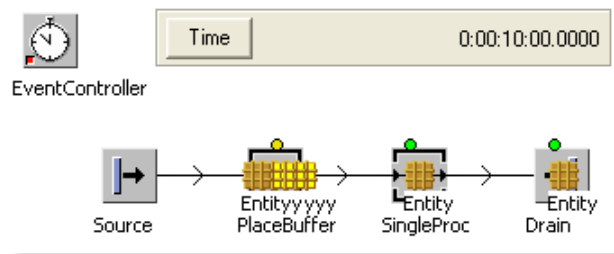
Note that when the entity is blocked – waiting for a free place in its successor object – the icon of the object changes to yellow. In fact for each MU we have a different icon for each state: operational or waiting. We can change these icons into what we want. Furthermore we can also create our own state. For instance, we can define a state “Painted” and if the MU represents a car we change the colour of the car in selecting the right state – at the exit of a painting workshop for example. We can define an unlimited number of icons by MU.

After two minutes the first MU on the *SingleProc* has been processed and can be moved away onto the *Drain* object. The second MU can thus enter on the *SingleProc* and a new MU is created by the *Source* and brought onto the *PlaceBuffer* – see Fig. 46.



**Fig. 46: Third step of the process**

After three minutes the first entity is deleted by the drain and thus disappears from the frame. Other entities make a step forward and a fourth MU is created. We can obviously guess that with such a system the number of entities on the *PlaceBuffer* will continuously increase until we reach the maximum number of MUs tolerated. For instance after several minutes we get the situation of Fig. 47 with five entities on the *PlaceBuffer*.



**Fig. 47: State of the system after ten minutes**

- The *Track* object

The *Track* represents part of a transport line, with or without automatic routing. The *Transporter* – see MUs section – is the only moving material flow object that can use the *Track* in a meaningful way. We might, for example, utilize both to model an AGV (automatically guided vehicle) system.

The distance the *Transporter* has to travel on the *Track*, defined by the *Transporter's* Length, and its speed, defined by the Final speed, determine the time the *Transporter* remains on the *Track*. As opposed to the other material flow objects, eM-Plant uses the actual Length you enter during your simulation run. One *Transporter* may not pass another one by moving in front of it. *Transporters* thus retain their order of moving onto and exiting of the *Track*.

When several *Transporters* travel along the *Track* at different speeds, and a faster one collides with a slower one, eM-Plant activates the Collision control of the faster *Transporter* and automatically reduces its speed to the speed of the slower *Transporter*.

The maximum capacity of the *Track* is defined by its length and the lengths of the individual *Transporters* moving on it, i.e. a *Track* that is three yards long accepts three

*Transporters* of one yard each at the most. We can use the attribute *Capacity* to further restrict the number of *Transporters* located on the *Track*.

We can insert the *Track* into a *Frame*:

- As a curve, which is the default.

By inserting any sequence of curved segments and straight segments, we can realistically model curved tracks on which the *Transporters* move.



**Fig. 48: Path Curved created by tracks objects**

- With its icon.

Then it looks like this:



### 3.2.4.2 Resources Objects

In order to use a “machine”, that machine should generally be managed by workers or resources. All objects of the type *Resources* contribute to the management of these workers.

Each resource has some abilities – for example a welder has the ability to weld. These abilities are generally called “services”. One resource can have several services – a welder can do the welding and the tacking. Almost every object might need resources. In fact, for these objects we have to define the required services and their quantity. There is even the possibility to define different alternatives: to do job number 1, we need two workers and one forklift *or* we need one worker and one crane bridge. There are in the simulation package some predefined objects that automatically manage requests/offers to provide resources to an object. To summarize, the machine has a set of attributes (an *Importer*) that requests services to another object called *Exporter* that contains different resources (different services). These objects are fictively linked by an object called *Broker* that manages the exchange of resources. Of course an *Exporter* can be linked to several *Importers*.

In eM-Plant workers are specific resources with their own object. They also have the possibility to move along the path. They also have a specific *Exporter* named the *WorkerPool*.

- The *Broker* Object

The *Broker* works in cooperation with the *Exporter* and the *Importers* of the *SingleProc*, the *ParallelProc*, the *Assembly* station, and the *DismantleStation*.

The *Broker* is the go-between for offered services and required services. Each *Broker* can manage several *Exporters*, that tender services, and it may receive requests from several *Importers*, that require services. A request consists of a list of services and the amount of services necessary.

Once a *Broker* receives a request from an *Importer*, it immediately attempts to fulfil it using the *Exporters* it manages. If the *Broker* does not succeed in doing this, it may also pass the request on to other *Brokers* it is connected to with a *Connector*. The direction of the *Connector* determines the direction in which eM-Plant passes the request on. If the *Exporters* of different *Brokers* are able to satisfy the request, those *Exporters* are assigned. If the request cannot be satisfied immediately, the *Broker* that received it first will save it.

The *Broker* then attempts to provide the service at a later point in time. To do this, the *Broker* goes from requested service to requested service and attempts to find an *Exporter* for each one. If an *Exporter* offers the service, it is going to reserve the maximum possible or the necessary number. Then the *Exporter* cannot offer this reserved amount of services for other services.

- The *Exporter* Object

The *Exporter* exports services. It works in cooperation with the *Broker* and the *Importers* of the *SingleProc*, the *ParallelProc*, the *Assembly* station and the *DismantleStation*. The *Exporter* offers services and provides them for *Importers*.

A single *Broker* manages the *Exporter* and assigns it to an *Importer*. After the *Exporter* has finished providing its service, it registers as being available with its *Broker*, which will assign it to other *Importers* as soon as its services are required.

- The *WorkerPool* Object

The *Worker-WorkerPool-Workplace-FootPath*-concept refines the *Broker-Importer-Exporter*-concept.



You can assign a *Workplace* to the material flow objects that support an *Importer*, i.e. to the *SingleProc*, the *ParallelProc*, the *Assembly station* and the *DismantleStation*.

When you do not use a *FootPath*, eM-Plant “beams” the *Worker* to the *Workplace*. Here the *Worker* functions as an *Exporter* with a capacity of 1.

The *Workers* are created in the *WorkerPool* and they stay there when they do not work and are waiting for an order. You can define teams of workers in the worker pool and assign them different skills, efficiencies, walking speeds, etc. in a worker Creation Table.

The *Broker* mediates the *Worker* to the individual work stations.

- The *WorkPlace* Object

The *Workplace* is the actual place at the station, where the worker performs his job. When we insert a *Workplace*, and when the *Worker*, who is supposed to work there, cannot reach it via a *FootPath*, eM-Plant “beams” the *Worker* to the *Workplace* and makes him active there.

When we do not insert a *Workplace*, when no *Workplace* is present that supports this service, or when all *Workplaces* that support this service are occupied, eM-Plant “beams” the *Worker* to the work station itself, where he will perform his task.

- The *Worker* Object

The *Workers* are created in the *WorkerPool* and they stay there when they do not work and are waiting for an order. Here the *Worker* functions as an *Exporter* with a capacity of 1.

- The *FootPath* Object

When we do not use a *FootPath*, eM-Plant “beams” the *Worker* to the *Workplace*. When we want to model and simulate distances between the *WorkerPool* and work stations or between work stations, employ the *FootPath*. eM-Plant activates the *Workers* on the *FootPath*, when they move to the stations or in between the stations.

We can link several *FootPaths* with *Connectors* to create a network of *FootPaths*. The *Worker* moves on this network from his *WorkerPool* to the *WorkPlace*. Provided the *WorkerPool* and the *WorkPlace* are connected to the same network of *FootPaths*, the *Worker* moves from the *WorkerPool* to the *WorkPlace* and back using the shortest possible route.

A network of *FootPaths* consists of all *FootPaths* that are joined with *Connectors*, as well as the *WorkerPools* and the *WorkPlaces* that are joined with one of these *FootPaths* with a *Connector*.

The time it takes the *Worker* to move from his *WorkerPool* to the *WorkPlace* depends on his *Speed* and the combined *Lengths* of the *FootPaths* he covers to reach his destination. eM-Plant recognizes on which *FootPaths* the *Worker* moves, when he exits a *FootPath* at another end than the one at which he entered it. When the *Worker* does not move to his destination on a *FootPath* or when he is beamed to his destination, he does not consume any time at all.

Although *Workers*, *Workplaces* and *WorkerPool* will be much used in the simulation of the PrePreFabrication workshop, we will not use *Footpaths* in our model. We have to use these objects only if we want a very detailed simulation model but the times needed for walking are really negligible compared to production operation times. Consequently this is really not adapted to model displacements of workers.

### 3.2.4.3 Information Flow Objects

Information Flow Objects are quite diversified. They are *Tables*, objects allowing to get graphics, interface objects to facilitate exchange with files (txt, XML, OleBD, etc), global variables or furthermore shift calendars. In this category we also have a fundamental object: the *Method Object* used to code anything to improve existing objects or to create new ones.

- The *TableFile* Object

The *TableFile* is a list with several columns. You can access the individual cells by employing their index, i.e. by their position designated by the number of the row and the number of the column. It is used to store data, or to save different statistics or information during a simulation run.

- The *QueueFile* Object

The *QueueFile* is a list with one column that eM-Plant accesses using the FIFO method (First In First Out). eM-Plant saves entries you add in the order you insert them in and removes the element waiting in the queue the longest first. Notice that there is also the *StackFile* Object using the LIFO method (Last In First Out) and the *CardFile* Object (The *CardFile* is a list with one column providing random access to the contents of the individual cells using their position, i.e. row number). Even if these objects are really practical to use,

they will generally be avoided in the simulation because CPU-time to access these objects is greater than for the *TableFile*. If we need to use FIFO or LIFO behaviour we will instead use the *TableFile*.

- The *Method* Object

We use the *Method* to program controls that other objects start and execute during the simulation run. This is one of the most important objects of the simulation because thanks to this object it is possible to represent any behaviour.

Although every object described earlier can be put in the model manually (for instance with drag-and-drop) and parameters filled in by the user, everything can be done automatically with these objects *Method*. For instance attributes of any objects can be changed with a *Method*. New attributes can even be created.

In a model global variables can be created with the *Variable* object – see next section. Methods can create these variables – as any other objects – and change their value. But furthermore it is possible to define local variables in the method. The variable is created when we access the method and is deleted at the end of the execution.

The structure of a method is divided into several parts. If a certain part is not needed, you can delete or omit it.

Each method is constructed like this:

```
[arguments]
[return value data type]
is
[local variables]
do
    [program code]
end;
```

The program code is written with a specific programming language: *SimTalk*. During a simulation run the built-in Interpreter executes the *SimTalk* source code. We can check the state of a simulation run at any time.

Here is a simple example of function – to compute the factorial number of a given number:

```
(n : integer) : integer -- computes the factorial of n
is
do
    if n <= 0 then
        return 1;
    end;
    return n * self.execute(n-1);
```

```
end;
```

The function returns the number given after the keyword *Return*. In that case this is the entry number multiplied by the command:

```
self.execute(n-1)
```

This command executes the current method with n-1 as entry argument. When n is less or equal to one the method returns 1 and does not execute again the self command.

Templates can also be created, saved and re-used by the user. Notice also that if we give the name *Reset* or *Init* to a method its icon changes automatically. The *Init* method is automatically run at the start of the simulation – or if we click on the Init button of the *EventController*. The method *Reset* is run when we want to run another simulation – or if we click on the *Reset* button of the *EventController*. These methods allow the user to initiate different variables and parameters.

- The *Variable* Object

The *Variable* is a global variable that other objects and methods can access during a simulation run. We might, for example, use it to store data over an extended period of time during the simulation run, to count values up and down, to assign values, etc.

#### 3.2.4.4 User Interface

In this category we will use only 2 objects: the *Chart* Object and the *Dialog* Object.

- The *Chart* Object

The *Chart* graphically displays data sets that eM-Plant recorded during a simulation run. We can select to show:

- Data from a table, in which we saved the simulation results, for example.
- Data from input channels we define, that dynamically record the values of attributes or of objects that we are interested in.

- The *Dialog* Object

We can create a dialog window similar to the built-in dialog windows eM-Plant provides with the object *Dialog* to:

- Provide a simple user interface for complex simulation models other users work with.

- Prevent another user from manipulating a *Frame*. The method will not be explained in detail here.

The first point is generally the reason the *Dialog* object is used. In our model these objects will be used to create a user-friendly interface.

#### 3.2.4.5 MUs

The Movable Units (MUs) are units that can move in the simulation process. They are classified into three categories: *Entities*, *Containers* and *Transporters*.

- The *Entity* Object

The *Entity* is a moving object without a propulsion system of its own. It does not have loading capabilities. It symbolizes goods being produced or transported.

- The *Container* Object

The *Container* is a moving object without a propulsion system of its own. It has loading capabilities to hold MUs. We can define its size and thus its loading capacity by specifying the dimension on the x and the y axes. The *Container* is used for modelling items that transport entities, such as palettes, etc.

- The *Transporter* Object

The *Transporter* is a moving object with a propulsion system of its own. We can define its loading capacity. The *Transporter* can hold MUs and move them about freely. It represents forklifts, AGVs, etc. The main difference with *Container* and *Entity* objects is that it can move on *Tracks* object.

#### 3.2.4.6 Tools

This category contains more complex objects developed to manage specific needs. We will describe only one: the *ExperimentManager* Object.

It is used to run several successive experiments in modifying some input parameters in order to observe their influence on the results. These results can be seen in the same way as statistical results – with mean, variance and so on. Graphical results are also available.

Before and after each simulation it is possible to run specific *Init* and *Reset* methods to modify values.

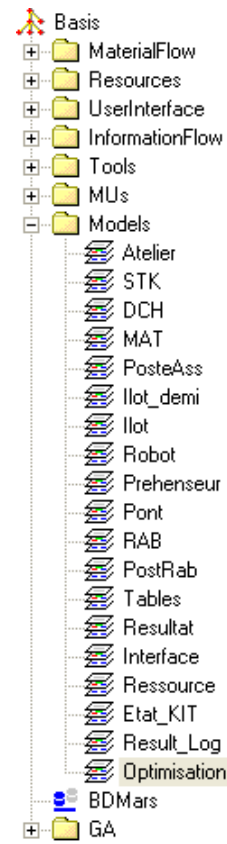
At the end of the whole simulation this object can create automatically an HTML report with all the desired results.

## 3.2.5 Development of the PrePreFabrication Model

### 3.2.5.1 Introduction

The main characteristic of the workshop is its symmetry; the left side (named “A” side) is identical to the right side (named “B” side). The only difference is the list of kits to build, assemblies of the A side belonging to different sub-panels than the B side. In addition to this “vertical” symmetry the workshop almost has a “horizontal” symmetry: the two upper cells are different than the two lower cells only in terms of their width. In fact there are eight half-cells that have to do the exact same work. Consequently it is only necessary to create one object (one object “Frame”) simulating the behaviour of one half-cell and to “copy” it eight times to model the entire workshop. Others objects/frames are still necessary and will be described in the following sections.

All the Frames of the workshop can be seen in the following structure:



The total number of parent Frames is 19. They will be detailed in this section.

**Fig. 49: Structure of the Frames of the PrePreFabrication**

The main frame (root) is the frame *Atelier*. All others frames are inserted (directly or indirectly) into this root frame. From an object-oriented point of view this implies that the frame *Atelier* is the only Parent frame. All others frames are children because they are derived from the root frame.

The hierarchy of the modelled workshop can be seen in Fig. 50.

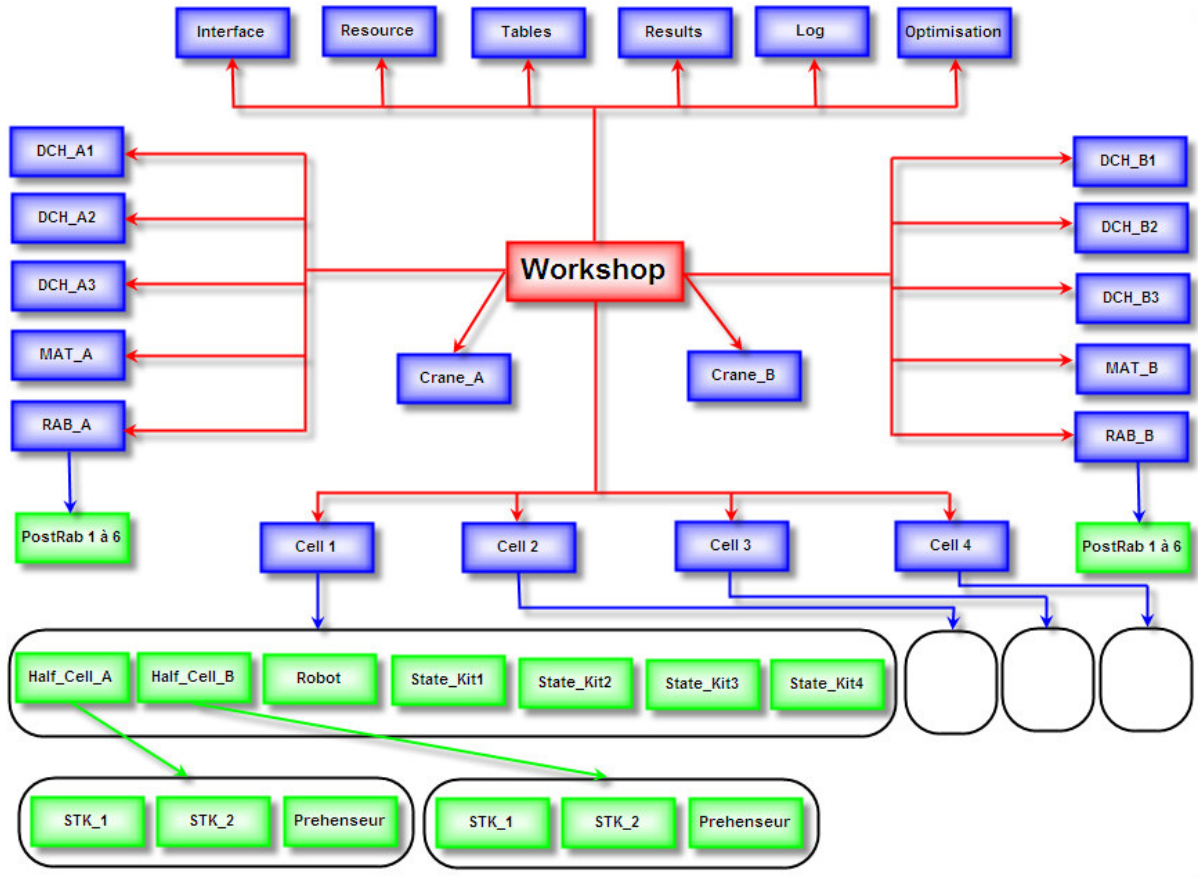


Fig. 50: Structure of frames

Here is a brief description of each of the workshop's frames:

- *DCH*: manages the *PM* containers (named *PM\_Tole* in the simulation). We have three pairs of *PM* for each side. The first two are used to store finished assemblies and the third one for the input of plates.
- *MAT*: this area is the entry area of *TCP* containers that contain *PRS*, long *UPS* and *AS4*. Generally we also use this place to store plates if the joining area is full, or to store finished assemblies if the *PMs* are not available.
- *RAB*: this is the joining area. Plates generally come from *PM* to be welded here. Some smaller pieces (such as *Palanque*) can directly come from a fork lift.

- *PostRab*: this is the area for modelling the joining operation of a plate. By default the total number of such an area is 6 but it can be changed.
- *Crane*: this is the object crane bridge. This object is parameterised and could be used for other workshops with few changes.
- *Cell*: this object represents one cell of the workshop. The total number is 4.
- *Half-Cell*: the half of a cell is modelled by this object. From this object are derived three other frames: two frames *State-Kit* – see below – and one Frame *Prehenseur* (mechanized gripper).
- *State-Kit*: This Frame contains all the information required about the kit that is currently being produced in the workshop. Number of stiffeners by plate, number of stiffeners already brought, number of stiffeners already tacked and so on. We have one frame like this for each real working area, in other words four frames by cell – two by half-cell.
- *Robot*: This frame manages the displacement and the welding of the automated welding robot – one by cell.
- *Prehenseur*: This frame manages the displacement of the mechanized gripper. Tacking operation is managed by the frame *PostAss*.
- *STK*: This frame manages the creation of small containers and individual pieces to be welded onto the main plate.
- *Interface*: Create dialog boxes to facilitate the use of the software by a non eM-Plant user. All the main parameters can be changed easily.
- *Resource*: Workers and their availability are managed in this frame.
- *Tables*: All the information of the database is imported into this frame. It contains many *TableFile* objects that are used or filled by the simulation. The sequence of kits manufacturing is also stored here.
- *Results*: Different results and statistics are filled by the simulation into this frame. This frame is mainly used when we do a parametric – multi-run – study.
- *Log*: Each event in the simulation – creation of a piece, displacement of a plate, etc. – can be stored in log files. This frame lists all these logs.



- *Optimisation*: This frame is used to automatically create objects needed for the optimisation with the genetic algorithm – such as chromosomes, parameters, etc.

Notice that there is also another frame used in the simulation: the frame “*PosteAss*”. This is the frame used to simulate tacking and finishing (by workers or with the mechanized gripper) for each assembly. This frame is inserted (more precisely created) in the root frame when we assign a new assembly onto an area.

### 3.2.5.2 The “Atelier” Frame

To model any system it is inevitable to have a root frame that contains at least the *EventController* object to manage the time of the simulation. In our simulation model, the root frame is the “Atelier” Frame – “Workshop” Frame.

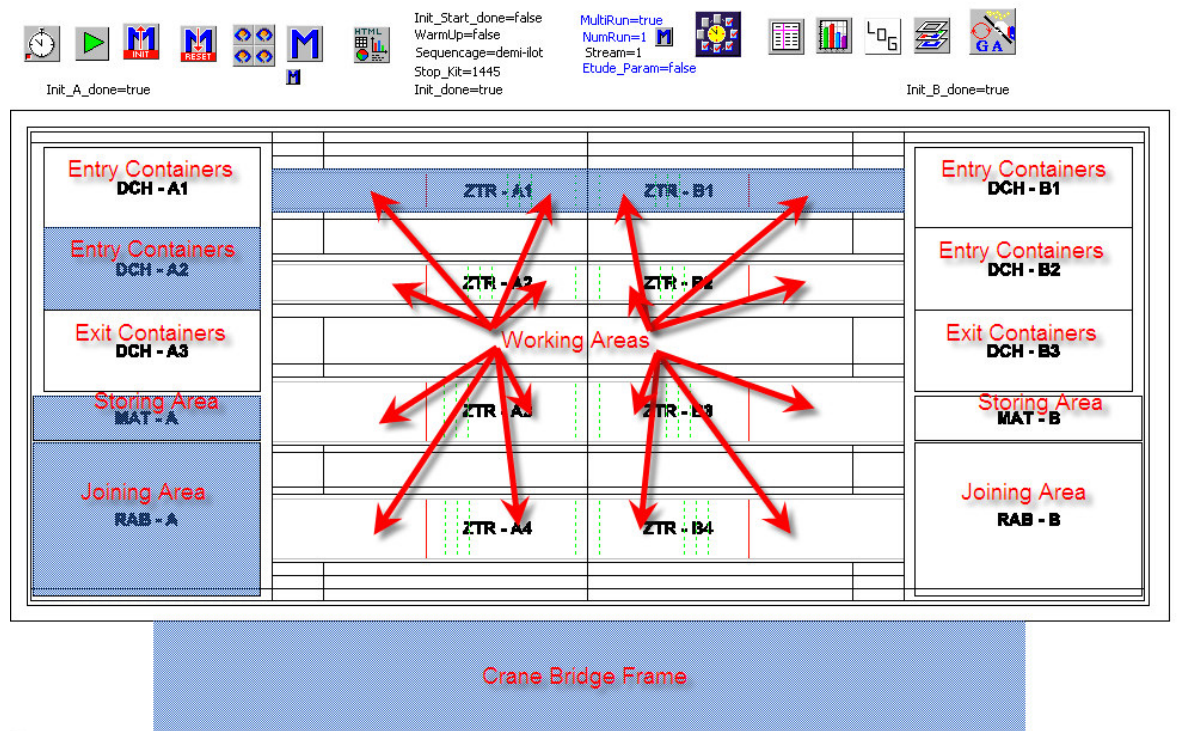


Fig. 51: The “Atelier” Frame

The workshop has five different areas: entry and exit areas of PM (DCH area), area reserved for pieces coming from *TCP* and also dedicated to store temporarily piled up plates coming from the *PM* (*MAT* area), joining areas (*RAB* area). Each of these areas is modelled by a specific Frame. We also have a frame *Demi-Ilot* containing one *Prehenseur* frame and two *STK* frames to model the smallest containers. Two *Demi-ilot* frames are grouped together to constitute the *Ilot* frame. Finally notice the *Pont* frame to model the behaviour of the crane bridge.

The Frame “Atelier” contains six frames DCH, four frames “Ilot”, two frames “MAT”, two frames RAB and two frames “Pont”. Several objects of this frame are selected on Fig. 51. If not selected, these frames are not visible. The root frame also contains others frames that have the size of an icon. By double clicking on any frame, we open the corresponding frame and have access to all the information. Remembering that these frames are children frames, to access the parent frame we have to open it directly from the main structure of the project – see Fig. 50. Modifying one element in a parent frame modifies this element in each of its children! Modifying directly a child does not have any effect on other children! This characteristic is available for each object. By convention we avoid modifying children objects and we work only on the parents. Indeed if we modify children, the program becomes too hard to be maintained: it is not convenient if some children have been modified, to remember which one is similar to its parent and which one is different. If there is a “bug” in an object, by modifying the parent object we have corrected the entire program. If we have the wish to break the inheritance between children and parent we have to correct the bug in all the children.

The total number of frames in the model – included directly in the root frame or into a child – is thus 87! We see clearly that with such a high number of frames we must be careful when doing a modification. These modifications must be done in all cases in the parent frame and never in one of these 87 frames – except the root frame *Atelier*. For instance if we find a bug in the frame *Prehenseur* we have to do the modifications in the parent frame or in each child i.e. in eight frames! The second option could lead easily to numerous programming errors. In fact we can have much more than 87 frames, as we have in the model temporary frames – one by assembly.

These frames can contain other frames. This happens for example for the half cell that still contains other frames. In our case, we can thus from the root frame “Atelier”, open the “Ilot1” (first cell) then open the “Demi\_IlotA” (left half-cell) then open the “STK\_1” (area where smallest containers are).

The selected frames (even if they are not selected) do not have the original icon – see Fig. 52 – but a transparent icon with dimensions corresponding to reality – multiplied by a factor scale.

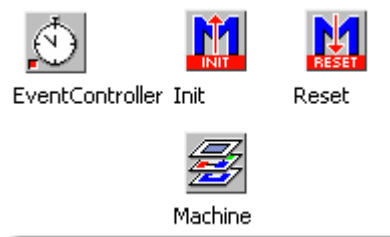


**Fig. 52: Original icon of a frame**

To facilitate the access to all these frames, the two frames “Pont” were put and stacked up just under the workshop (shaded area in blue under the workshop on Fig. 51). If we put it directly on the workshop it will not be possible to have access to other frames (cells, etc) just by clicking on it. Notice that this frame also contains a frame “Tables” (with all the main tables of the simulation), a frame “Resultats” (with results of multi-simulation, for instance if a parametric study is done), a frame “Result\_log (that saves all the important events during a simulation) and a frame “Optimisation” (creates chromosomes for the genetic algorithm). By clicking on these objects we open the corresponding frame.

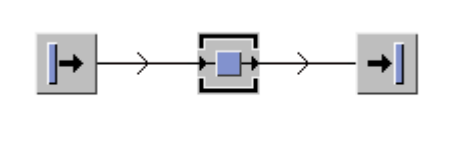
On each icon it is possible to create animation points (or lines) that are linked to objects contained in the frame. The following example clearly shows the functioning.

This simple model is constituted of a main frame Workshop and a child frame derived from the first one:



**Contents of the Workshop Frame**

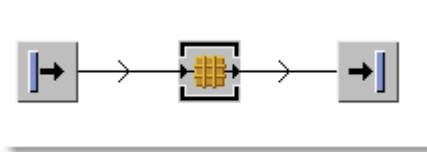
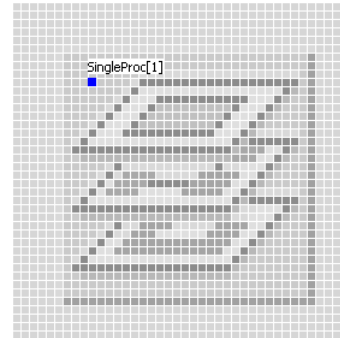
At the initial state, both frames give the state shown on the right.



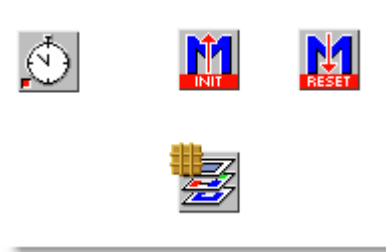
**Contents of the Machine Frame**

We will add onto the frame icon Machine an animation point and link this point to the SingleProc object:

In blue we have created an animation point. Each time that a MU will be found on a *SingleProc* of the *Machine* frame, the MU will be visible on that point in the main frame *Machine* – see below.



Contents of the Machine Frame



Contents of the Workshop Frame

This technique does not modify the results of the simulation but allows the user to visualize what happens. In the same way it is possible to create lines of animation in order to visualize for instance the movement of a crane bridge really transported into a “secondary” frame.

For the model of the PrePreFabrication workshop almost each frame has animation points and/or lines. The main principle having been highlighted, we will not describe in detail these particular points/lines because they do not really contribute to improving the model but only to facilitate the visualisation.

The *Init* method in the root frame puts “tools” (crane bride, mechanized grippers and welding robots) in their position. The *Reset* method allows for the deleting of all MUs in the model when we want to run a new animation. Notice that each sub-frame in the model has its *Init* and *Reset* methods. They are all run simultaneously, which is an important difficulty during the programming because it is not possible to know the execution order of the programming code.

Three global variables are defined in the workshop: *Sequencage*, *NumRun* and *Stream*. The first one allows the user to select the kind of sequence (half-cell or half-workshop). This is a *string* variable. The second one is used to indicate the number of successive simulations to be executed. When this number is changed, the method *Num\_Run\_update* is automatically run to modify parameters of the *ExperimentManager* object. The third variable is very

important. To generate a random number eM-Plant starts from a number named Stream. A stream must be defined for each time-distribution used in the simulation – or more generally for each random function used. In the simulation each time we will need to use a stream value, we will use the stream value indicated in the root frame “Atelier”. If we run the simulation without any change of this value we will get each time the same result, despite the fact that the system is completely stochastic. The advantage is that this is much more convenient when we debug the program. If the results change at each run it would not be easy to re-create bugs in order to catch and delete them. Of course when we run successively the simulation we have to modify this value at each run. The object *ExperimentManager* and the method *NumRun\_update* make these changes.

### 3.2.5.3 The STK Frame

This object generates all pieces coming from small containers and that have to be built on the assembly with all their characteristics. Individual pieces and containers that we can find are:

- UPS containers:



These containers store UPS girders.

They are represented in eM-Plant by:



under the name of “BenneUPS”, pieces contained are called UPS.

- “Tôles” containers:



They contain all very small pieces (brackets, etc.).

They are represented in eM-Plant by:



under the name of *BenneTôle*, pieces contained are called BTôle

- UPC containers:



UPC containers are the most common containers into the workshop. This kind of container is divided into two parts: the first one for long pieces and the second one with doors for smaller pieces.

They are represented in eM-Plant by:



under the name “Benne UPC”. Pieces in the container are called UPCA (for the long ones) and UPCB (for the sort ones).

- “Mini PRS” container:

“MiniPRS” pieces (“*Profils reconstitués spéciaux*” in French – special girders) will arrive in identical containers that UPS containers and will be represented in eM-Plant by:



- The “palanqué”:

That French term includes plates that are brought into the workshop by fork lift. These plates are shorter than plates coming by the PM container. “Palanqué” is thus not really a container, these plates are stored directly on the ground. Nevertheless for the simulation we

cannot consider the ground as a container. Consequently we have to create virtual container to place these plates in. In the simulation, this virtual container will be represented by:



How are all these containers created into the simulation? The structure of the STK frame is the following one:

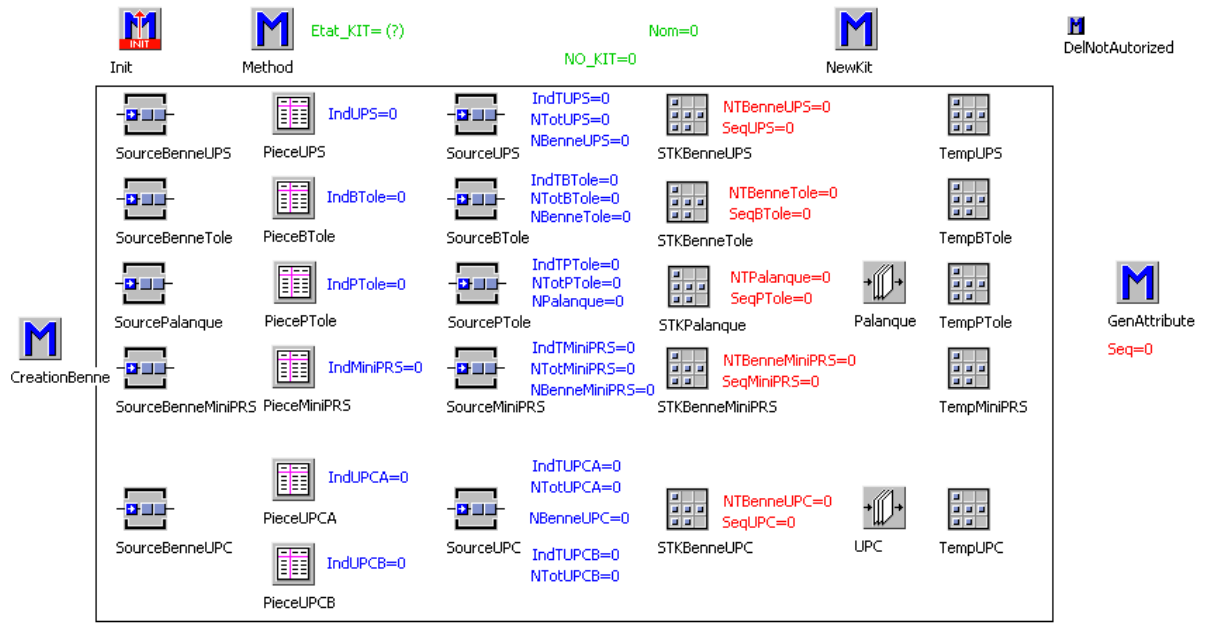


Fig. 53: The STK Frame

The frame contains also a list of methods that we will briefly describe in order to understand the overall functioning of the software. Detailed information is not given.

- The *Init* Method

Initialise all tables and variables of the Frame.

- The *DelNotAuthorized* Method

eM-Plant has an important weakness in its functioning way: there is no Undo function! This point could lead to disagreeable situation because if we are working in a frame and want to delete an object the whole frame could be deleted if the object was not correctly selected! To avoid this annoying situation when deleting a frame (the parent frame or a children frame) this method is run just before the operation. This method prevents the delete operation.

- The *NewKit* Method

This method is run when we want to create and bring all the containers of a new kit. The methods read into a table supply time to bring containers and launch the method *CreationBenne* for each kind of container.

- The *CreationBenne* Method

In input of this method we have the kind of container to bring and the corresponding kit. The method will search how many pieces are necessary for this kit. It could happen that the container is too small for the number of pieces. In that case we have to bring several containers of the same kind.

The first is thus to determine how many containers will be needed. Then we create these objects and put it on the corresponding object *SourceBenne* – example *SourceBenneUPS* if we create an UPS container. Then all the pieces are created on the corresponding object *SourcePiece*.

Notice that each kind of piece/container corresponds to a MU in eM-Plant as we can see in the next structure:

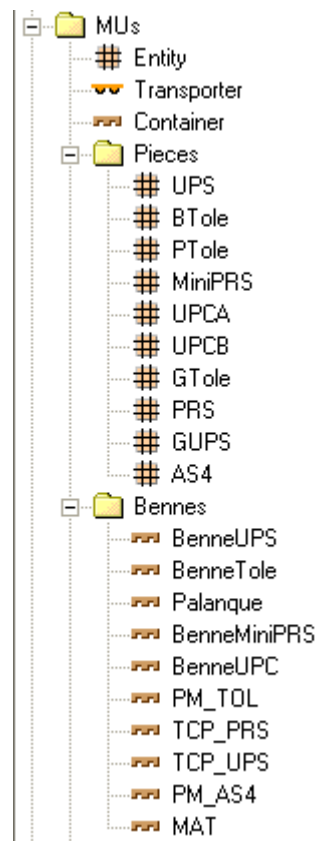


Fig. 54: MUs of the model

The basic icon of all the pieces is a simple square ■ but its dimensions will be adapted to respect the true length and width. These modifications are done in the method *GenAttribute* that will also save in each piece all its important characteristics (name of the piece, etc.).



Note that UPC containers are specific because they contain two compartments for two different kinds of pieces. Some modifications are thus necessary in the code to take into account these special containers.

- The *GenAttribute* Method

Each piece coming from the *SourcePiece* object will trigger successively the *GenAttribute* method. The method gives all the attributes to the piece: tacking time in seconds (AGRAFTIME), name of the container (“Benne”), main dimensions (DIM\_B, DIM\_L, DIM\_T), identification number (ELM\_NO), index of upper hierarchically sub-assemblies (IND), kit number (NO\_KIT), panel number (NO\_PAN), identification number of the assembly (PARENT), weight (Poids), its storing area (STK), parameter – Boolean – to indicate if the piece is the basic plate or not (TOLBASE), handling system – manually, with the crane bridge, with the mechanized gripper, etc. – of the piece (TRANSPORT) and the coordinates of the assembly on the working area (X\_Local, Y\_Local).

When creating a piece, we check if this is not the piece of a new sub-panel. In that case we save the simulation time in a table (table *PanInd* of the frame *Tables*) because this value is important to determine production time of all the assemblies of a similar sub-panel.

After, depending on the dimensions DIM\_L and DIM\_B of piece, we change the size of the icon to get the true visible dimensions of the piece – at a scale factor. If the piece is a basic plate we put it into the container otherwise, we put it into a temporary area. When all the pieces of a container are generated we transfer pieces from the temporary area into the container. The first piece to go into a container is the first to exit. With this tip we are guaranteed to always have the basic plate on the top of the container, which is much subsequently more convenient for the simulation.

When the container is full (or all the pieces created) we take the container and put it on the corresponding *STKBenne* object (example *STKBenneUPS* for the UPS container, etc.). Animation points are attached to this object and at this moment the container appears thus into the virtual workshop. We also give a list of attributes to the container: number of pieces (*NbrPiece*), number of Kit (*NO\_KIT*), name of the container (*Nom*) and number of the container (*NumBenne*) that allows the user to know the number of created containers since the start of the simulation in this STK area.

Here again a specific programming code has been developed for the UPC container. This is also the case for *Palanqué* that are not limited by the number of pieces but by the

maximum acceptable tonnage. We remember that these containers do not really exist but are simply stacked on the ground but it is normal to limit this stack. These characteristics have forced us to create specific objects for *Palanqué* and UPC containers: a *QueueFile* object – respectively named *Palanque* and UPC on the draw of the frame.

Different global variables (*IndUPS*, *IndTole*, etc.) are used in the two methods previously described.

### 3.2.5.4 The MAT Frame

As was the case for the STK frame, the MAT frame creates different kinds of pieces for the workshop. Furthermore plates coming from PM containers can be stacked in that area.

Pieces and containers that we can find are:

- TCP containers:



They are used to bring two kinds of pieces: PRS and long UPS.

Fig. 55: TCP containers

Practically these TCP come into the workshop by one DCH entry (DCH\_3 and after having removed PM containers) and pieces are thus stacked in the MAT area then we remove the TCP container.

They are represented in eM-Plant by:



Fig. 56: TCP containers in the simulation model

- The AS4 :

These objects are sub-assemblies already built in the PrePreFabrication workshop (the manual or the automated) and that are waiting to be mounted on another assembly. In practice

these pieces are not a bottleneck and we can suppose that they are already built and ready in the workshop, we do not have to model their production. Supply time of these objects is thus supposedly equal to 0.

This section explains the creation of these containers in the simulation. The structure of the MAT frame is the following:

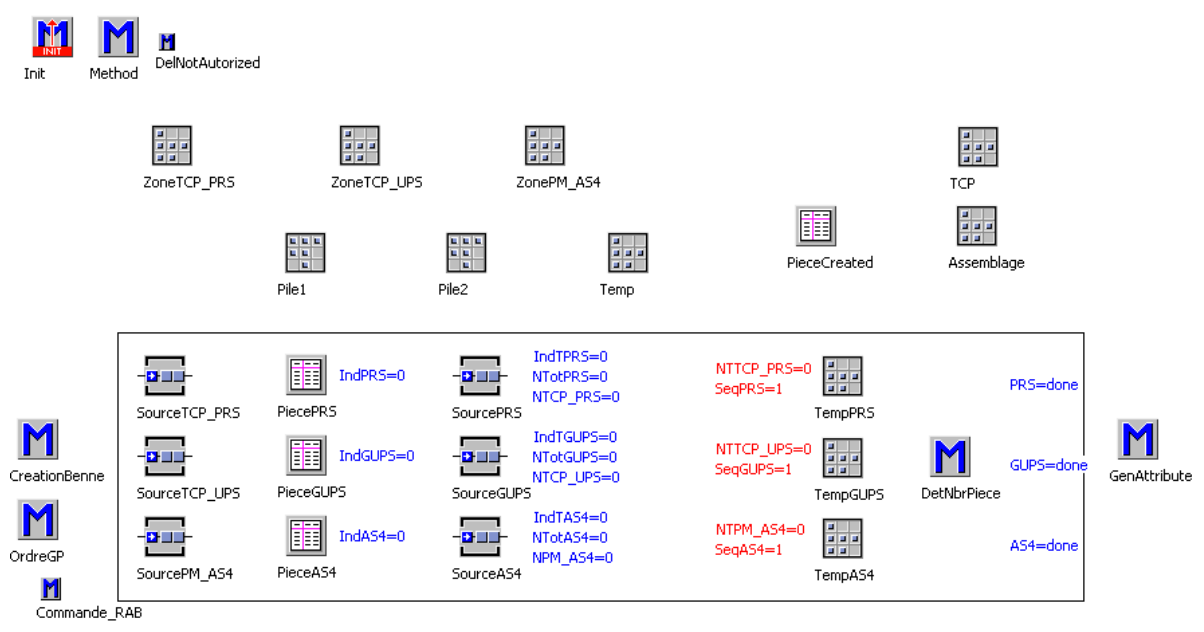


Fig. 57: The MAT Frame

The structure is very similar to the structure of the STK frame for the creation of containers. Nevertheless some differences remain because in the STK area we create containers relative to the kit to be created but here pieces in the TCP belong to different kits. When a needed piece is missing – not yet in the workshop – or when the container is empty we call the next container that empties eventually the TCP container. The procedure call is really different and that explains why we do not find here the *NewKit* method.

Three *Zone* objects are designated to receive containers that will store PRS pieces, long UPS and AS4. Object *Pile* can receive stacked plates. There are two *Piles* instead of one in the case where entry pieces will arrive on PM on two distinct stacks. The object *Assemblage* is used to store temporarily finished assembly waiting for free exit PMs. All these objects have animation points and consequently we can visualize them in the workshop.

- The *Init* method

Initialises all tables and variables of the Frame. Furthermore the method creates virtual containers that will receive stacked plates (*Pile1* and *Pile2*), finished assemblies waiting for free PMs and finally containers for TCP\_PRS, TCP\_UPS and AS4.

Finally this method runs the *CreationBenne* method once for each kind of container to supply the workshop with the first pieces to start the simulation.

- The *CreationBenne* method

This method is called by the *AgrafGP* method of the frame *Etat\_Kit* that gives the order to the crane bridge to take the piece to tack. To determine the number of pieces to bring and which pieces to take we have to look into the table *SeqAtelier*. Principle is to look into the list of kits given into this table and to fill containers according to this order until the TCP is full. That TCP will thus contain pieces of different kits. Rule of selection of kits is explained below in the section “The *CreationBenne* method” that will create the table *Piece* with the list of pieces to put into the TCP (or into the AS4 container). The first thing done by *CreationBenne* is thus to run this method *DetNbrPiece*.

Contrary to the STK frame it is useless to determine the number of containers because we build only one container at the same time. For practical reasons we will create one container for long UPS, one for PRS and one for the AS4 in the method *Init* and then we will always use these containers during the simulation – without deleting them and creating new ones.

The first step is to move containers situated on the *Zone* object (for example *ZoneTCP\_UPS*, *Zone\_TCP\_PRS*) and to put them on the corresponding *Source* object. Finally we create pieces on the *Source* Object. As for the STK frame at this step the pieces have no dimension (icon ■) and no attributes. On this *Source* object these pieces will trigger one by one the *GenAttribute* method that will do these operations.

- The *DetNbrPiece* Method

As explained previously pieces to be put in *PM* are determined by the *SeqAtelier* table. To determine that list of pieces we use intermediary tables (*Kit\_TCP\_PRS* or *KitTCP\_UPS* or *Kit\_AS4* depending on the kind of container) that contain all the pieces of the same kind for all the simulation. We extract from this table the appropriate list and copy the table into the *Piece* table (example *PiecePRS*, *PieceGUPS*) of the *MAT* frame.

	integer 1	integer 2
23	3	44
24	4	45
25	14	46
26	15	47
27	16	48
28	17	49
29	18	50
30	5	51
31	6	33
32	7	34
33	8	35
34	9	36
35	10	37
36	11	38
37	12	39
38	13	40
39	19	41
40	20	52
41	201	242
42	202	243

This example shows the creation of the table Piece (below) in function of the table *SeqAtelier* (see to the left) for a *TCP* limited to 15 pieces.

string	NO_KIT	NOPAN	IND	WEIGHT	DIM_L	DIM_B	DIM_T	X_LOCAL
1	44	3110	1	1157	6147	2205	12	3
2	44	5997	0	690	4550	2794	7	10
3	46	2019	1	670	3969	2513	10	3
4	46	2019	1	905	3265	2900	16	3
5	46	3110	1	1036	5414	2205	12	9
6	47	3110	1	1157	6147	2205	12	9
7	47	5997	0	688	4554	2793	7	3
8	47	5997	0	281	3143	2275	5	10
9	48	2019	1	639	3767	2513	10	3
10	48	2019	1	883	3154	2900	16	3
11	48	3110	1	1156	6141	2205	12	9
12	49	2019	1	588	3461	2513	10	9
13	49	2019	1	825	3429	2854	16	9
14	49	3110	1	1157	6147	2205	12	3
15	50	2019	1	686	6146	2000	10	2.5

- The *GenAttribute* method

The method principle is exactly the same as its homonym method for the *STK* frame: pieces receive attributes, their icon is changed to the right dimension, pieces are placed in corresponding containers that are themselves put in their original position – the *Zone* object.

### 3.2.5.5 The DCH frame

That frame has two different parts because *PM* has a double use. Either they are used for the entry of pieces or for the exit of finished assemblies. By default the two upper *DCH* (*DCH\_1* and *DCH\_2*) are dedicated to the moving away of assemblies and the third one (*DCH\_3*) to the incoming pieces. This choice can of course be changed. In both case the same containers will be used – the *PMs*.

- The *PM* container:

Containers are brought into the workshop by special trucks. They enter the workshop, then put down containers and leave the workshop. The trucks can take two containers and these containers will always be handled in pairs.



**Fig. 58: PM containers**

They are represented in eM-Plant by :



under the name of *PM\_TOL*.

Pieces in *PM* are either long plates (*GTole*) in entry or finished assemblies (*Assemblage*) at the exit. Notice that *PMs* are always in pairs. They come in and out together.

The structure of the *DCH* frame is given in Fig. 59.

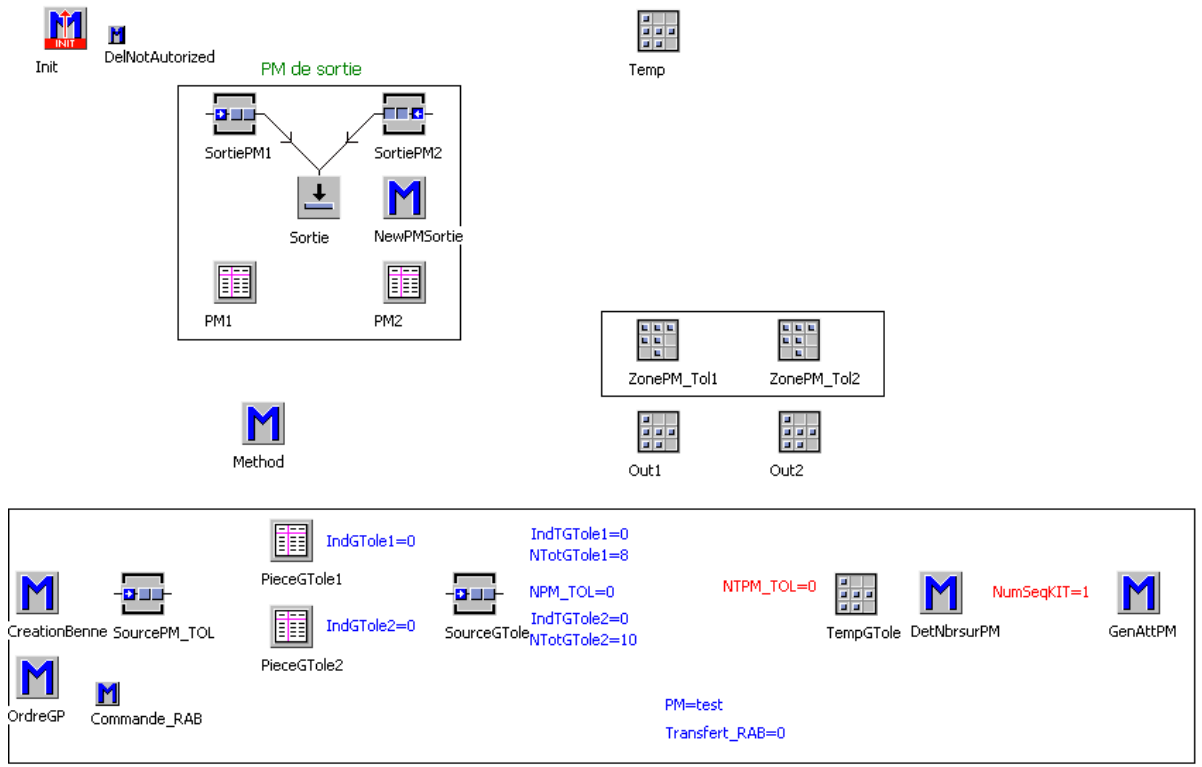


Fig. 59: Structure of the DCH frame

The upper part concerns the exit of the PM, the lower one their entry. For the entry the structure is similar to the STK and MAT frames. Both entries PM are still on the object *Zone* (*ZonePM\_Tol1* and *ZonePM\_Tol2*), the exit on the *SortiePM1* and *SortiePM2* objects. These objects also have animation points and will be visible in the root frame *Atelier*.

- The *Init* method

Initialises all the tables and variables of the Frame. Furthermore:

- for the entry PM: run the *CreationBenne* method that will bring long plates into the workshop;
- for the exit PM: creation of exit *PMs* on the *SortiePM1* and *SortiePM2* objects.

- The *CreationBenne* method

This method is run only for entry DCH. The structure is the same as that in its homonym in the MAT frame. At the first step the *DetNbrsurPM* method is run to create the list of pieces to put on the PM. In entry we have only one stack of plates although we have two PMs (actually some long plates are stacked on the two PMs simultaneously). Nevertheless we have modelled the possibility to have two entry stacks. The second table *PieceGTole2* is used in that case. From this table we deduce the number of pieces to be

created, move the PM from the *ZonePM\_Tol1* to the *SourcePM\_Tol* object and finally create pieces on the object *SourceGTole*.

Notice that the fact of modelling only one stack allows us to use only one of the two containers. The second is created and displayable in the workshop but in practical terms do not contain any pieces.

- The *DetNbrsurPM* method

This method is run only for the entry PM. It has the same structure as its homonym in the MAT frame. The table *SeqAtelier* is used in order to choose kits and from there pieces that will fill the PM.

- The *GenAttribute* method

This method is used only for entry PM. The structure is the same as its homonym in the MAT frame: pieces receive their attributes, their icon is changed to the correct size, pieces are put into the container which is itself put on the *ZonePM\_Tol1* object.

Furthermore this method feeds virtual stacks necessary for the handling crane bridge. These virtual stacks are indispensable! In concrete terms the crane bridge receives a list of orders to achieve. These orders are saved in the *Orders* table of the frame *Pont*. Suppose that at the time *x* we wish to obtain the second plate situated in second position on a PM stack. If the table *Orders* is not empty the crane bridge will first of all accomplish all previous orders before executing our request. Now it is really possible that one of these orders is to stack down into the area MAT – and thus to put our plate in that area. When the crane bridge will try to satisfy our request it will not find the piece because the piece is not at its initial position! To solve this problem – that can happen very often in the simulation – it is thus necessary to create virtual stacks that represent the PM state when all orders contained in the *Orders* table have been carried out. These virtual stacks allow the crane bridge to avoid these tricky situations.

At each modification of a stack (creation of a new piece, move on a working area, etc.) we have to modify these virtual stacks. The *GenAttribute* method does these modifications for the creation of ordered pieces.

Notice that these virtual stacks are necessary not only for the PM but also for the MAT and RAB areas where stacked can also be found.

- The *NewPMSortie* method



This method is used to manage the exit of the finished assembly – so DCH\_1 and DCH\_2 objects. PMs that are waiting to be filled are located on *SortiePM1* and *SortiePM2* objects. These objects do not move towards the *Sortie* object because the entry is artificially blocked – this is a default parameter of the object (it is possible to lock or unlock the entry gate as we want). When PMs are full or they contain all the assemblies of the same sub-panel we must move them away and bring new empty PMs. When this is the case we unlock the *Sortie* object. This object runs automatically the *NewPMSortie* method that will move away PMs.

When two PMs are on the *Sortie* object and have thus left the workshop we lock again the entry of the object. Both objects *SortiePM1* and *SortiePM2* have an attribute named *Etat* (meaning State, string variable) and we modify its value in “Moving Away” (“Evacuation”) – this variable is useful for other methods. After that, we determine the time needed to supply the other containers with the *Times* method of the *Table* frame.

We then stop the method during its runtime duration. Notice that this waiting time represents the evacuation time and the supply time although the fact that containers will visually disappear as soon as they are full, as if the moving away time was zero. When the time has elapsed we create again empty containers on objects *SortiePM1* and *SortiePM2* and we delete their value of the variable *Etat*.

When new empty PMs have arrived the first thing to do is to fill them with finished assemblies waiting to be moved away – if they exist. We try to put the first assembly arrived in the MAT area and we “book” one or two PMS depending on the size of the longer sub-panel to which it belongs. When a PM knows that the sub-panel of assemblies will arrive, the *Etat* attribute becomes “Occupe” (meaning Occupied) for the *SortiePM* object corresponding. When the PM is full, that variable becomes “Plein” (meaning Full). When a sub-panel needs two PMs we will put all assemblies on the first one but fix the variable on the second as “Plein”.

We will finally put other assemblies – still by their chronological arrival order – if there is space and if the containers are not full. Notice that we must be careful: waiting assemblies in MAT may have already been put on a new PM from another DCH frame – but are not there yet because of crane bridge overloading. That condition must be checked. Notice that when PMs arrive the crane bridge was maybe bringing an assembly to the MAT area. We thus have to modify the crane bridge order so as to bring the PM.

This example – and the problem of virtual stacks – clearly shows the difficulty of the development of a complex simulation model. Even if the modelling process seems clear and easy there are often different non direct effects to take into account. These examples are simple but that kind of effect can sometimes lead to non-realistic situations that are very difficult to solve.

The method thus runs a list of orders to the crane bridge to execute. These orders are saved in the *Orders* table of the frame *Pont* – the frame of the side where the DCH frame is. At the end of the method and if the crane bridge was at rest it is necessary to run the Action method of the frame *Pont*. This last method gives the order to the crane bridge to accomplish commands of the *Orders* table.

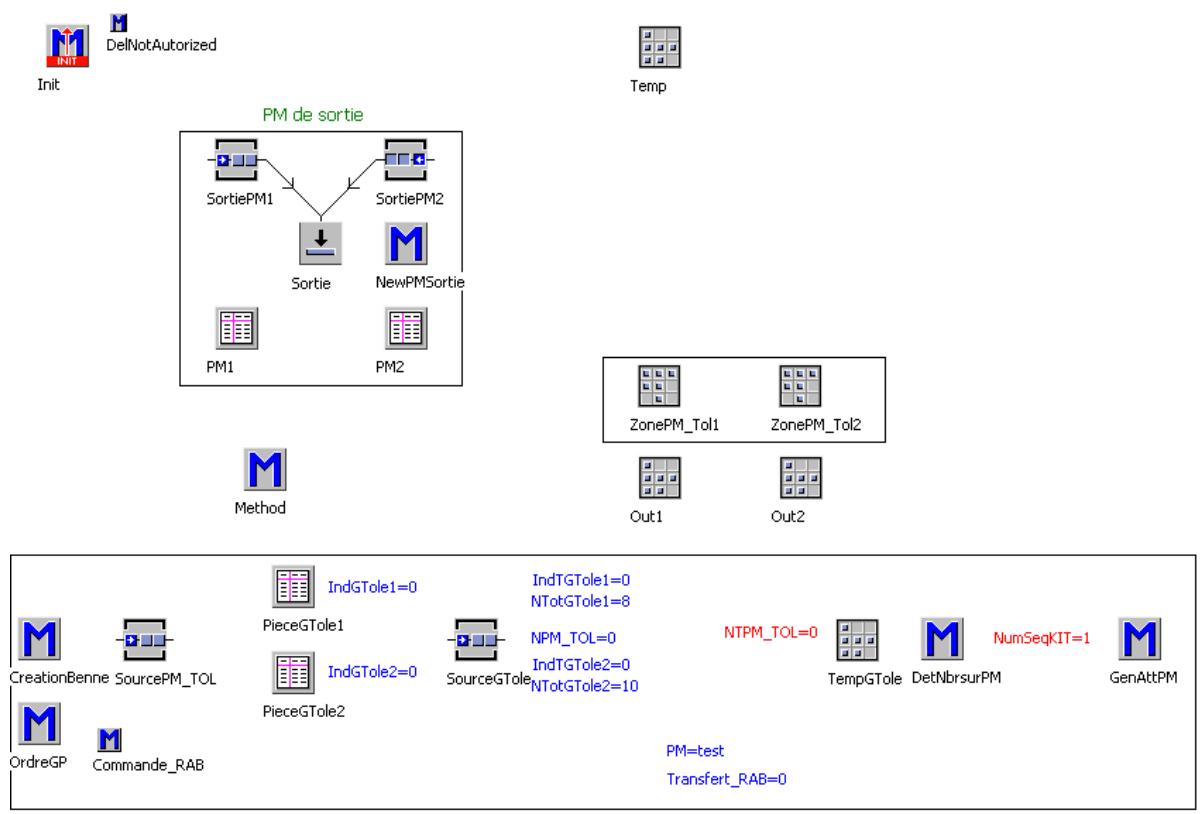


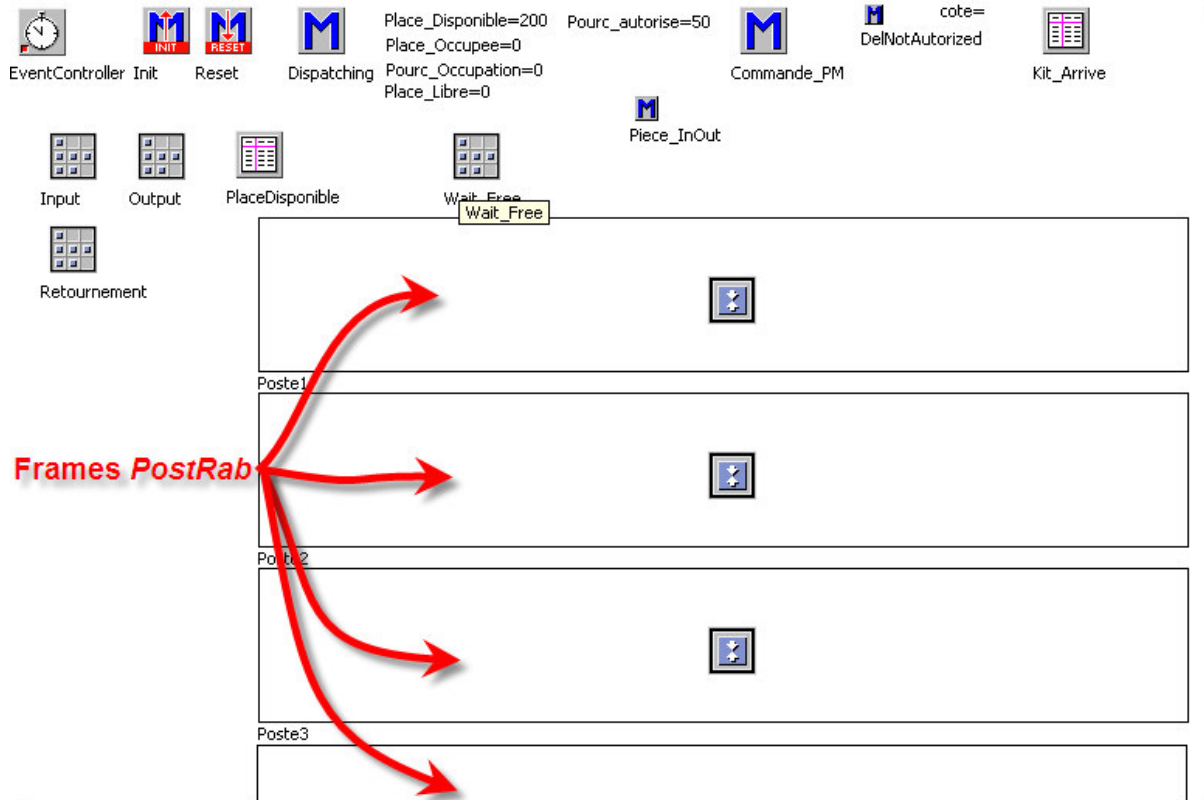
Fig. 60: The DCH Frame

### 3.2.5.6 The RAB Frame

The frame is used to model joining areas. Plates come from PM to be manually welded together by workers. The resultant plate will be the main plate of the assembly – the basic plate. This is the first plate brought to the working area before the girders and other smaller plates. All plates in the joining area do not necessarily come from the PM containers. Smaller

plates can sometimes be brought with a forklift. For instance this is the case for plates named “Palanque”.

The structure of the frame is given in the following figure:



**Fig. 61: The RAB Frame**

The surface space is obviously limited in this area and we cannot tolerate an infinite number of plates in the joining area. A limitation should thus be implemented. The problem is a space allocation problem that is not obvious to solve. In other ways we know that this joining area is not really a bottleneck in the simulation. Consequently it is not really crucial to model the area with very high precision. Thus we can suppose that the surface limitation is just a number to respect: we define a maximal surface utilization in square meters and we count the surface of all plates in the joining area. This rule is simplified and much easier to model than developing a space allocation solver only for this area. Even if the area is not a bottleneck we need to model it because this area occupies the crane bridge for a long time.

The production sequence is the following one. First of all if there is available space, plates are brought either by the crane bridge – most of the time – or by a fork lift for smaller plates. Afterwards, some preparation activity plates are welded together by workers to constitute the basic plate. Sometimes we need more than two plates to build it. Secondly the crane bridge is called in order to turn the plate over. Once the operation is finished workers

can do the welds on the opposite side of the plate. Each plate has thus two welds: one on each side. When the basic plate is finished it remains at the RAB area waiting for a cell to be free to start the assembly production.

Welding operations can be done only on a frame named “PostRAB” – see next section. This last frame manages operations for only one assembly. The RAB frame contains six *PostRAB* frames – named *Poste1*, *Poste2*, *Poste3*, *Poste4*, *Poste5* and *Poste6*. They are not all visible in Fig. 61. In other words it means that it is not possible to build simultaneously more than six different basic plates. However we can have more than six plates in the area if the space limitation is respected. These plates are then just waiting for a free *PostRAB*.

Different variables are defined in the workshop:

- *Place\_Disponible*: Indicates the free surface in the area – in square meters;
- *Place\_Occupee*: Indicates the occupied surface in the area – in square meters;
- *Pourc\_Occupation*: Indicates the ratio between two previous variables;
- *Pourc\_Autorise*: If the *Pourc\_Occupation* value is inferior to this variable the frame calls a new PM – to fill the joining area. This value is important because it is the main factor to decide when PMs are called;
- *Place\_Libre*: Number of *PostRab* free;
- *Cote*: Side of the joining area – A or B.

Methods of the frame are briefly explained in the following sections.

- The *Init* method

Initialise tables and variables of the frame.

- The *Reset* method

Reset values of tables of the frame.

- The *DelNotAuthorized* method

As explained before the method prevents the user accidentally deleting the frame.

- The *Dispatching* method

Dispatch entry plates at the right location. If the plate is the first of an assembly the method assigns it at a free *PostRAB* object. If there is no free *PostRAB* the piece is temporarily stored on the *Wait\_Free* object. The method also saves important information

about this dispatching into the *PlaceDisponible* table. An update of variables of the frame is also done.

- The *Piece\_InOut* method

The method is run when a piece enters or exits the frame to update variables of the frame – surface occupied, number of *PostRAB* free, etc.

### 3.2.5.7 The “PostRAB” Frame

This frame models the joining operation of two or more plates. This operation is done manually by workers.

The structure of the frame can be seen in the following figure:

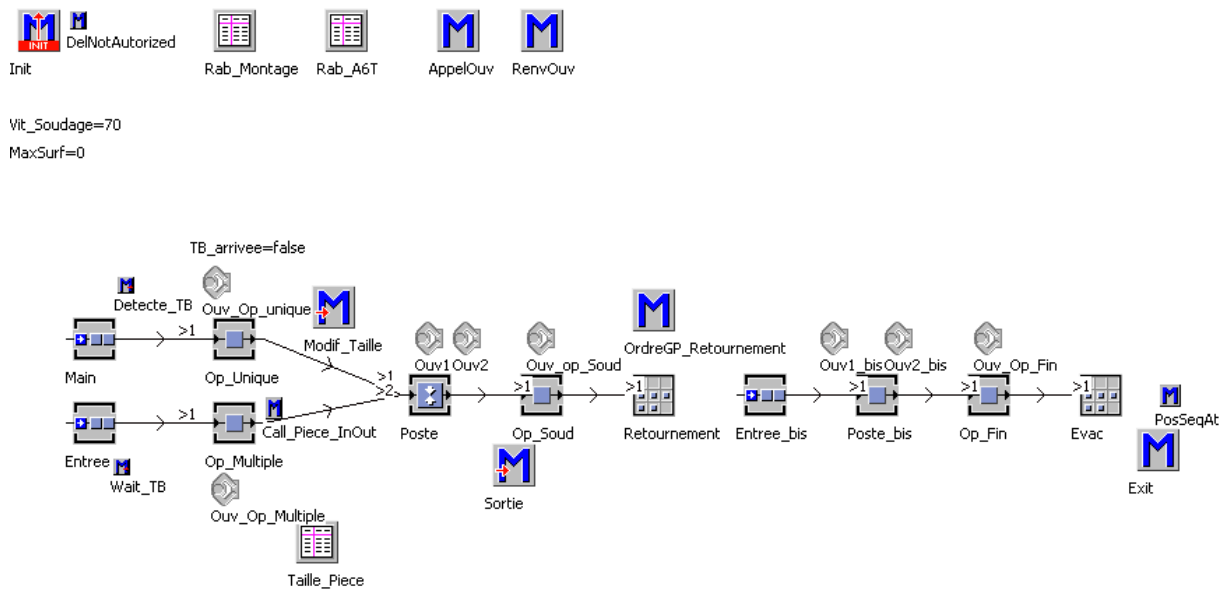


Fig. 62: The "PostRAB" Frame

- The *Init* method

The method initialises variables and tables. Some time parameters are also set in the model – on “machine” objects.

- The *AppelOuv* method

The method indicates in the “*Resource*” frame that a worker is welding in this joining area. This method is triggered each time a worker is called.

- The *RenvOuv* method

The method indicates in the “*Resource*” frame that a worker has finished welding in this joining area. This method is triggered each time a worker is sent back to the WorkerPool.

- The *Modif\_Taille* method

When two plates are welded we have to create a new MU with the right dimensions or to adapt the dimensions of the first plate. It is the second solution that has been chosen.

- The *Sortie* method

A special method that checks some attributes of pieces.

- The *OrdreGP\_Retournement* method

Method launched when plates have been welded on one side. An order is given to the crane bridge to come and turn the plate over. When this operation is done workers can come back to finish the welding.

Others methods of the frame are too specific to be explained in detail. They are generally used to manage particulars or to change some attributes to inform other methods.

#### **3.2.5.8 The “Ilot” Frame**

In this frame we find all the management of the successive operations to be carried out on the two half-cells: different orders are launched from here and also orders for the welding robots that will move from one side to the other one. Operation times of each step are also saved here in global variables.

The structure of the frame is the following one:

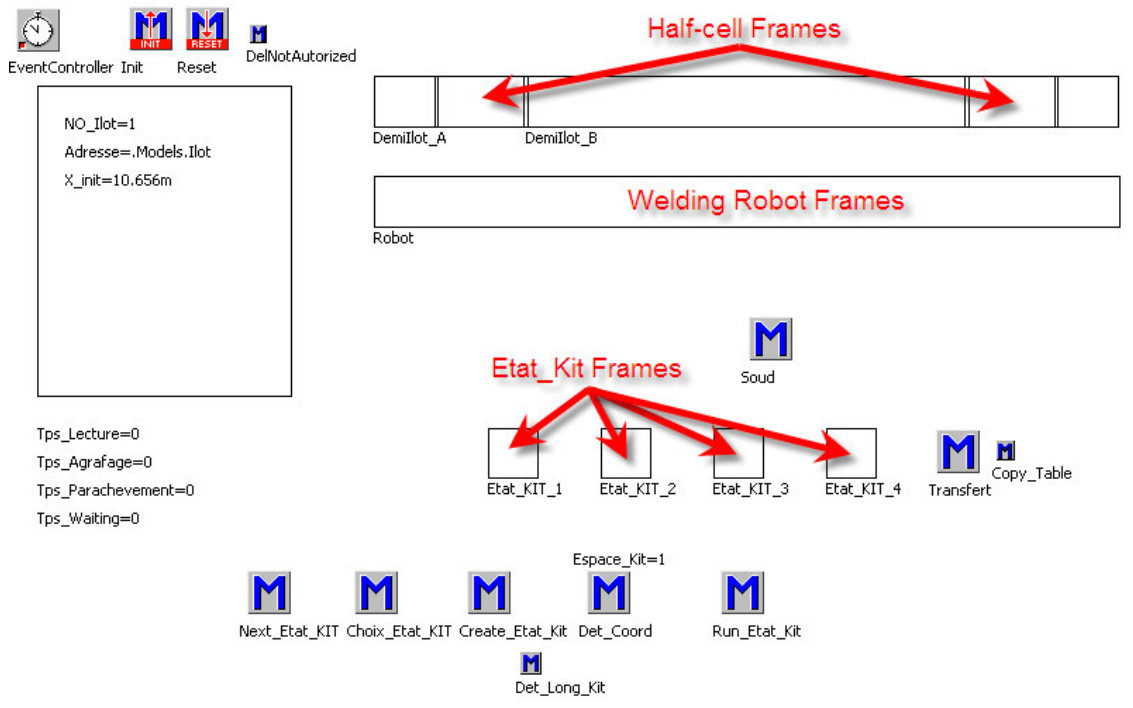


Fig. 63: The "Ilot" Frame

The frame contains one frame *Robot* and two frames *Ilot\_Demi* – half-cell.

On a cell there are always four fundamental frames: the *Etat\_Kit* frame. This frame manages the production of a kit on a working area. By clicking on one of these frames we open it and have thus access to detailed information on the process evolution.

- The *Next\_Etat\_Kit* method

In input of the method we enter an *Etat\_kit* of the frame. The method runs with the same entry as the method *Choix\_Etat\_Kit*. Depending on the entrance *Etat\_Kit* – if it is assigned or not – and on the state of its next *Etat\_Kit* the method *Choix\_Etat\_Kit* could also been launched for its neighbour. This method manages different rules.

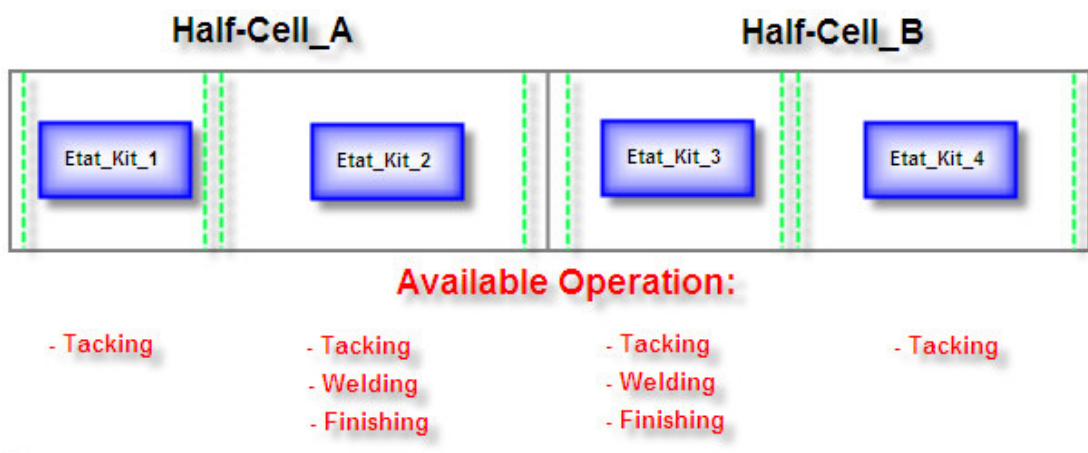


Fig. 64: Schematic representation of a cell

Objects *Etat\_Kit\_1* and *Etat\_Kit\_4* are at the extremity of the cell. Most kits are tacked in these areas. When kits on “interior *Etat\_Kit*” are moved away kits are transferred from the external area to the internal area. Then they are welded, finished and finally moved away. The length of the kit ordered depends on the assemblies that constitute it. When a kit is assigned at a working area delimitations are visible on the simulation – even if assemblies are not there yet. These limitations are vertical dot lines in the simulation and in Fig. 64. They are calculated into the method *Trace\_Borne* of the frame *Etat\_Kit*. We thus have to keep in mind that working areas thus have variable working space!

Sometimes it might be the case that the kit is too long for the external *Etat\_Kit* area. In that case we work only in the internal working area. When the previous kit is finished we can thus bring this long kit into the internal working area. All the operations are thus done there: tacking – welding – finishing.

- The *Choix\_Etat\_Kit* method

Again in entry we must specify an *Etat\_Kit*. For this working area the method will determinate which kit must be assigned. The method checks the dimensions of the kits and available space of the working area. Finally the method launches the *Create\_Etat\_Kit* method with parameters: the *Etat\_Kit* and the number of the kit.

- The *Create\_Etat\_Kit* method

Modify some attributes of the *Etat\_kit* given in entry in function of the kit. For instance the variable *No\_Kit* of the frame *Etat\_Kit* is set to the number of the kit.

Run the *Det\_coord* method – see explanations in the corresponding section.



Depending on the kit – if it is used for the initialisation or if it is a “studied” kit – the method runs the method “Preparation” of the *Etat\_Kit\_Frame* or the method *Run\_Etat\_Kit* of this frame.

- The *Run\_Etat\_Kit* method

This frame manages the succession of operations on the *Etat\_Kit* given in entry. It successively runs the method *Preparation*, *LecturePlan*, *Agrafage*, and *Transfert*. The first three methods are located in the *Etat\_Kit* frame.

It also checks if the next working area is free or not to eventually launch the *Next\_Etat\_Kit* method on the area.

- The *Transfert* method

When a kit is moved from the external working area to the internal working area we also have to transfer all the information from the first *Etat\_Kit* to the second one. This operation is done by the *Transfert* method. Orders for the transfer are also given to the crane bridge and the exchange of information is only executed when the last assembly has been transferred.

- The *Soudage* method

The method manages the welding operation. The crane bridge is called successively for each assembly for the welding. When the robot is on an assembly the “Assemblage” MU is moved from the *PostAss* frame on the *In* object of the frame *Robot*. Once the welding is done we move the assembly from the *Robot* frame again into the *PostAss* frame.

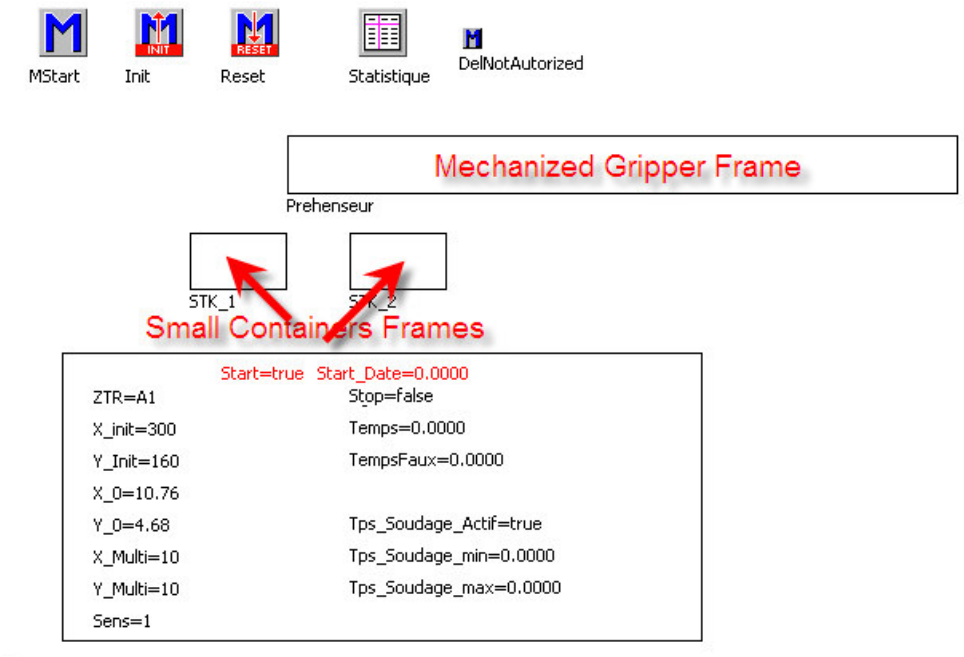
- The *Det\_Coord* method

For the number of a given kit this method returns the obstruction length in meters. This value is used to calculate the new coordinates of assemblies on the working area.

### 3.2.5.9 The “Ilot\_Demi” Frame

This frame corresponds to the behaviour of a half-cell.

The structure of the frame can be seen in the following figure:



**Fig. 65: The "Ilot\_Demi" Frame**

The frame contains one frame *Prehenseur* and two frames STK – STK\_1 and STK\_2. The frame STK\_1 manages containers of the external working area and the frame STK\_2 manages containers of the internal working area.

### 3.2.5.10 The “Etat\_Kit” Frame

When a kit is assigned to a working area we need to manage its process in a frame. This frame plays that role. All the information of the kits is stored in different tables and variables – number of assemblies, number of pieces for each assembly, number of pieces tacked, number of piece arrived, etc.

The structure of the frame can be seen in the following figure:

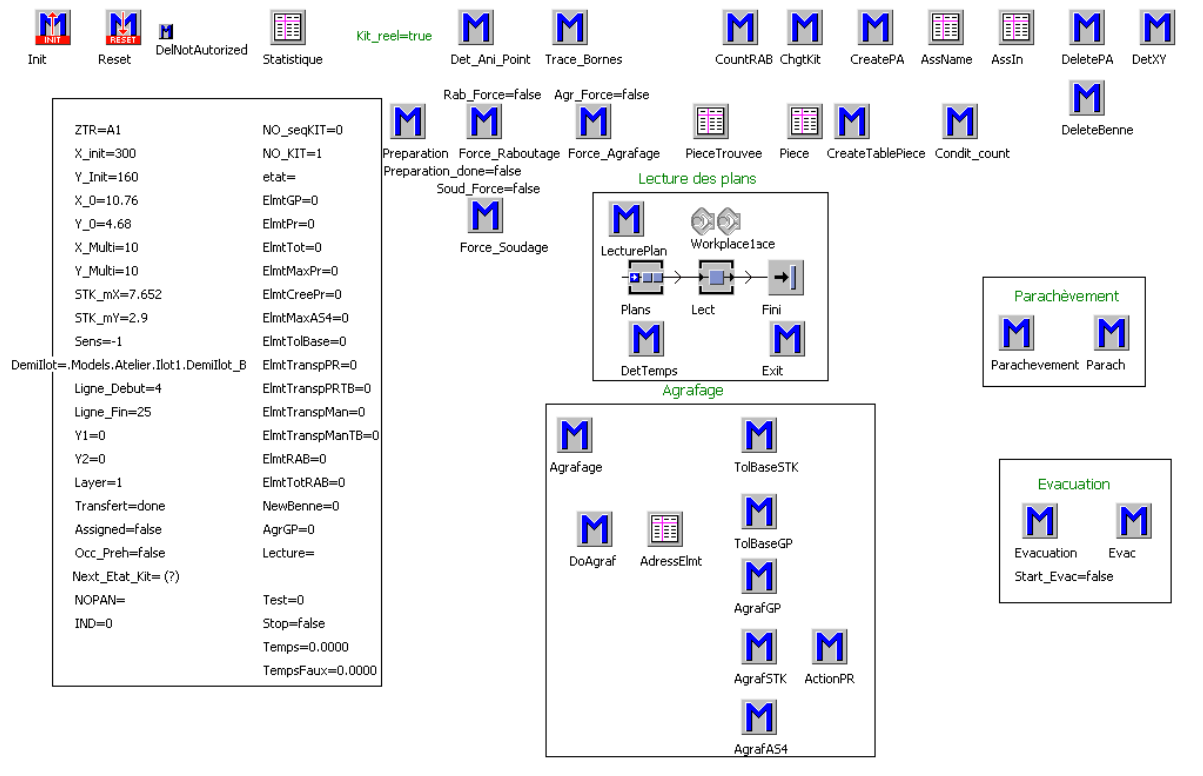


Fig. 66: The "Etat\_Kit" Frame

The frame contains four fundamental methods: *LecturePlan*, *Agrafage*, *Parachevement*, *Evacuation*. The first one corresponds to the first operation before starting the work: reading assembly maps by the workers. Now this operation also includes preparation of the working area before receiving the first assembly. The second one manages the tacking operation: calls workers, mechanized grippers, orders new containers, etc. The third one manages the finishing operation and the last one gives orders to the crane bridge to move away finished assemblies.

This frame contains plenty of methods and only the fundamental ones will be explained in detail.

- The *CreatePA* method

The method manages the creation of all frames *PostAss*. We have one *PostAss* frame for each assembly of the kit. The frame is created and placed at the central location of the assembly on the root frame *Atelier*.

- The *DeletePA* method

When a kit is finished the method deletes all corresponding *PostAss* frames. These frames thus remain temporarily in the simulation model.

- The *DetXY* method

If assemblies do not have predefined coordinates in the working area – in other words if the user did not enter this data – the method automatically calculates it. The goal is to minimize the total length of the kit.

- The *Force\_Raboutage* method

When we create the initialisation of the workshop some kits could be assumed to be already joined. This method adapts all frames, variables, tables to create the kit at the end of the joining area. A Boolean variable *Rab\_Force* indicates if we have to use this method or not.

- The *Force\_Agrafage* method

The method does exactly the same operation as the previous method but sets the kit at the end of the tacking operation. A Boolean variable *Agr\_Force* indicates if we have to use this method or not.

- The *Force\_Soudage* method

The method does exactly the same operation as the previous method but sets the kit at the end of the welding operation. A Boolean variable *Soud\_Force* indicates if we have to use this method or not.


- The *Traces\_Bornes* method

A cell contains a maximum of four working areas. Due to the disparity of kits we cannot consider that these working areas are fixed. When we know that a kit will be allocated to a working area we straightaway book the appropriate space. Limitations are represented in the simulation by green vertical dot lines – see Fig. 64. The *Traces\_Bornes* method automatically determines these limits and draws it on the root frame.

### 3.2.5.11 The “PosteAss” Frame

This is the only temporary frame of the workshop. This frame is created for each assembly to build and its coordinates correspond to the position of the assembly in the working area. The frame contains all the objects to be modelled, set-up time, tacking time, finishing time done by workers of the cell – with eventually a mechanized gripper for the tacking operation. Remember that the welding is done in the special frame *Robot*.

This is also the only frame (except the root frame *Atelier*) that has an icon/representation that is not completely transparent. In this case the representation of the

frame in the simulation is: . If we click on it we will open the corresponding frame. That icon allows the user to see straightway where assemblies of a kit are situated; even if the basic plate has not yet been brought.

The structure of the frame is the following one:

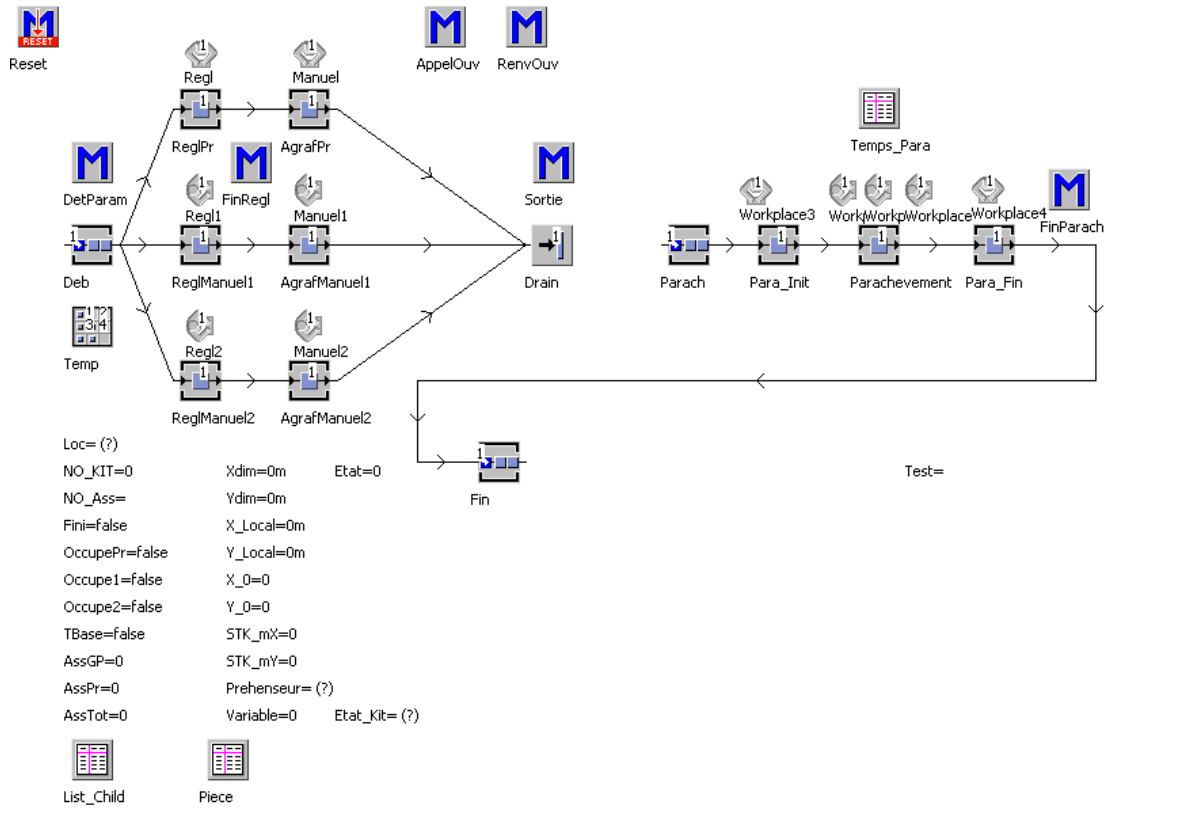


Fig. 67: The "PosteAss" Frame

Objects *ReglPr* and *AgrafPr* are used to model set-up and tacking operations with the mechanized gripper. Manually these operations are done by objects *ReglManuel1* and *AgrafManuel1*. When the mechanized gripper is no longer being used and two workers are working simultaneously on the assembly we use objects *ReglManuel2* and *AgrafManuel2* for the second worker. Notice thus that the maximum number of workers on an assembly is two. That is not restrictive because in practise this is also the case, and workers prefer to work on different assemblies to not get in each other's way. At each of these three “production lines” is associated a string variable *OccupePr*, *Occupe1* and *Occupe2* to know if they are used by workers.

We also find two tables in the frame. The first one (*Liste\_Child*) contains the fields ELM\_NO, QUANTITY, CONDIT and TRANSPORT of the YL\_KIT\_TABLE for the considered kit. The second table (*Piece*) counts the number of pieces still to be tacked.

Pieces to be tacked are stored on the *Deb* object that will trigger the *DetParam* method at the exit. The *ReglPr* object triggers at the exit the *FinRegl* method (a method that concerns only the mechanized gripper). The three objects *AgrafPr*, *AgrafManuell1* and *AgrafManuel2* trigger the exit method *Sortie*.

Right objects concern the finishing operation. The method *FinParach* is executed at the exit by the object *Parachevement*.

Notice that each object modelling an operation is linked to a workplace object. As a consequence it is indispensable to have a free worker to accomplish the task. They also have animation points linked to the root frame.

The *Fin* object is used to store temporarily an assembly. An animation point is linked to the object.

Finally notice variables *Loc* that give us the address of the half-cell to which the assembly belongs. Effectively the object *PosteAss* is created directly in the root frame and it thus has no information about the half-cell to which it belongs! In the same way the variable *Prehenseur* gives the address of the *Prehenseur* frame.

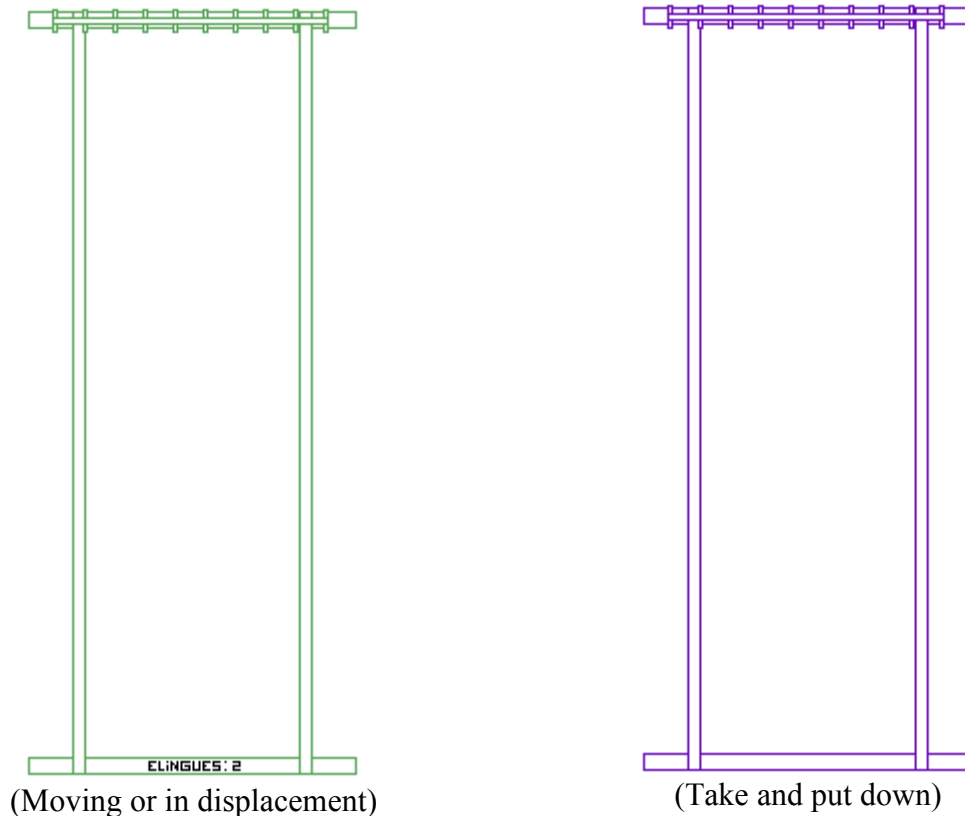
- The *DetParam* method

Each piece that must be tacked is put down on the *Deb* Object. That event will trigger the *DetParam* method.

First of all the method locks the entry of the next three objects. If the piece is a basic tool (the first one of the assembly) the variable *TBase* (Boolean variable) becomes true. In that case tacking time is fixed to zero but the set-up time is still present. Indeed in reality workers put down the piece and then make it fit with adapted tools. We record also plates' dimensions in *Xdim* and *Ydim* variables. These dimensions are useful to create the MU object *Assemblage* – the assembly.

If it is not a basic plate, set-up and tacking time are calculated thanks to the method *Time* of the frame *Tables* and with the length of pieces directly read from the attribute of the piece – *DIM\_L*.

If the piece comes from the crane bridge (this information is given in the attribute *Transport* of the piece) then the crane bridge stays in place during the tacking – to hold the piece. In that case the tacking is done by the mechanized gripper. During this time we modify the icon of the crane bridge, which becomes purple.



**Fig. 68: Representation of the crane bridge**

As for the mechanized gripper the icon becomes purple when the crane bridge takes or puts down pieces.

Finally we open one of the three objects *ReglPR*, *ReglManuel1* or *ReglManuel2* depending on each case to treat the piece.

- The *FinRegl* method

At the end of the set-up operation the piece – if it needs to use the mechanized gripper – triggers the *FinRegl* method. It also changes the mechanized gripper to blue – the set-up is considered as being a part of the put down stage.

If the piece was put down by the crane bridge we will give it back its original icon. After that, we have to call the mechanized gripper to tack this piece – except if it a basic plate – which is done by running the *GoX* method of the corresponding *Prehenseur* frame.

Notice that we delete the value of the variable named “*Depose*” of the *Pont* frame in order to indicate to the crane bridge that it is free again to perform a new task.

- The *Sortie* method

Each piece at the end of the tacking operation (or set-up operation for basics plates) triggers this method.

First of all an update of the *Piece* table is done to indicate that a piece has been tacked. If the piece is a basic plate we have to create the MU that will represent the assembly with its own characteristics. These attributes are NO\_KIT, X\_Local, Y\_Local, NOPAN, IND, STK, Dim\_L and RowPanInd. The last attribute is relative to the *PanInd* table of the *Tables* frame. This table contains information about all of the sub-panel in the simulation – entry hours in the workshop, exit hours, etc. We have one row in the table by sub-panel and the variable RowPanInd is the index of the table. This attribute allows the software to access easily the correct row of the table. The MU *Assemblage* is created on the *Fin* object and its dimensions are fixed by xDim and yDim variables previously calculated in *DetParam*. By default the colour of that assembly is yellow. The colour will change when all pieces will be tacked – assembly ready to be welded.

The next step is to modify the *List\_Child* table that contains the list of pieces to tack: if the field QUANTITY is greater than one unit we reduce this value of a unit, if it was the last piece with this nomenclature we delete the corresponding row in the table. When the table is empty it means that all pieces have been tacked. In that case we change the colour of the assembly:



Fig. 69: Representation of a plate

The value of the *ElmtTot* variable in the *Etat\_Kit* frame is also reduced to one unit. The worker – or the mechanized gripper – is now free and we can adjust the value of the *Occupe1* – or *Occupe2* or *OccupePr* – variable in false.

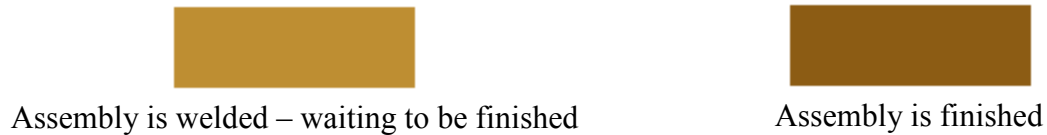
If the piece was not a basic plate and came from the crane bridge, it means that it was tacked by the mechanized gripper. Thus we send it to the centre of the cell with the *GoX* method. The crane bridge has maybe just brought the last piece (without considering the AS4 that are tacked later) and thus has modified the *AgrGP* value of the *Etat\_Kit* frame in “AgrafGP\_one” just after having put down the piece – see *TolBaseGP* method. In that case we can finally change the *Etat* variable in “agrafGP\_done” because this piece is now tacked.



In the same way the crane bridge has maybe just brought the last AS4 and thus modified *AgrafGP* in “agraf\_done”. We can change the *Etat* value in “agraf\_done”.

- The *FinParach* method

This method is executed when the assembly has just been finished – the call for this method is done with the *Parachevement* object. We thus modify the assembly icon:



**Fig. 70: Representation of an assembly**

Then the method launches the *Evac* method in the *Etat\_Kit* frame that will give the order to the crane bridge to move away this assembly.

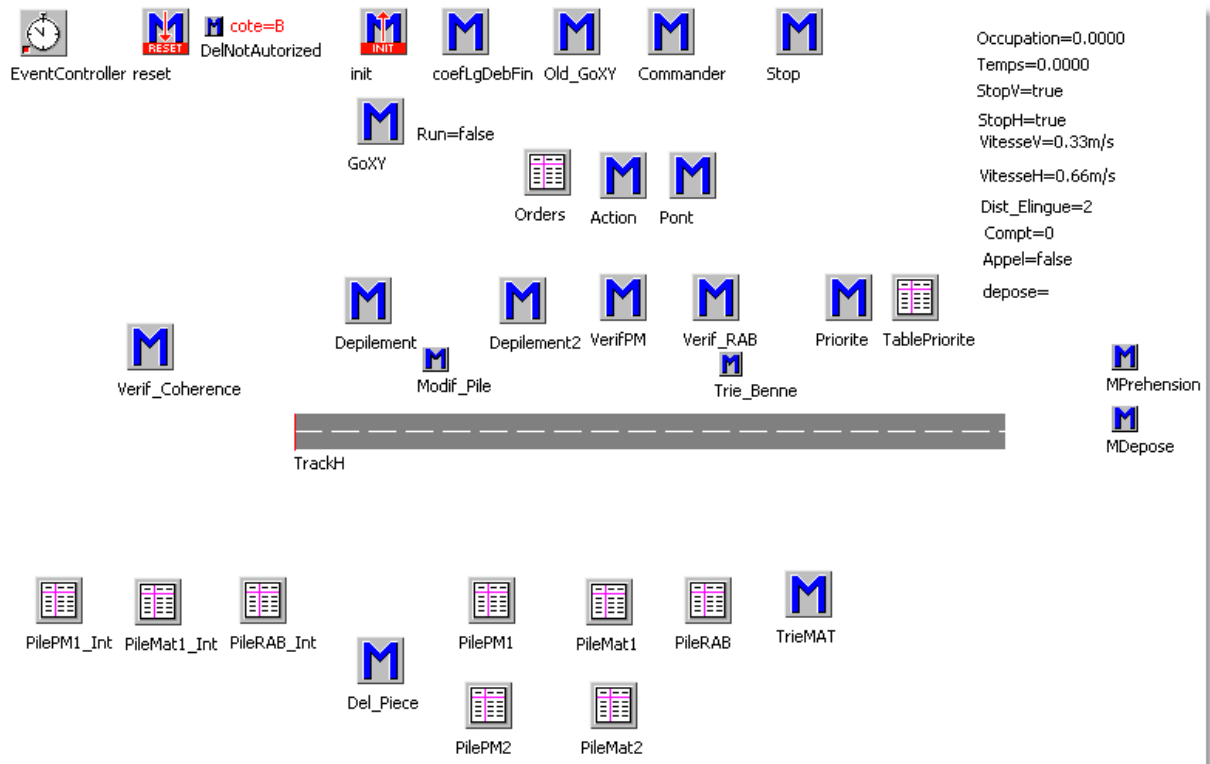
### 3.2.5.12 The “Pont” Frame

The frame is used to model the movements of the crane bridge in the workshop. We have one crane bridge by the side so two frames for the whole workshop. Orders for the crane bridge have different priorities. A management of these priorities has thus been implemented. Here is in the priority, in descending order, of the different operations that the crane bridge could carry out:

- Move away finished assemblies temporarily stored on the MAT area – because no PM was free;
- Bring basic plates and long stiffeners towards working areas – in this operation the eventual stacking of plates on the MAT area is enhanced. The transfer operation also has the same priority;
- Bring AS4 to the working areas;
- Move away finished assemblies
- Unload new TCP containers on the MAT area;
- Bring plates to be joined to the joining area or come to turn the plate over for the second welding.

To each order of the crane bridge will be linked one priority order (from 1 to 6, level 1 orders having greater priority).

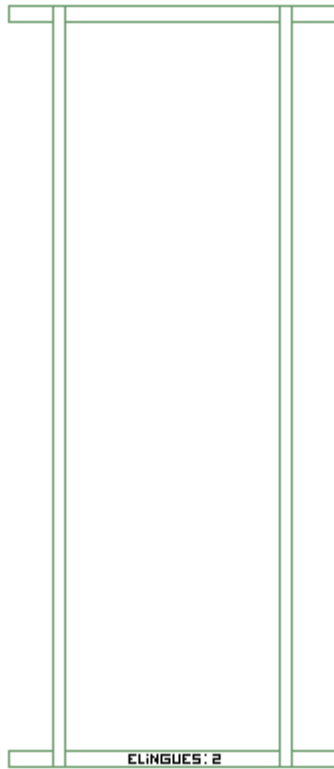
The structure of the frame is the following one:



**Fig. 71: The "Pont" Frame**

The frame contains two *Tracks* objects (*TrackV* and *TrackH*) to simulate the horizontal and vertical movements of the crane bridge. Animation lines in the root frame “Atelier” are linked to these *Tracks*. Consequently the movements of crane bridges can be seen in the root frame.

Notice that vertical moves can be done only at the extremity of the workshop – on the DCH, MAT and RAB areas. For security reasons for workers vertical moves can not be done above the cell. Consequently horizontal and vertical moves are thus completely separated. Both these movements are done by two different *Transporter* MUs represented by:



MU for the horizontal moves – name GP\_A and GP\_B in the simulation



MU for the vertical moves – named Port in the simulation.

Pieces to be handled are always put down into the *Port* MU.

During a horizontal move the *Port* MU is put down on the GP that is moving on *TrackH*. Notice that it is necessary to play on animation points to see the object *Port* at the right vertical coordinate during the horizontal move.

When we have a vertical displacement the *Port* object is put down on the *TrackV*, the object *GP* staying motionless.

Depending on the piece to be loaded the crane bridge will use either a magnet system or slings – six maximum. When we have to change the handling system, add or remove slings the icon of the crane bridge becomes orange. This is the set-up operation. Notice that if the piece needs two slings and four are set on the crane we do not remove slings. Set-up time is zero in that particular case. Slings are all removed together when we have to use the magnet system. In the simulation we can see the kind of tools used in the lower part of the icon. The number of slings (“Elingues” in French on the figure) also appears.



**Fig. 72: Crane Bridge Handling Systems**

*Source* and *TableFile* objects are used to create these two MUs at the start of the simulation. The table *Orders* is a very important one: all pieces to be handled with the crane bridge and their destination are saved in the table. Actually it contains eight columns:

- 1<sup>st</sup>: commentary on the order with generally the number of kit of the piece;

- 2<sup>nd</sup>: indicates if the order is a load or an unload order. Sometimes a special code is mentioned (as newPM → new PMs must be ordered; etc). In that case special orders are carried out such as modifying the value of a global value.
- 3<sup>rd</sup>: indicates the original position of the piece (A1, A2, ..., B1, ..., B4, DCH1, DCH2, DCH3, MAT, RAB, Default). The position by default of the crane bridge is at the extremity of the workshop above the top PM. That column also indicates the value that the variable should take in the case of a special order;
- 4<sup>th</sup>: indicates the name of the container where the piece to be taken is. If this is a special order this column is used to give the name of the variable to change;
- 5<sup>th</sup>: indicates the name of the piece;
- 6<sup>th</sup>: indicates the local coordinates (on the X direction) of its destination on the working area. If the order does not concern a move from or towards a working area this column is not used;
- 7<sup>th</sup>: indicates local coordinate (on the Y direction) of its destination on the working area. If the order does not concern a move from or towards a working area this column is not used;
- 8<sup>th</sup>: indicates the priority of the order (from 1 to 6).

*PilePMI* and *PileMATI* tables represent the state (i.e. the order of pieces) of stacks on entry PMs on the temporary storing area MAT when all the orders of the table Orders have been done – see method *GenAttribute* of the *DCH* frame. These objects are doubled in case we used two stacks in entry.

- The *Init* method

Initialises all tables and variables of the Frame.

- The *Reset* method

Deletes all MUs of the frame.

- The *GoXY* method

This method sends the crane bridge to the (X, Y) position. Coordinates are given as input data. As said before vertical moves are executed only outside of cells. Each move of the crane is saved. The total movement time is set into the *Occupation* variable. Initially we record the simulation time in the *Temps* variable.

At first we run the *CoefLgDebFin* method that will execute a coordinate transformation of the Y ordinate initially given in meters to change it to pixels. This value is necessary during a horizontal move to adjust an animation point of the *Port* on the *GP*.

On a *Track* object it is possible to add at any place sensors that trigger a method when a MU gets through it. *Tracks* objects on the frame thus represent the total horizontal and vertical lengths that the crane bridge can make. In the method we insert sensors at coordinates that will stop the move. The first move is always a horizontal move – to bring back the crane bridge to the zero abscissa if it is not yet there. After that we put down the *Port* object on the *TrackV* object to make the vertical move. Then we again put the *Port* object on the *GP* – with a change into the animation point – and make a horizontal move on the *TrackH*. For each move we use Boolean variables *StopH* and *StopV*. Their initial value is set on False. The *Stop* method is launched with the trigger of sensors and this method stops the crane bridge movements and modifies variable values to true. The *GoXY* method can thus continue to do the next displacement.

At the end of a move we deduce the simulation time saved in the *Temps* variable and add it to the value of the *Occupation* variable. Nevertheless this operation is done only if the *Stat* variable (*Stat\_A* or *Stat\_B* depending on the crane bridge side) of the *Tables* frame is true. This means that we have already started to record statistics. During the “warm-up period” we do not want to record statistics to avoid influencing results.

- The *CoefLgDebFin* method

This method changes vertical coordinates (given in meters) into pixels. This value is necessary to adjust animation points between *Port* and *GP* objects.

- The *Stop* method

The method is run when a MU gets on a sensor situated either on the *TrackH* object or the *TrackV*. The speed of the MU is set to zero. The value of *StopH* (or *StopV*) is also set to True.

- The *Action* method

This is the most important method of the *Pont* frame. The method executes all the orders contained in the *Orders* Table. The first row of the table is the next to be executed. When the order is carried out we delete the first row and shift all the rows of the table. The crane bridge is thus always carrying out the order of the first row of the table. Four kinds of orders can be

done by the crane bridge: loading order, unloading order, special command (changing the value of a global variable of a frame) and orders for new PMs on a DCH area.

- load a piece:

We send the crane bridge above the piece with the *Pont* method. We look if it is necessary to proceed to a change of handling equipment. The crane bridge has the attribute Slings – “*Elingues*” in French, a variable that indicates the number of slings mounted on the crane bridge. A zero number means that the magnet system is used. If the piece should be handled by magnets and the number of slings is not zero we have to model a set-up time. We change the colour of the crane bridge – that becomes orange – and force the crane bridge to wait for the set-up. The waiting time is calculated in the *Times* method of the *Tables* frame. Thus we change the value of the *Elingue* attribute and modify the icon of the crane bridge. If the piece needs to be brought with slings we calculate the adequate number necessary. It is directly linked to the length of the piece. Generally the distance between two slings is two meters. If the number of slings to be used is greater than the number currently in place we do a set-up time proportional to the number of slings to add – again with the *Times* method.

We move the piece on the crane bridge. Notice that if the taken piece comes from the MAT area we have to run the method *TrieMat1*. This last method will sort assemblies in this area – see section *TrieMat1* method for more details.

- unload a piece:

The first possibility is to bring a piece to the working area. In that case the mechanized gripper or a worker is maybe tacking another piece. The crane bridge has thus to wait for them to finish their piece before putting down the new one. The crane bridge is then waiting just at the entry of the working area. When the tacking of the piece is finished, the crane bridge can come to the right position. The variable named “*Depose*” (put down) of this frame is then modified in “*encours*” – meaning that the unloading operation is currently being carried out. The crane bridge icon is not changed. It will be changed in the *DetParam* method of the *PosteAss* frame.

The second possibility is to bring a piece from DCH to MAT. If this is an assembly we run the *VerifPM* method that checks the destination: an adequate PM – with a good *Sub-panel* – and its maximal capacity cannot be reached. Then we move the crane bridge to the unloading location (with the *Pont* method), modify its icon to purple and finally simulate unloading time (*Times* method) to give back its initial icon.

Finally in both cases we can put down the piece. As previously, if the piece is an assembly to put in MAT we run the *TrieMat1* method. When the *Depose* variable has found its initial state the unloading operation is finished and the crane bridge is free to execute a new order.

After these operations we have to check now if PMs are not full and thus been moved away. If *Etats* attributes of *SortiePM1* and *SortiePM2* objects of *DCH* frame have the value “*plein*” (full) then we unlock the entry of the *Sortie* object that will manage the moving away operation. The *PanInd* table of the *Tables* frame contains the main information of all the sub-panels encountered in the simulation. One column indicates the number of assemblies by sub-panel to be moved away. If this number is zero it means that we just have moved away the last assembly of this sub-panel. In that case we save in the table its exit hour and its total duration in the workshop. We also check if all sub-panels studied at this workshop side have not been moved away. If this is the case, we save the end time in variables *Fin\_Stat\_A* or *Fin\_Stat\_B* – depending on the side. These variables are located in the *Tables* frame. If both workshops have finished all their sub-panels the simulation is stopped. The method *CreateResult* of the *Tables* frame is run to update statistics, results and some graphics and charts.

- Modify a variable value:

The label of this operation is “command”. When we have put down some pieces it is sometimes necessary to report to another frame that a special order has just been moved away. We have to modify the value of some variables: we modify the value of the variable indicated in the fourth column of the *Orders* table by giving it the value indicated in the third column.

- Order new PMs:

The label of this operation is “NewPM”. Here again we will modify the value of a variable but the goal is to order new PMs in the *DCH* area.

- The *Pont* method

The method calculates destination global coordinates of the crane bridge. In input the method receives the global destination (*MAT*, *DCH*, etc.) and local coordinates if the destination is a working area. When calculation is done (simple coordinates addition) the *GoXY* method is run in order to really send the crane bridge to the calculated coordinates.

- The *Depilement* method

The method is run when we ask the crane bridge to bring a basic plate to the working area – for instance by the *ToleBaseGP* method. The piece is not obligatory at the top of the stack. Consequently we have to stack plates that are above the wanted plate. The method automatically introduces in the *Orders* table all the information that has to be executed to bring our plate on the working area.

First step: we look if the piece is not already stacked in the MAT area. In that case we give the order to bring back all the pieces above on the PM. Then we can take our piece and bring it to the working area.

Second step: then we look if the wanted piece is not in the PM stack. Again we stack upper pieces.

At the end of the method we run again the *Action* method if new orders have been added during the execution of the method.

- The *VerifPM* method

That method checks if the assembly final destination – in that case a PM – is not occupied: we need to move away PMs when they reach a specific weight – or a number of pieces or a specific surface area – or when all assemblies of the same sub-panel are on the PM. Currently the PM limit is a number of pieces but that limitation can be changed easily.

In entry we have the name of the piece (or more exactly the assembly). We look into the *PanInd* table if the sub-panel of the assembly is already assigned to one (or two) PMs. If this is the case and if the PM is not full we introduce in the *Orders* tables the coordinates of this PM. If the sub-panel was not allocated we look to see if some PMs are free. This is the case when *Etat* attribute of *SortiePM1* and *SortiePM2* objects is void.

If the first one is empty we allocate this PM to the corresponding sub-panel and indicate in the *PanInd* table in modifying the *Etat* attribute of *SortiePM1* in “Occupe”. Dimension of the longest assembly of this Sub-panel is saved in the table *PanInd*. If this value is greater than a specified value (by default eight meters) we suppose that two PMs will be allocated for this sub-panel. In the simulation we then fix the *Etat* attribute of *SortiePM2* to the value “Plein”. No assembly will be put down on this PM.

If the first one is not empty we then look at the second PM. To allocate it to the assembly the Sub-panel should not be too long – as previously we look at the longest of its



assemblies – and of course the PM must be free. If both these conditions are satisfied we can allocate the PM.

In this way we check PMs of the two *DCH* areas – *DCH\_1* and *DCH\_2*. Now after having placed the assembly the PM can be full. We check if this is the case and eventually change the corresponding *Etat* variable into “*Plein*”. We do not forget that during the taking away we each time move away both PMs so the moving away order will be given only when both PMs are full. There is an exception to this rule: if the second one is empty we can move away PMs straightaway.

When we just have established that a PM is full we must delete eventual orders to bring assemblies from MAT to the PM! We thus delete corresponding rows into the *Orders* table.

- The *Priorite* method

The method *Priorite* (priority) gives back an integer that indicates the number of the row in the table *Orders* where the order must be recorded – depending on the priority order specified by the user. Each time we give an order to the crane bridge we should introduce some information (name of the piece, destination, etc.) that we save in the *Orders* table. These orders do not have the same priority and thus should be introduced at the right place in the table. This place is given by this method. The method is thus run each time we want to insert a new order into the table.

We have six priority orders given in the introduction of this frame. The method parses thus the *Orders* table to determine where put the order and inserts empty rows at the right place.

- The *TrieMat1* method

This method is run when we add or remove an assembly from the MAT area. Assemblies of this area are put onto a virtual stack and we want them to be classified by their chronological order of arrival in order to move them away with the same order. An internal table – automatically generated by eM-Plant – is linked to the container and contains information of all the pieces of the container. When we take a piece from the container – or put down a new one – this table is modified automatically without clear logic. This means that the first row does not necessarily contain the first piece. This is one of the software’s known bugs. Our method *TrieMat1* allows us to avoid this problem in reclassifying correctly all the pieces of the container.

### 3.2.5.13 The “Prehenseur” Frame

In each *Ilot\_Demi* frame we can find a *Prehenseur* frame. This frame manages the movements of the mechanized gripper. It always moves horizontally along a half cell but it contains a sliding arm allowing the worker to move pieces vertically on the width of cells.

The structure of the frame is the following one:

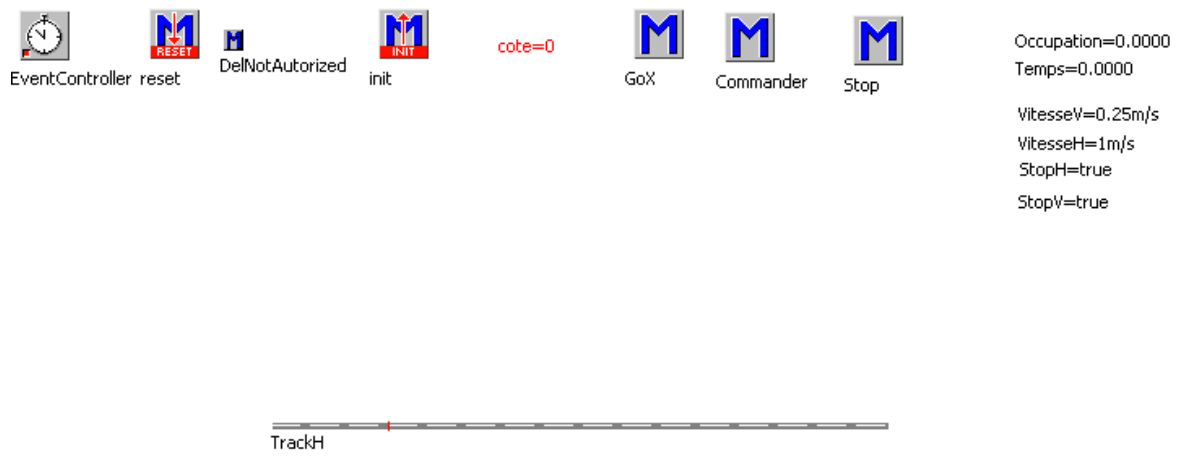


Fig. 73: The “Prehenseur” Frame

The frame contains very few objects: mainly one track (called *TrackH*) on which the mechanized gripper moves. As for the crane bridge the mechanized gripper is constituted of two different transporter MUs:

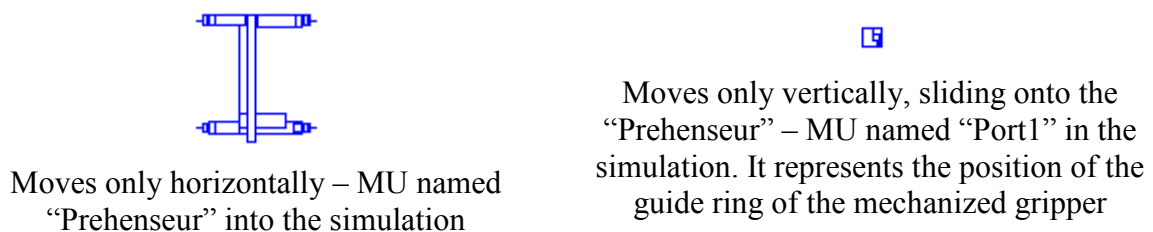


Fig. 74: Representation of the Mechanized Gripper

*Remark: four different icons are in the simulation, one for each cell.*

Contrary to the crane bridge there is no vertical track for the horizontal moves of Port1. Actually it is possible for any Transporter MU to be considered as a “track” on which another Transporter MU can move. It will be the case for this object. The Port1 will move directly on the Prehenseur MU. The advantage is that horizontal and vertical moves can be executed simultaneously. We did not need to have this characteristic for the crane bridge because, as mentioned before, vertical and horizontal moves must be completely separated – for the security of workers in the workshop.

Notice also Occupation, Temps, StopH and StopV variables that have exactly the same function as in the crane bridge frame.

- The *Init* method

The method creates the *Prehenseur* MU on the object *TrackH* and the *Port* MU. The method also resets variables of the frame and sends the mechanized gripper to the STK area.

- The *Reset* method

The method deletes all MUs of the frame.

- The *GoX* method

The method sends the mechanized gripper to X and Y coordinates given in entry of the method. X corresponds to horizontal moves and Y to vertical moves. The main principle is identical to the *GoXY* method of the crane bridge. We will use sensors located on tracks to trigger *Stop* method that will stop the movements of the mechanized gripper. Notice that there are some differences because horizontal and vertical moves can be executed simultaneously.

- The *Stop* method

The method is run when a MU gets over a sensor situated either on the *TrackH* or on the fictive *Track* on the *Prehenseur* MU. The method thus stops the MU in cancelling its speed. Value of *StopH* (or *StopV*) is also changed to *True*.

#### **3.2.5.14 The “Robot” Frame**

We have a *Robot* frame in each *Ilot* frame. The frame is used to model the movements of the welding robot and also to simulate welding time.

The structure of the frame is the following one:

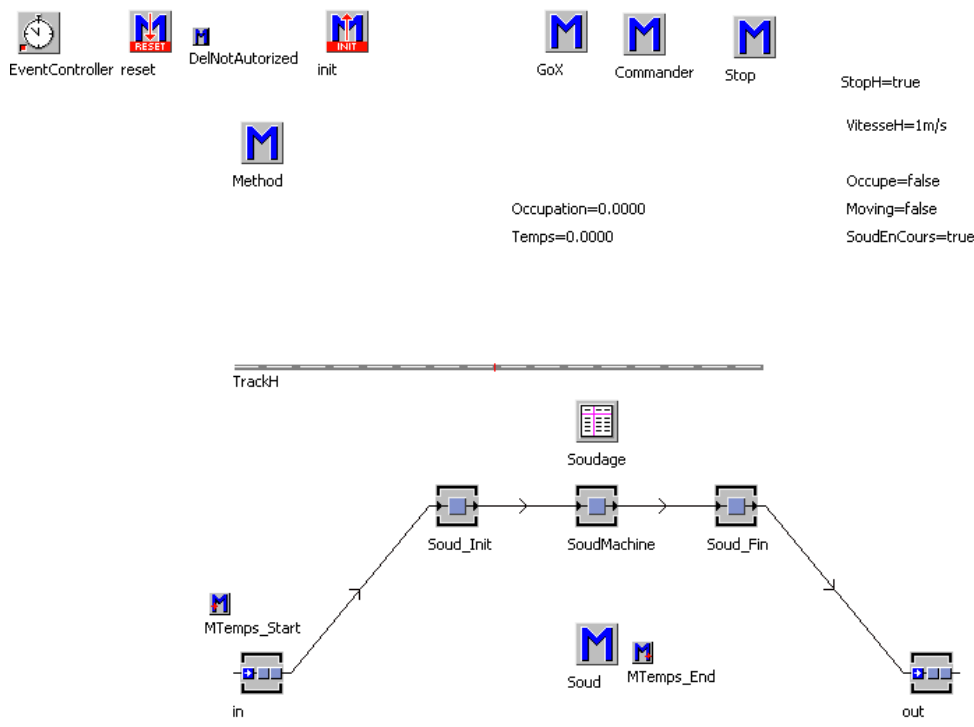


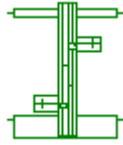
Fig. 75: The "Robot" Frame

The movements of the welding robot are easier than the movements of the crane bridge or the mechanized grippers because we have here only horizontal moves. Indeed we model only movements between assemblies. We do not model detailed welding moves. The first reason is that we do not have these data and the second one is maybe more important: we gain nothing in the simulation in having such details. If we have in entry the total welding time for each assembly we have no interest in trying to model detailed moves. For these reasons we only have a *Track* object and one *StopH* variable.

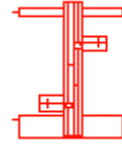
Nevertheless we have an object named *SoudMachine* used to model the welding time of pieces. This machine is not in the *PosteAss* frame, as it is for the tacking or welding operation. The advantage is that we do not lose statistics relative to the welding operation because everything is recorded in the *SoudMachine* object. If we had put this frame in *PosteAss* these statistics would have been lost because the *PosteAss* object is deleted when the assembly is finished.

When the welding operation starts we put the assembly MU from the *PosteAss* frame on the object named "In" that will trigger the *Soud* method. The *SoudMachine* object will model the welding time. This time has been calculated previously in the *Soud* method of the *Ilot* frame. After the welding the assembly goes to the *Out* object that triggers again the *Soud* method of this frame.

The welding robot – named *Robot* in the simulation – is modelled in the simulation by:



In rest or in motion



During the welding operation

**Fig. 76: Representation of the welding robot**

*Remark: four different icons are used one for each cell.*

- The *Init* method

The method creates the *Robot* MU on the *TrackH* and sends it to its initial position at the middle of the cell.

- The *Reset* method

The method deletes all MUs of the frame.

- The *GoX* method

The method sends the welding robot to the input coordinate. This is a simplified method in comparison with the *GoXY* and *GoX* methods previously described. See these methods for further details.

- The *Stop* method

The method is run when the welding robot get through a sensor located on the *TrackH* object. The MU is then directly stopped. The value of the *StopH* variable becomes true.

- The *Soud* method

The method modifies the value of the *SoudEnCours* variable to *True* just before simulating the welding time and changes it to *False* when the welding is finished. This variable is used in the homonym method *Soud* of the *Ilot* frame.

### 3.2.5.15 The *Interface* Frame

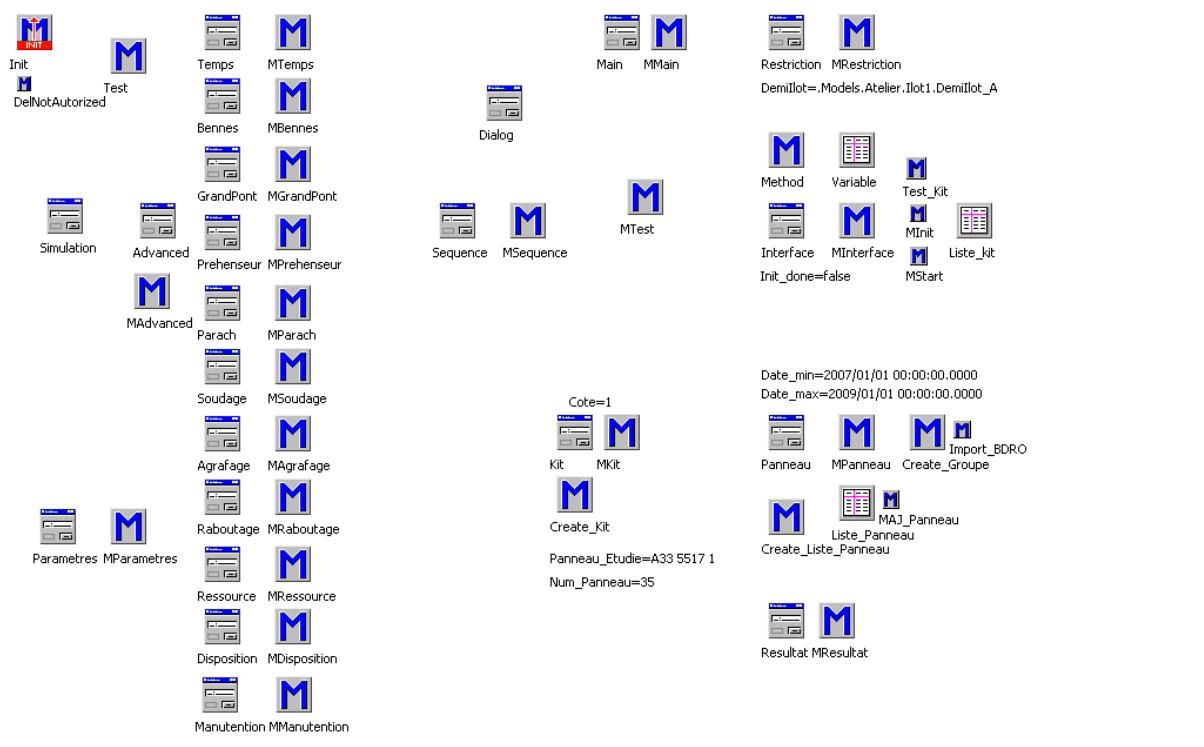
The simulation has numerous parameters. We have two different kinds of parameters:

- Those relative to the basic functioning of the simulation. The most important goal is to have a simulation model that represents reality as well as possible. To have this correspondence basic parameters must be adjusted – for instance the speed of the crane bridge or mean tacking time by meter;

- Those relative to the products that we want to simulate. Once the model has been developed we will try different lists of assemblies. For instance an important parameter is the order of construction.

For convenient reasons it is more useful to have access to all of these parameters in one location. It is also important to have a user-friendly interface because due to this huge number of parameters the user could be quickly overwhelmed. This frame manages all the dialog boxes to make the software more user-friendly. Consequently the tool can be used by someone who does not have knowledge of the software.

The structure of the Interface frame is the following one:

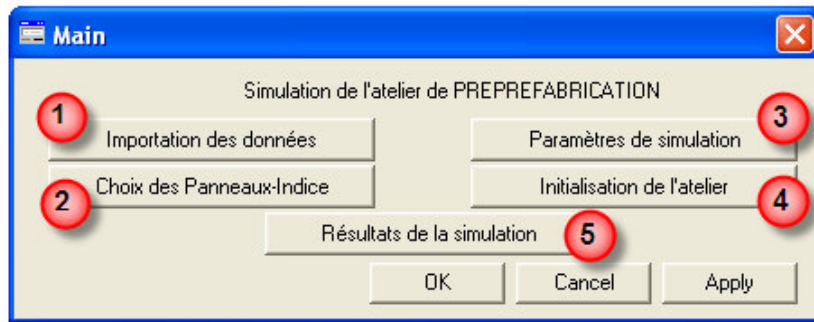


**Fig. 77: The "Interface" Frame**

In the simulation the icon of the Interface frame has been changed and is represented by:



Contrary to all other frames of the model, if we click on the frame we do not open the child frame – same structure as in Fig. 77 – but we open a dialog box as in Fig. 78.



**Fig. 78: The Main dialog box**

- ① : Button to import data from the Database Access;
- ② : Button to select sub-panel to simulate in the model;
- ③ : Button to access the main parameters of the simulation;
- ④ : Button to initialise the workshop;
- ⑤ : Button to access the results of the simulation.

Each frame has in fact an attribute *OpenCtrl*. In this attribute we can specify a method path. When we open the object – in this case by clicking on it – the corresponding method is run. The attribute of this frame has been linked to the method *Main* of the frame. The method opens the dialog box of Fig. 78.

In the following sections we will describe in detail this dialog box.

- The Data importation button

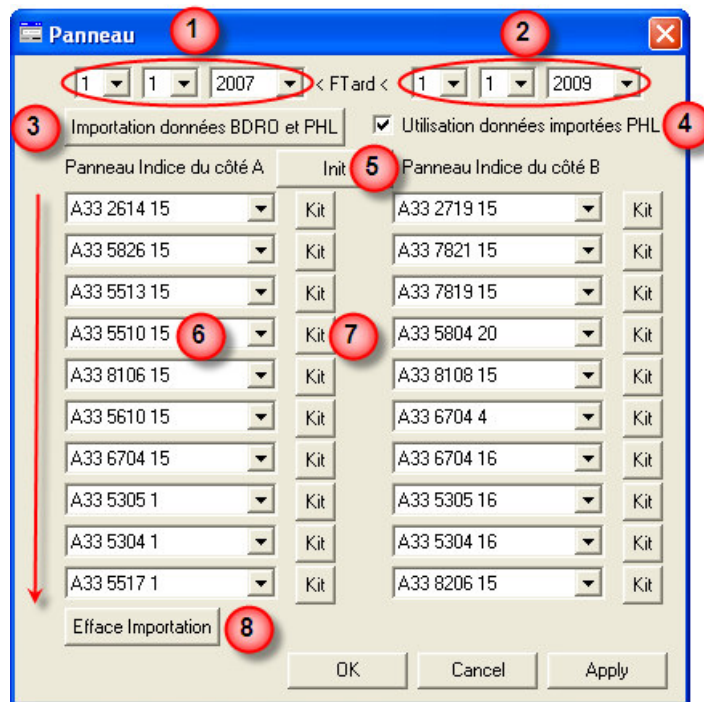
This button does not open another dialog box but straightaway imports the Access Database. Button runs the method *ImportAccess* of the *Tables* frame. Geometrical information of pieces is thus available.

The button also imports information from the BDRO: the list of all sub-panels with their latest starting date. If a sub-panel does not have a latest starting date it will not be taken into account in the simulation.

It also checks if there is no incoherence in the database Access – for example if there is at least one basic plate for each assembly, etc.

- The Sub-panel selection button

Clicking on this button launches the method *MPanneau*. This method runs and manages the dialog box of Fig. 79.



**Fig. 79: The sub-panel selection button**

- ① : List boxes to select the earliest date of the sub-panel;
- ② : List boxes to select the latest date of the sub-panel;
- ③ : Import BDRO and PHL information;
- ④ : Checkbox to decide if we use data from the PHL or not;
- ⑤ : Initialise each element to have a sequence of identical sub-panels at the BDRO;
- ⑥ : Allow the user to select sub-panels and their order;
- ⑦ : Open a new dialog box to select kits of the corresponding sub-panel;
- ⑧ : Delete the importation.

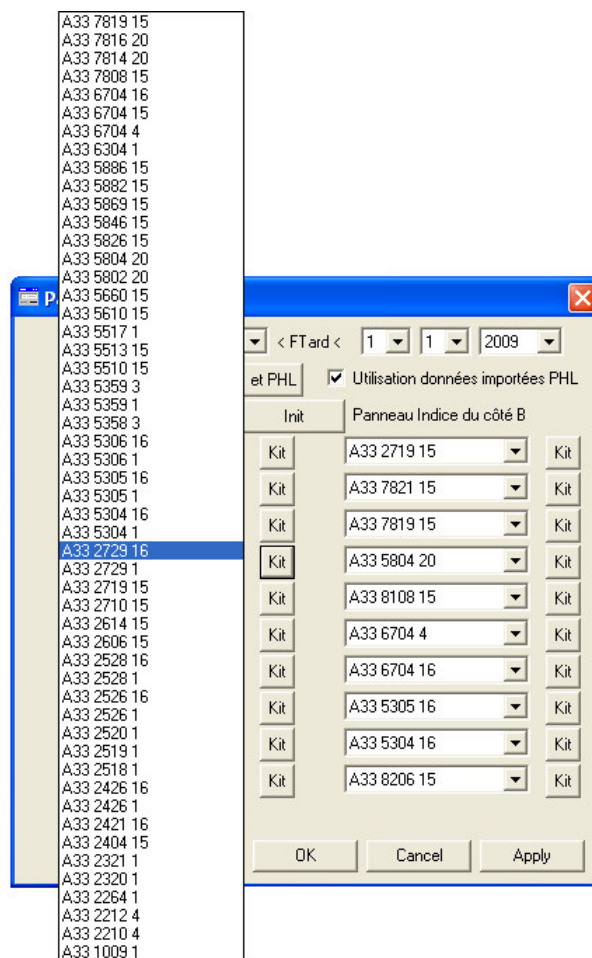
This dialog box is very important in selecting the sub-panel to be simulated. First of all we have to select sub-panels that have a “Latest date” greater than a certain date – defined with list boxes 1 – and inferior to a maximal date – defined with list boxes 2. Button 3 imports data from the BDRO and from the PHL. Only sub-panels with correct dates are imported. Checkbox 4 allows the user to take into account (or not) data from the PHL.

On the interface we can see two columns with the possibility of choosing a sub-panel. The first column corresponds to the left side of the workshop and the second one to the right side. The upper sub-panel will be the first one to be made in the workshop. When we click on the Init button (5) available sub-panels are automatically classed by their latest starting date – coming from the BDRO – and then put into each list box (6). Even if there are only 10 list boxes we can simulate an infinite number. Only the first ten are accessible via this interface.



We have to access other tables directly if we want to manually change the sequence. In practice it is not really interesting to do the operation manually. Indeed there is no reason to build in the workshop sub-panels in a different order than those given by their latest starting date. The first sub-panels are important because in order to create an initial situation in the workshop, we will use kits of these sub-panels. Consequently we must have the possibility of modifying manually the first, second or third – or even more – sub-panel of the workshop. That is the reason for this interface and why we have limited the dialog box to ten entries. Anyway every change can be done without using the interface.

In each list box we have a list with all sub-panels available – see Fig. 80.



**Fig. 80: List of sub-panels available**

That list is only available after the importation from the BDRO. Next to each selected sub-panel we have a button named Kit (7). This button allows the user to have access to the list of kits that belong to the sub-panel – see Fig. 81.

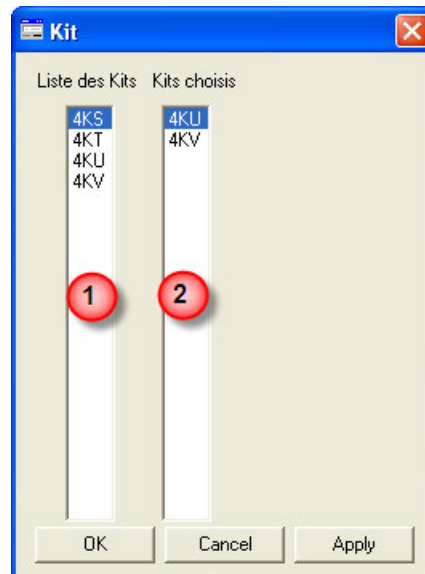


Fig. 81: Selection of kits to be simulated for each sub-panel

- ① : List of kits of the selected sub-panel;
- ② : List of selected kits for the simulation.

It is important to have the possibility to select only some kits by sub-panel. This is generally the case for the sub-panel that will be used for the initialisation. Some of its kits have maybe already been done and thus do not have to be simulated.

Button 8 of Fig. 79 is used to delete information of the importation.

- The parameters button

This button gives access to different parameters of the simulation. Button opens the dialog box of Fig. 82.

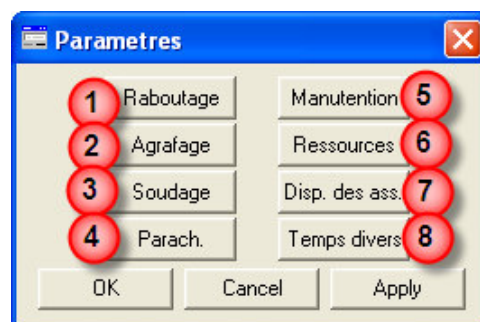
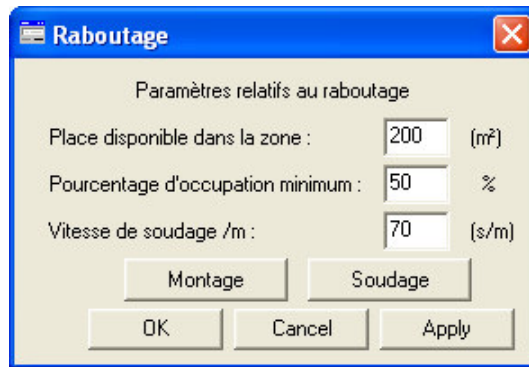


Fig. 82: Parameters dialog box

- ① : Open parameters related to the joining area;
- ② : Open parameters related to the tacking operation;
- ③ : Open parameters related to the welding operation;
- ④ : Open parameters related to the finishing operation;

- 5 : Open parameters related to handling operations – crane bridge, mechanized gripper;
- 6 : Open parameters related to Resources – workers;
- 7 : Open parameters related to allocation of assemblies in the working area;
- 8 : Open parameters related to different simulation times.

Button 1 opens the following dialog box:



**Fig. 83: Dialog box related to joining areas**

The first box is used to define the available space in a working area in square meters. There is not really a management of the space allocation in the simulation for these areas. We just have this parameter to limit the number of plates in the area.

The second box is a percentage of occupation of the area – surface occupied divided by the previous parameter. When the percentage is lower than the specified percentage of this box we call new PM containers. This parameter is important because this factor influences a very important aspect of the workshop: arrival of PMs. Due to the fact that most of the basic plates arrived on PM we see that this parameter is important.

The third parameter is the welding speed (by meter) of the workers. The unit is second by meter.

The left button (“Montage”) directly opens a table that contains the times needed to carry out some operations in the joining area.

The right button (“Soudage”) also opens a table with times but they are related to the welding (always only for the joining area, not for the *true* welding operation on working areas).

Button 2 (“Agrafage”) of Fig. 82 opens a new dialog box with parameters related to the tacking operation:

VITESSE_AGRAFAGE	Valeur	Unités
Petit UPS	84	s/m
Benne Tole	72	s/m
Palanqué	207	s/m
Mini PRS	230	s/m
Grand UPC	72	s/m
Petit UPC	72	s/m
Grand PRS	61	s/m
Grand UPS	61	s/m
AS4	230	s/m

**Fig. 84: Parameters relative to the Tacking operation**

For each kind of piece (UPS, UPC, etc.) we have a different tacking time due to the complexity of the piece. Indeed it takes more time to tack an AS4 (which is a small assembly) than a simple girder. The dialog box contains these times for each piece in seconds by meter.

Button 3 opens a dialog box about parameters related to the welding. The dialog box contains parameters explained in section 3.2.3.7. Nevertheless the total welding time is calculated in the database treatment (in Access) and once the importation has been done these parameters cannot be modified. The parameters can thus be seen in the dialog box but cannot be changed.

Button 4 opens a dialog box related to the finishing operation – see Fig. 85.

	Valeur
RATIO PARACHEVEMENT	0.8
RATIO 1 OP PARACH	1
RATIO 2 OP PARACH	0.75
RATIO 3 OP PARACH	0.3

**Fig. 85: Parameters related to the finishing operations**

The finishing operation is the most complex to simulate in this workshop. This is due to the difficulty of foreseeing the work to be done on each assembly. The general idea is: the more time it takes for the welding the more time it will take for the finishing. This is not

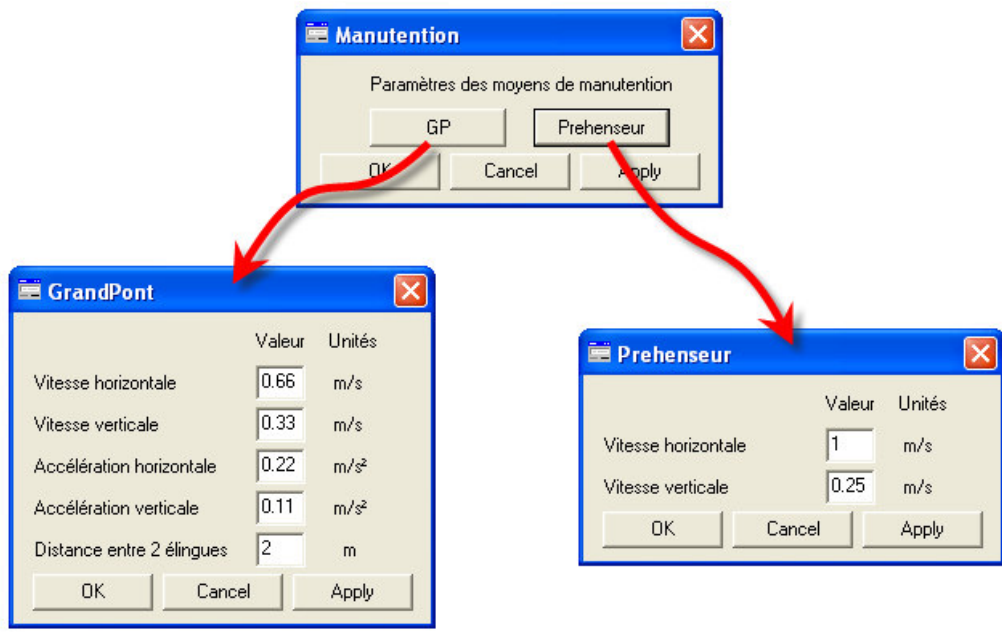
always true but this rule corresponds quite well to reality. Consequently we have decided to estimate the finishing time by assembly as a ratio of its welding time. This ratio is entered in the first square of the dialog box of Fig. 85. By default the value is set at 80%.

This finishing time depends naturally of the number of workers working on the assembly. The time will lessen when the number rises. Of course this number is not directly proportional: if we have two workers the finishing time is not necessary divided by two. The three rations of the figure correspond to the effect on the time of different numbers of workers (one, two and three). For instance if we have two workers the finishing time of an assembly will be:

$$\text{WELDING TIME} \times \text{RATIO FINISHING (FIRST SQUARE)} \times \text{RATIO 2 WORKERS (THIRD SQUARE)}$$

By convention we cannot have more than three workers at the same time on an assembly. This is the case in reality because assemblies are not so long so we have to avoid workers getting in each other's way.

Button 5 opens the dialog that contains two buttons. The first one opens a new dialog box related to the crane bridge and the second opens one related to the mechanized gripper – see Fig. 86.



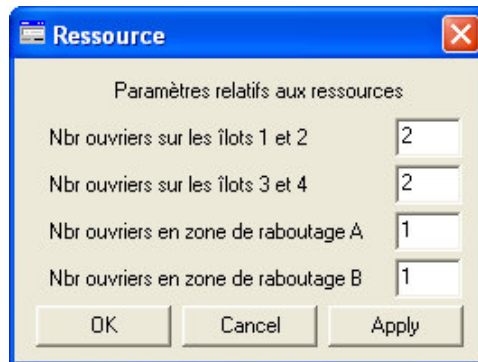
**Fig. 86: Handling parameters**

Available parameters for the crane bridge are: horizontal speed, vertical speed, horizontal acceleration, vertical acceleration and distance between two slings. Notice that accelerations are not really important compared to other approximations that remain in the

workshop. The distance between two slings is a parameter that influences the set-up time of the crane bridge.

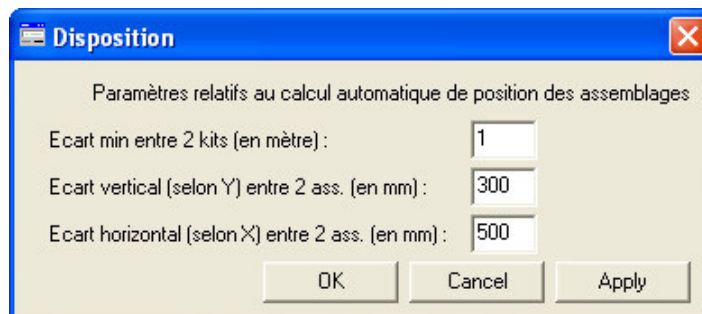
For the mechanized gripper we have only horizontal and vertical speeds.

Button 6 opens a dialog box related to the number of workers in each WorkerPool – see Resource frame for more details:



**Fig. 87: WorkerPools parameters**

Button 7 opens a dialog box related to the layout of assemblies on working areas – see Fig. 88.



**Fig. 88: Parameters of assemblies' layout on working areas**

The position of assemblies on the working area is not necessary given in the database. In that case we have to automatically generate their positions in order to minimize the total obstruction of the kit. For security and technological reasons assemblies must be separated by a certain distance. Boxes 2 and 3 give the minimum distance to be respected between assemblies – on the width direction and on the length direction. Kits must also be separated. The security distance is greater because it is possible to have tacking on a kit when the automated robot is welding on the next kit. The minimal distance between two kits (in meters) is given in the first box of the dialog box.

The last button of Fig. 82 (number 8) opens a dialog box related to different simulation times:

Paramètres relatifs au Prehenseur						
Description	P	x1	x2	m	sig	Distribution
Temps de Prehension	1	14	60	30	0	triangle
Temps de dépose	1	30	60	40	0	triangle
Temps de Set-up	0.95	30	90	60	15.3	normal

Paramètres relatifs aux ouvriers						
Description	P	x1	x2	m	sig	Distribution
Temps de réglage /mètre	1	14	60	30	0	triangle

**Fig. 89: Times and their distribution used in the simulation**

There are five different tabs with many parameters related to a simulation time: containers tab, preparation tab, tacking tab, finishing tab and crane bridge tab. On Fig. 89 we can see the tacking parameters time. As explained earlier any time used in the simulation can be linked to a distribution. Even if the kind of distribution can be really varied in practice we will only use Normal distribution and Triangular distribution.

For a triangular distribution we only need three parameters: the upper bound, the lower bound and the mean. For a normal distribution we need to know: the means and the variance. Practically it is much more convenient to estimate the distribution with a mean, a lower bound, an upper bound and a probability to have a time included in between these bounds. With these parameters and abacus it is possible to calculate the corresponding variance.

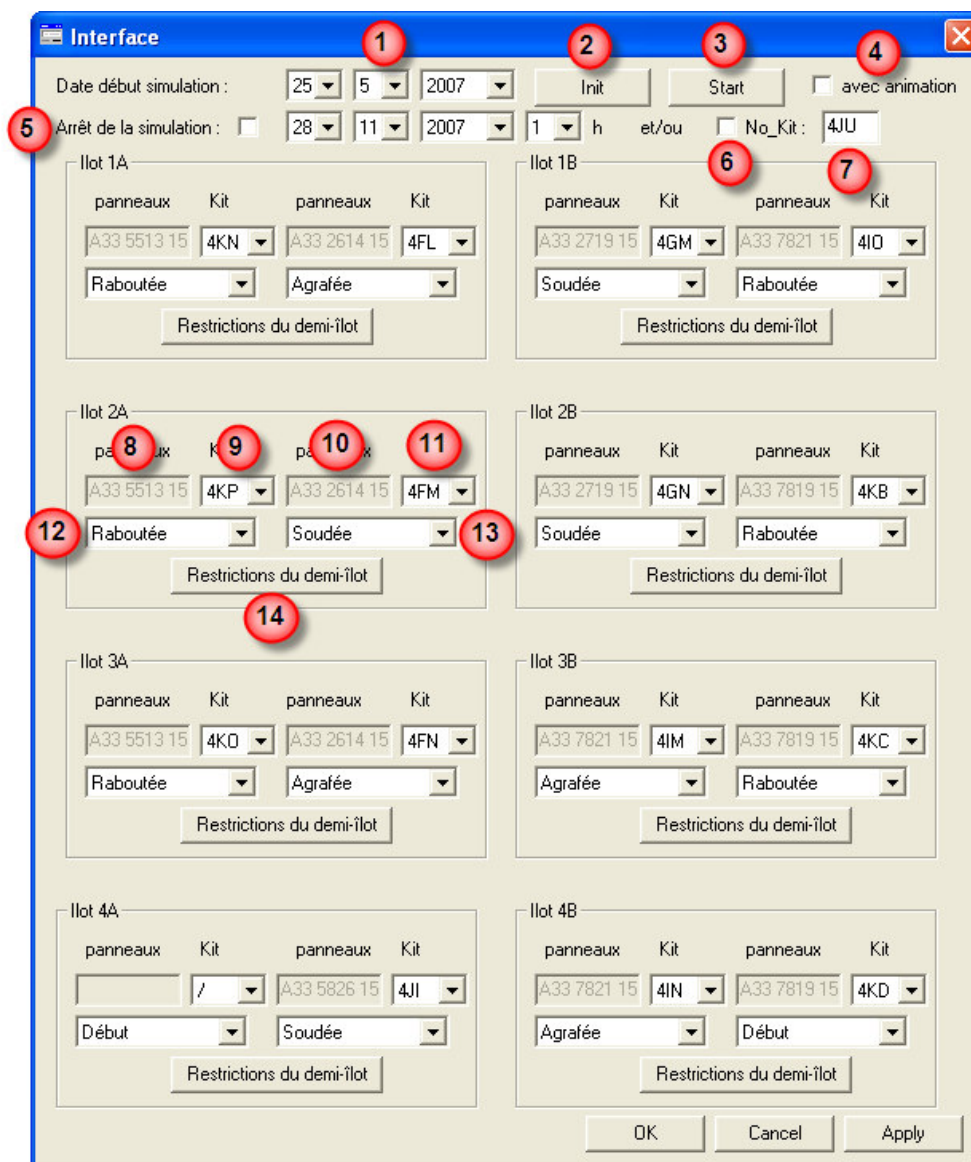
Depending on the chosen distribution in the last column we have to fill a different number of parameters in the previous column.

For the first tab we have times related to containers supplies and taking away. The second tab is related to the preparation time of the working area before to receive a new kit. The third tab is related to the tacking operation – time to take a piece, to put down, for the set-up, adjustment time. The fourth tab is related to the finishing time. The fifth tab is related to

the crane bridge – time to take a piece (with slings and with magnets), to put down, for the set-up, etc.

- The initialization button

This button opens a very important new dialog box – see Fig. 90. Here we can select the configuration of the workshop before the simulation start. We can select kits and put them on any working area at any production level – tacked, welded, etc.



**Fig. 90: Interface of initialization of the Workshop**

- ① : We can specify here the start date of the simulation. We can also define a date when the simulation must stop even if all kits are not yet simulated;
- ② : Executes the initialization of the workshop;
- ③ : Start the simulation process;



- ④ : Checkbox to see the simulation or not – the simulation is faster if there is no visualisation;
- ⑤ : Checkbox to stop (or not) the simulation at the chosen date;
- ⑥ : Checkbox to stop (or not) the simulation when the specified kit (in 7) will start to be built;
- ⑦ : Identification number of the kit that will trigger the end of the simulation – only if checkbox 6 is selected;
- ⑧ : Sub-panel of the kit that will be initialized directly on a *external* working area;
- ⑨ : Identification number of the kit that will be initialized directly on a *external* working area;
- ⑩ : Sub-panel of the kit that will be initialized directly on a *internal* working area;
- ⑪ : Identification number of the kit that will be initialized directly on a *external* working area;
- ⑫ : Selection of the state of the kit that will be initialized directly on a *internal* working area;
- ⑬ : Selection of the state of the kit that will be initialized directly on a *external* working area;
- ⑭ : Button to add constraints to the corresponding half-cell.

We have 4 cells, 8 half-cells and 16 true working areas. To avoid starting with a completely empty workshop – that is not realistic – we can, for each working area, define its situation: which kit is already there (maybe none) and at which level of production – joining done or tacking done or welding done or finishing done.

Button 14 needs more information. When we click on it we open a new dialog box – see Fig. 91.



**Fig. 91: Constraint for kits on a half-cell**

The first checkbox is used to activate or deactivate the half-cell. We can suppose for instance that one half-cell is completely blocked – failure, or restrictive functioning mode of the workshop, etc. The second line is used to authorize (or not) for the half-cell only the kits that have a welding time included in an interval specified by the user. That constraint allows the user to test different alternatives to see the effect on the total production time. For instance

we can try to book some cells for kits with a lot of welding time to see if the results are better than if we have a mix of bigger and smaller kits.

New constraints can easily be added to this dialog box – constraint on tacking time, finishing time or a completely different criterion.

- The result button

This button allows the user to have access to a new dialog box – see Fig. 92.



**Fig. 92: Access to results**

The first button gives an access to a table that contains the times of each operation for each kit. It is a very important table to see very detailed results. Waiting times are also highlighted.

The second button creates an HTML report of the simulation results. The report contains graphics, tables, variables, etc. selected by the user. It is much more convenient to have all the results in the same report than having to open each chart, each table, and each variable to understand the simulation results.

The description of the structure of the *Interface* frame – see Fig. 77 – will not be detailed for each object. The frame is mainly composed of pairs of objects: a *method* is always linked to a *Dialog* object. The Dialog object contains different tabs, checkboxes, list boxes, etc defined as we could see in the previous section. The associated method supplies the behaviour to be effected when we click on any of these objects/buttons. For instance it calculates list of available sub-panel when we click on a list box of the dialog box of Fig. 79.

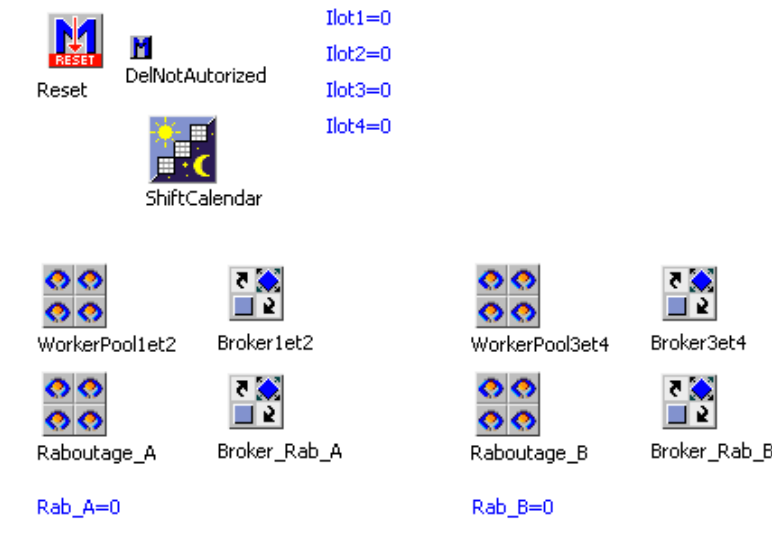
### 3.2.5.16 The Resource Frame

This frame is really simple and is used to manage workers into the workshop. The frame is directly in the root frame and its icon has been changed to:



Note that the chosen icon is the same as for the WorkerPool object.

The structure of the frame is given in the following figure:

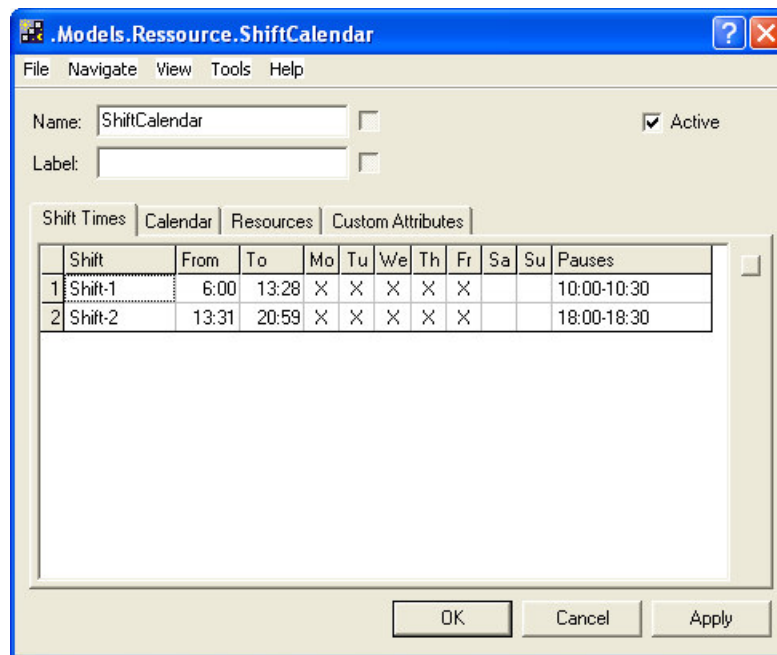


**Fig. 93: The "Resource" Frame**

We have four *WorkerPool* objects with an associated *Broker* object for each one. This frame is directly organized like the workers organisation in the real workshop. We have one pool of workers for each joining area (*RAB*), one pool for the two upper cells and one for the two lower cells. It means that if we need a worker in the half-cell A1, then we can call only the worker of the corresponding pool. If a worker is free in a joining area he cannot go to this half-cell! The number of workers available on each pool is a parameter that can be changed by the user.

Four variables are also created on the frame: *Ilot1*, *Ilot2*, *Ilot3* and *Ilot4*. Each variable indicates the number of workers that are tacking on the corresponding cell. The value of the variable is used in other frames. There are also similar variables for the joining areas: *Rab\_A* and *Rab\_B* that indicates the number of workers welding in the joining area.

Notice an important object in the frame: the *ShiftCalendar* object – see Fig. 94. This object is used to define the timetable of workers and eventually shifts and pauses.



**Fig. 94: The ShiftCalendar Object**

The object has one checkbox named *Active*. If the checkbox is selected, then the object is used in the simulation, if not it has no effect. In the tab *Shift Times* we can define different shifts, time tables, working days and pauses. In the tab *Calendar* we can define working days. The tab *Resources* is used to select which *WorkerPools* are linked to this object. In our case we have 6 *WorkerPools*: the four for this frame and one for each joining area (RAB). The last tab can be used to create new custom attributes as is possible in any eM-Plant default object.

When a *ShiftCalendar* starts a shift, it sets the attribute *Unplanned* for the material flow of objects, and for any of the *Frames* it controls is set to 'false', i.e. it deactivates unplanned times. If need be, we can program an *Unplanned* time control that executes other actions our modelling situation requires.

The statistics of the material flow objects then start to collect the unplanned time. This is the time when a resource is not planned to work.

When we define shifts that are active from 6 o'clock in the morning until 10 o'clock at night, for example, the planned time lasts from 6 o'clock to 22 o'clock. The unplanned time lasts from 22 o'clock to 6 o'clock the following morning.

When a *ShiftCalendar* finishes a shift, it sets the attribute *Unplanned* of the material flow object and *Frames* it controls to true, i.e. it activates unplanned times. Then, the objects stop processing the current part and release all services.

When a *ShiftCalendar* starts a break, it sets the attribute *Pause* of the material flow objects and of the *Frames* it controls to true, i.e. it activates pauses. For the material objects activating a pause also affects statistics. For *Frames* eM-Plant only changes the state of the attribute. If need be, we can program a *Pause* control for our *Frames* that executes the actions our modelling situation requires.

When a *ShiftCalendar* ends a break, it sets the attribute *Pause* of the material flow object and *Frames* it controls to false, i.e. it deactivates pauses. For the *Frames* we can react to the end of a pause in a *Pause* control that executes the actions our modelling situation requires.

During a simulation run a resource object is either paused or not paused. When not paused, the resource is either available or not available. Not available means that the resource is either failed or blocked. In the available state the resource is either processing or waiting.

Altogether there are six non-overlapping states a resource can be in. The statistics collection period is the sum of the states *Waiting* + *Working* + *Blocked* + *Failed* + *Paused* + *Unplanned*.

This diagram shows how the statistics collection period of a resource is divided up.

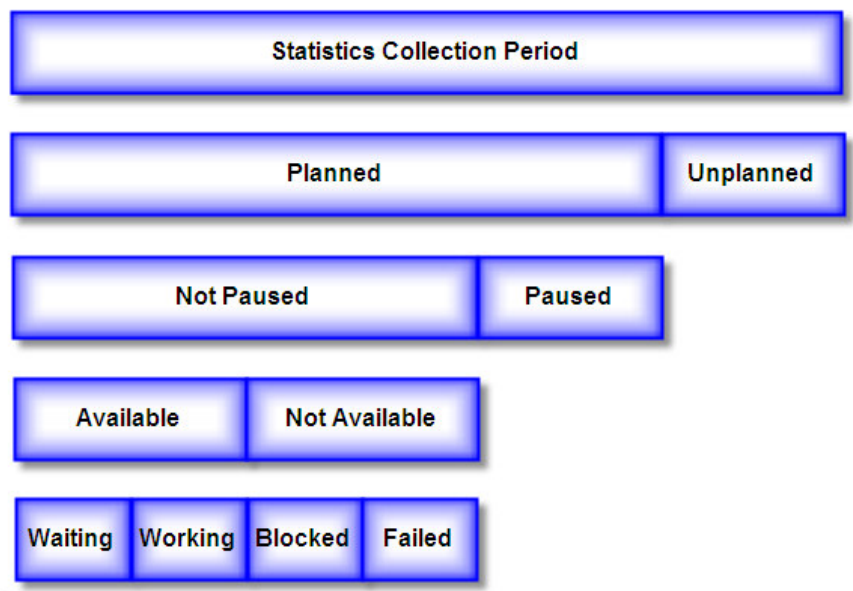


Fig. 95: Different possible states of an object

### 3.2.5.17 The Tables Frame

The frame contains diverse tables and Chart objects allowing the user to visualise results. Some tables are extractions of Access tables – from the database – others are new tables filled in during or after the simulation.

The structure of the frame is the following one:

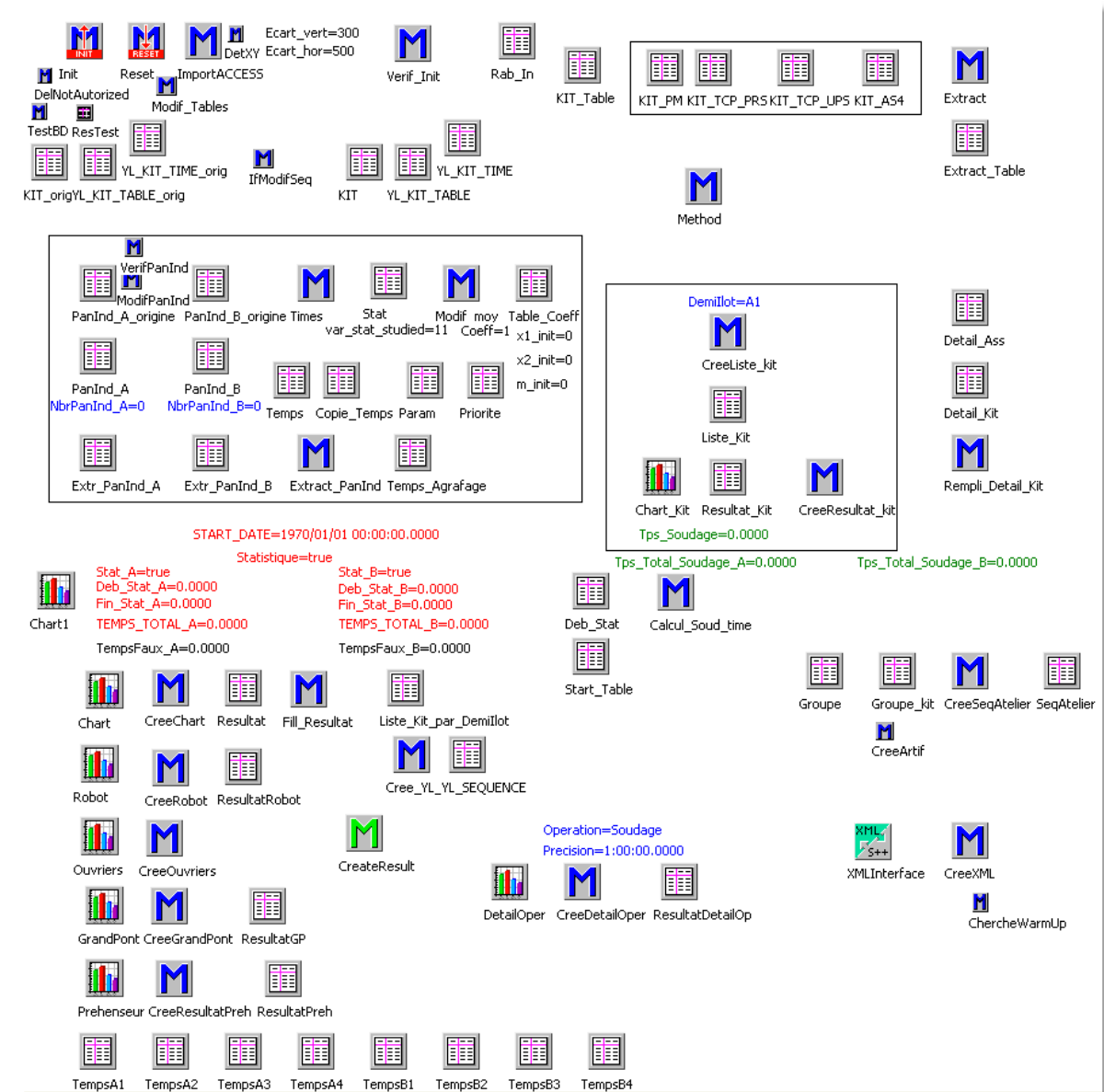


Fig. 96: The "Tables" Frame

Methods of the frame can be grouped in different categories:

- Initialization methods: *Init*, *Reset*, *Extract*;
- Methods used during the simulation: *Times*, *Calcul\_Soud\_Time*;

- Methods generating results at the end of a simulation run: *CreateResult*, *CreeChart*, *CreeOuvriers*, *CreeGrandPont*, *CreeResultatPreh*, *CreeListe\_kit*, *CreeResultat\_kit*, *CreeDetailOper*;
- Methods not used during the simulation but useful for the user in order to generate some tables: *YL\_SEQUENCE*, *CreeSeqAtelier*.

The *CreateResult* method is launched at the end of the simulation and executes all other methods that create results. This method is really important and that is the reason for its special icon – green and not blue as is the case for other methods.

The role of different tables in this frame is explained in methods that create these tables.

Some variables of the frames are quite important and need some explications:

- *Statistique*: this is a Boolean variable that becomes true when the first kit studied enters the workshop. Its value is changed in the *ChgKit* method of the *Ilot\_Demi* frame;
- *Stat\_A* (or *Stat\_B*): same variable as *Statistique* but is triggered only for kits of the A side (or B side). If *Stat\_A* or *Stat\_B* is set on true the *Statistique* variable must also be set on true.
- *Deb\_Stat\_A* (or *Deb\_Stat\_B*): variable with a time format. When the variable *Stat\_A* (or *Stat\_B*) becomes true we record the simulation time.
- *Fin\_Stat\_A* (or *Fin\_Stat\_B*): variable with a time format. When the last sub-panel of the A side (or B) of the simulation is evacuated we record the simulation time.
- *TEMPS\_TOTAL\_A* (or *TEMPS\_TOTAL\_B*): This is not simply the difference between two variables previously described. This is the sum of all red durations of Fig. 97. The variable gives an indication of the total time needed to build all sub-panel of the A side of the workshop (or of the B side).

The last variable is very important. This is this variable that we have to minimize in order to improve the productivity of the workshop. In other words it will be the objective function of the optimisation algorithm.

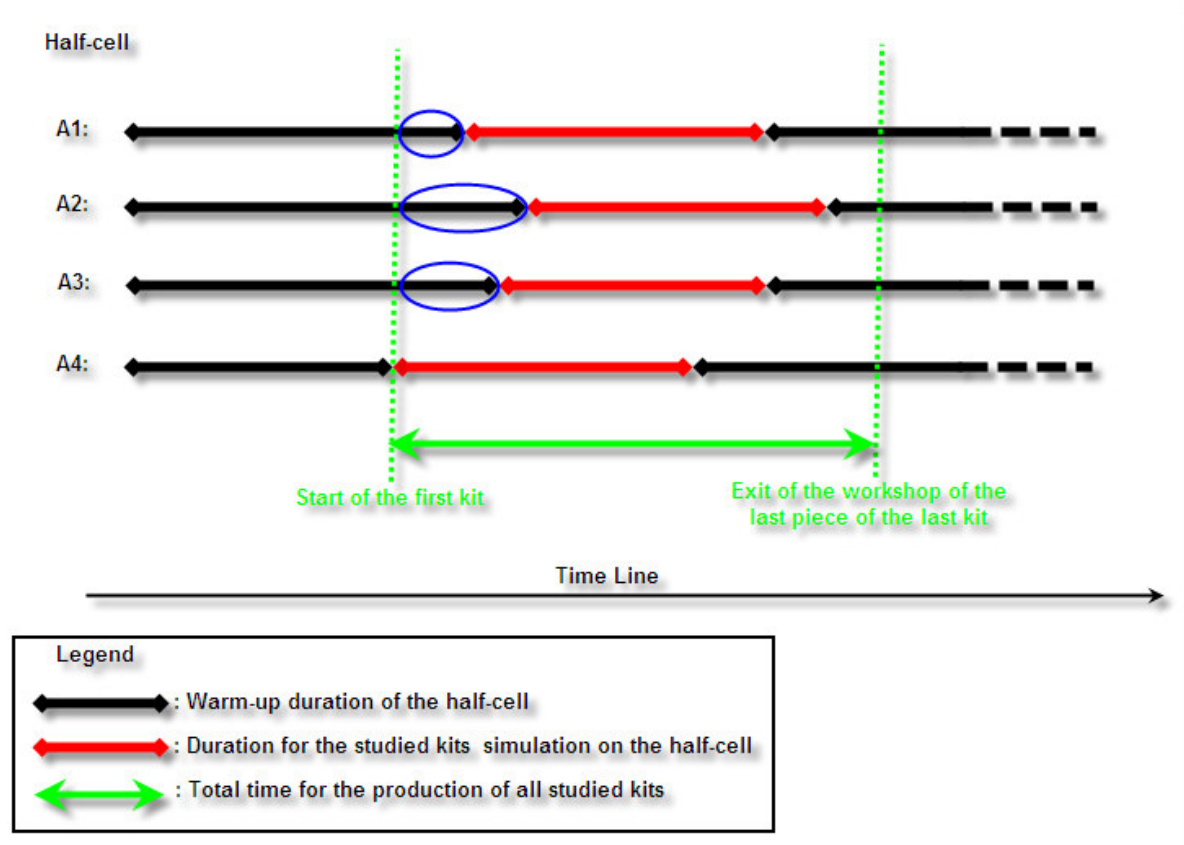


Fig. 97: Total production time

Each horizontal line represents the production on a half-cell. The green dot represents the total time needed to fabricate and move away all assemblies – that value is slightly different than *TEMPS\_TOTAL* variable. Blue circles are time windows during the end of the warm-up period for this half-cell. These times distort productivity results. We have the same phenomena at the end of the simulation in which kits that maybe do not interest our study are built. Consequently we will use for the simulation the sum of the four red lines divided by four. That value is given in the variable *TEMPS\_TOTAL\_A* – or *TEMPS\_TOTAL\_B* for the other side of the workshop. The start and the end of each red line for each half-cell are given in the *Deb\_Stat* table.

- The *Init* method

The method deletes all tables and global variables of the Frame.

- The *Reset* method

Creates the *SeqAtelier* table with the *CreeSeqAtelier* method that uses *Groupe* and *Groupe\_kit* tables – see previous explanations.



Creates also *PanInd\_A* and *PanInd\_B* tables. These tables are a copy of *PanInd\_A\_origine* and *PanInd\_B\_origine* tables – see method *ImportAccess* – with three more columns to save arrival hour, exit hour and duration of sub-panel in the workshop. We create also *NbrPanInd\_A* (or *Nbr\_PanInd\_B*) that indicates the total number of sub-panel to build in the side A (or B) in the workshop.

Tables *Kit\_PM*, *Kit\_TCP\_PRS* et *Kit\_TCP\_UPS* are extractions of the *YL\_KIT\_TABLE* table but they contains one column more to indicate if the piece has already been built or not. The *Reset* method initializes this column.

- The *ImportACCESS* method

This method will extract all the needed information from the Access Database and copy it all to eM-Plant tables. During the whole simulation when any information is required (length of a piece, weight of an assembly, etc.) we do not look into the Access Database! We look into these extractions created with this method. The main reason is that it takes a lot of CPU-time to access external software. It is crucial to minimize these external communications. With this method we do this communication only once. When all the data are loaded we work only in eM-Plant without external links. To make the link with the external software we use ODBC interface with a special ODBC object in eM-Plant:



We just have to link the object to the database and thus we can use SQL request to import information.

In eM-plant we create the following tables with this method:

- *YL\_KIT\_TABLE*: copy of its homonym table Access;
- *KIT\_TABLE*: this is a part of the *YL\_KIT\_TABLE* table where we keep only *NO\_KIT*, *QUANTITY*, *PARENT*, *TRANSPORT*, *TOLBASE* et *CONDIT* fields;
- *YL\_KIT\_TIME*: this is an extraction of its homonym table Access with only *NO\_KIT*, *ELM\_NO*, *X\_LOCAL*, *Y\_LOCAL*, *SOUDTIME* fields;
- *Kit\_PM*: this is an extraction of the *YL\_KIT\_TIME* table where only pieces coming by PM are taken;
- *Kit\_TCP\_PRS*: this is an extraction of the *YL\_KIT\_TIME* table where only PRS pieces are taken;

- Kit\_TCP\_UPS: this is an extraction of the YL\_KIT\_TIME table where only long UPS pieces are taken;
- Kit\_AS4: this is an extraction of the YL\_KIT\_TIME table where only AS4 pieces are taken;
- PanInd\_A\_origine (or PanInd\_B\_origine): this is an extraction of the table Access PanInd\_A (ou PanInd\_B)
- Temps: copy of its homonym Access table;
- Param: of its homonym Access table;
- Priority: of its homonym Access table.

Finally it should be noticed that this frame contains eight tables TempsA1, TempsA2, ..., TempsB4. These tables save for each half-cell the starting and ending time of each operation. At the end of a simulation run we can export these tables to an Excel file and treat results by VBA<sup>11</sup> macros to get a Gantt<sup>12</sup> diagram. We can select all the workshop or only specified cells. Indeed by default eM-Plant does not contain predefined objects to create Gantt diagrams. Nevertheless they are really useful to interpret results mainly as a supplement of a PERT diagram – see Chapter 4. It was easier to use Excel and VBA macros to create our Gantt diagram rather than to try to build specific objects in eM-Plant.

To create the Gantt diagram we have to run an Excel file named Gantt.xls. The file must be in the same repertory as the current eM-Plant file. Once the file is open we have to run a specific macro of the file – named Start – that opens the following dialog box:



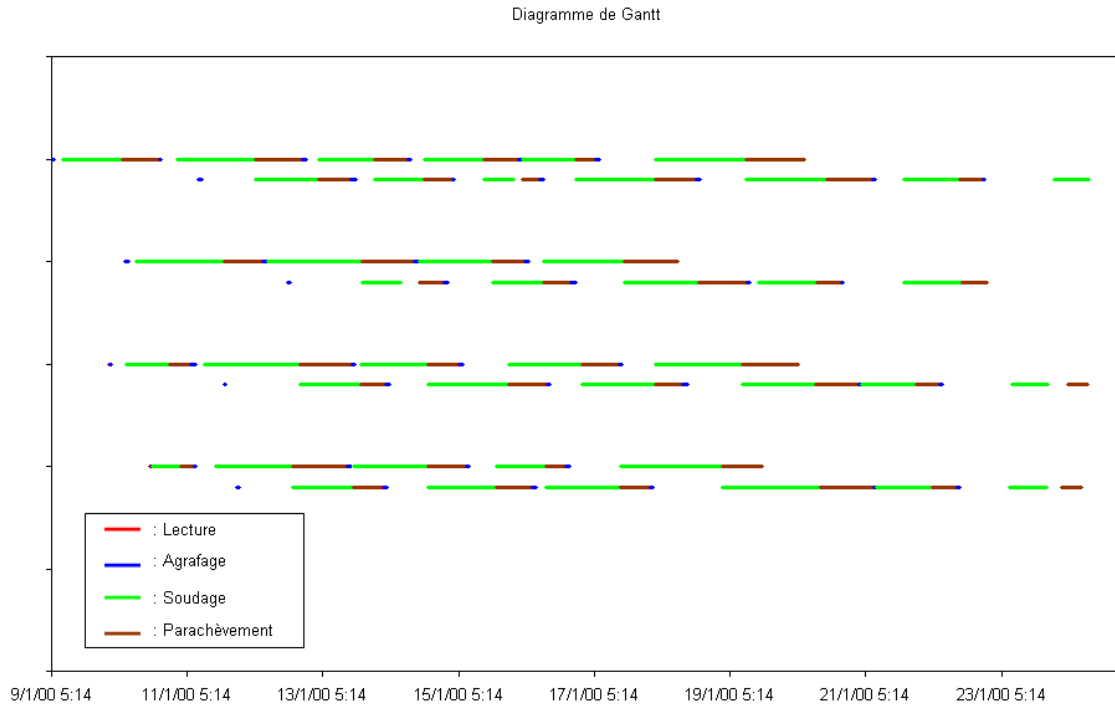
**Fig. 98: Dialog box of the Gantt.xls file**

---

<sup>11</sup> Visual Basic for Applications

<sup>12</sup> A Gantt diagram is a type of bar chart that illustrates a project schedule

In that box we can select interesting half-cells to create the Gantt diagram – A1 is the upper left half-cell, B1 the upper right half-cell, etc. In Fig. 99 an example of a Gantt diagram with the selection of all half-cells is seen. The time line is represented on the horizontal axis and on the vertical axis we have the different working areas. Each operation has its own colour.



**Fig. 99: Gantt diagram of the PrePreFabrication Workshop**

In this example the first line corresponds to the A1 half-cell and the second one to the opposite half-cell B1. The third line is A2, then B2, A3, B3, A4 and finally B4. In one glance we see directly the waiting times and we have a better understanding of what is happening in the workshop. For example the green line gives the welding times We can thus see straightaway if they are saturated or not.

This method *Import Access* should be run manually for these CPU-time reasons. If we want to simulate new kits/assemblies we do a new importation. If we change only the sequence to optimize the total production time we do not need to run the method.

- The *Extract* method

The method copies into the *Extract\_Table* part of the YL\_KIT\_TABLE corresponding to the kit number and “container” given in input of the method. This method is used to generate tables containing all the pieces to be built in STK, DCH and MAT frames.

- The *Times* method

The method uses the *Temps* table that contains the mean times necessary for each event (take a piece with the crane bridge, put down an assembly, etc.) and an associated distribution – normal or triangular. In input we have the name of the event that we will simulate. Two entry fields with a string format are necessary – example “*Evacuation*” and “*Container\_name*”. At the exit we have a time that will correspond to the time of the event to be simulated.

If the distribution chosen is “normal” the time calculated can be negative! To avoid this situation we introduce limits on this time. For example when the calculated time is below the tenth of the mean we impose the value on this tenth. In this way we distort the distribution – that will no more respect the mean – but this difference is really minimal.

- The *Extract\_PanInd* method

The *PanInd\_A* and *PanInd\_B* tables contain information related to sub-panels of all the simulation – thus even those that will not be simulated; the user does not necessarily select and simulate all the assemblies of the simulation. The method is executed in the *CreateResult* method.

- The *Calcul\_Soud\_Time* method

The *CreeList\_Kit* method provides some results for a specific half-cell. This half-cell is selected with the string variable *Demillot*. Results that can be obtained are in particular welding times. The *Calcul\_Soud\_Time* method will calculate total welding time for each half workshop in running the *CreeList\_Kit* for each half-cell and adding times into variables *Tps\_Total\_Soudage\_A* and *Tps\_Total\_Soudage\_B*.

- The *CreeChart* method

The *Chart* object is used to represent in graphics different kinds of results contained in a table. This object is linked to the *Resultat* table. The table contains for each kit the duration of each operation: preparation time, tacking time, transfer time, welding time, finishing time and evacuation time with waiting time between each one. The method adjusts and modifies the presentation of the graphic. An example can be seen in the Fig. 100.

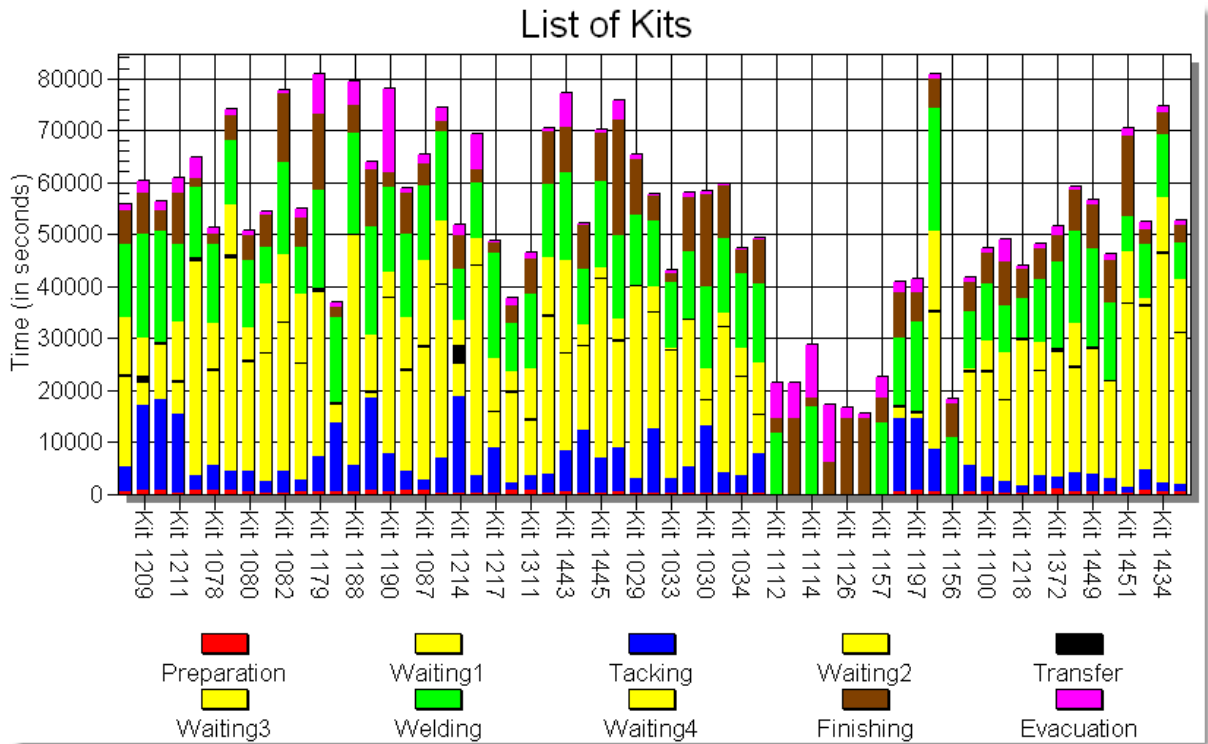
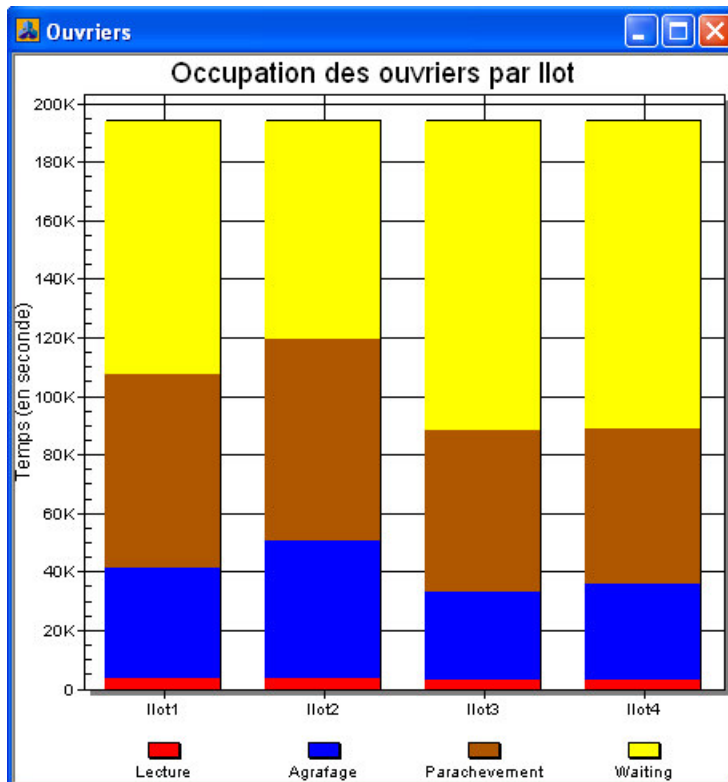


Fig. 100: Graphic representation of time operation for a list of kits

- The *CreeOuvriers* method

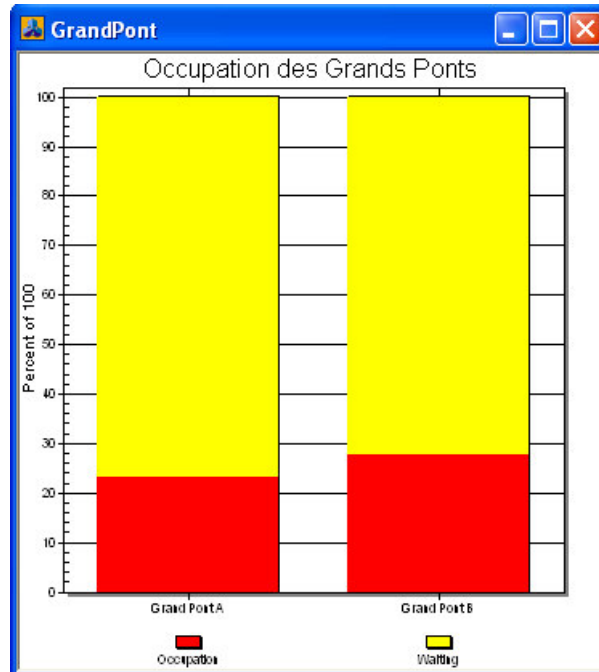
The *Chart* object name “*Ouvrier*” is linked to an internal table of the object that contains *Tps\_Preparation*, *Tps\_Agrafage*, *Tps\_Parachevement* and *Tps\_Waiting* variables contained in each *Ilot* frame. The object is used to represent the workers occupation.



**Fig. 101: Occupation of workers for each cell**

- The *CreeGrandPont* method

This method creates the *ResultatGP* method with crane bridge occupation times coming directly from the *Pont* frame. Waiting times are also deduced. The *GrandPont* chart object is linked to this table and can give results such as in Fig. 102.

**Fig. 102: Occupation of cranes bridges**

- The *CreeResultatPreh* method

The method creates the *ResultatPreh* table with the occupation times of the eight mechanized grippers. Waiting times are also deduced. The *Prehenseur* chart object is linked to this table and can give results such as in Fig. 103.

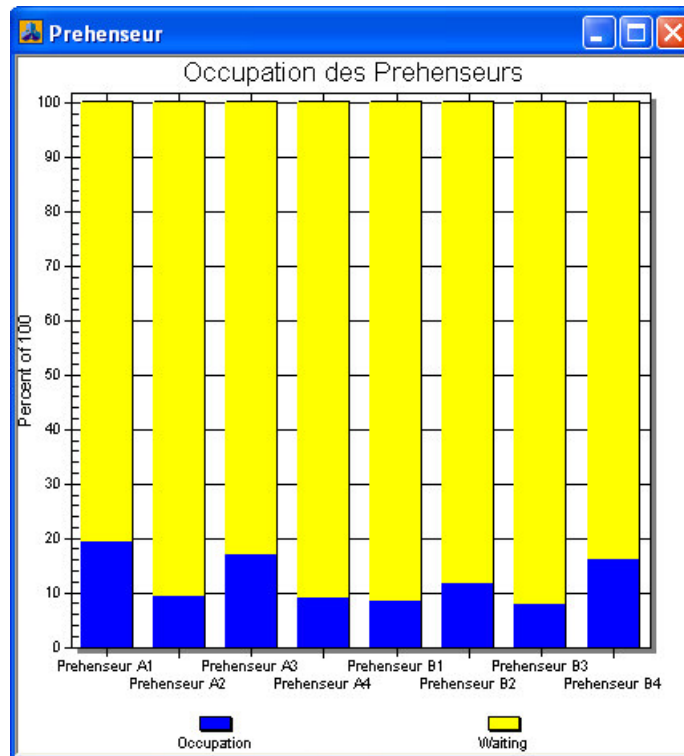
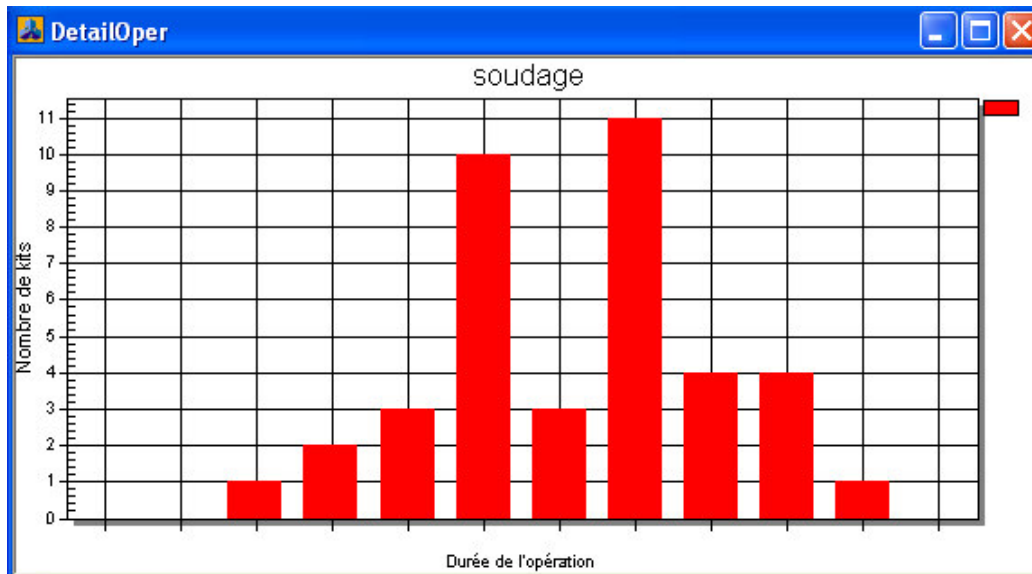


Fig. 103: Occupation of the mechanized gripper for each half-cell

- The *CreeDetailOper* method

The method creates the *ResultatDetailop* table that gives some information about a selected operation – chosen in the string variable *Operation*. For this selected operation we introduce a time interval. The method gives the number of kits that have this operation time in the meantime. For instance we can see kits that have a welding time greater than 3 hours and lower than 4 hours. This could be useful if we want to balance the workload for the welding robot or to see other effects. Each operation can be studied. This table is linked to the Chart object *DetailOp* that can generate a graphic – see Fig. 104.



**Fig. 104: Welding duration division**

In the figure we observe that one kit has a welding time included between one hour and one hour and a half, two kits between one hour and a half and two hours, etc... In this case most of the kits have a welding time between two hours and a half and four hours.

- The *CreeListe\_Kit* method

This method creates the *Liste\_Kit* table. The table contains list of kits built on the half-cell selected by the user in the *Demi\_Ilot* variable. We then run the *CreeResultat\_kit* method that will – thanks to this table – generate the *Resultat\_Kit* table. This last table is an extraction of the *Resultat* table that contains the duration of each operation for simulated kits. We extract the needed data (given in *Liste\_Kit*) and insert it into *Resultat\_Kit*. Thanks to this table we can find the total welding time of this half-cell and save it in the *Tps\_Soudage* variable.

- The *CreeResultat\_kit* method

As explained before, the method extracts from the *Resultat* table kits given in *Liste\_kit* into the *Resultat\_Kit* table. This table is linked to the chart object *Chart\_Kit* that will give the times of each operation for all the kits built on the selected half-cell. The resulting graphic is thus a kind of extraction of the *Chart* object. An example can be seen in Fig. 105.



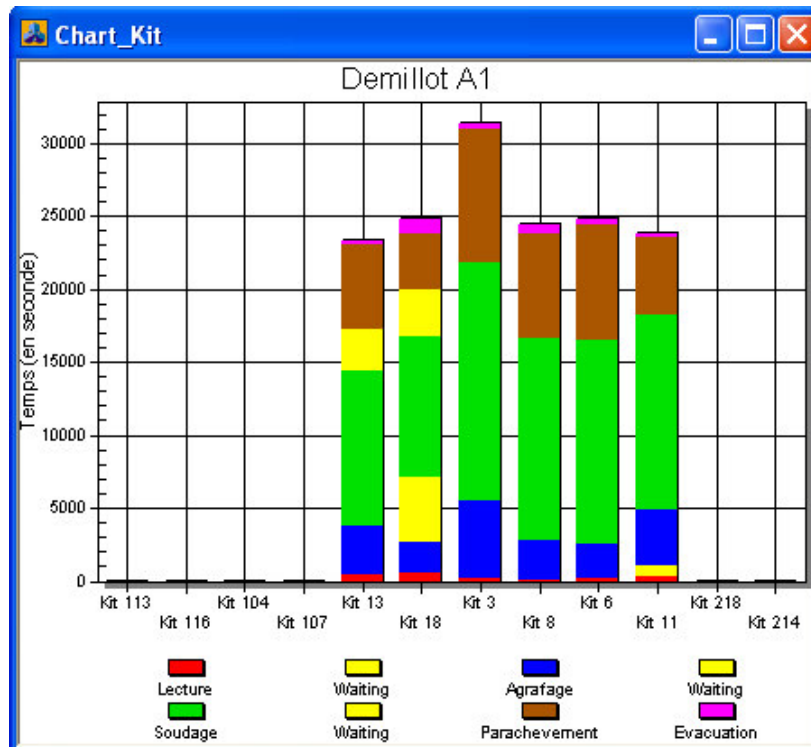


Fig. 105: Detailed operations done on a selected half-cell

Note that kits are given in their arrival order on the half-cell and not in their number order.

- The *CreateResult* method

The method generates all the graphic results. This is the last method run in the simulation. It is triggered by the *Action* method of the frame *Pont*. The method calculates also *Temps\_Total\_A* and *Temps\_Total\_B* variables depending on other time variables of the frame.

The following methods are successively run in *CreateResult*: *Extract\_PanInd*, *CreeChart*, *CreeOuvriers*, *CreeGrandPont*, *CreeResultatPre*, *CreeDetailOp*, *CreeListe\_kit* and *Calcul\_Soud\_Time*.

The method also copies eight eM-Plant tables *TempsA1*, *TempsA2*, ..., *TempsB4* in an Excel file name *Gantt1.xls*. Each table is copied in a different sheet of the file. By default this file is situated in the same repertory as the current eM-Plant file but its path can be changed in the method. This Excel file will be treated by some VBA macros in order to establish the Gantt diagram of the workshop.

Notice that the *Robot* chart does not necessitate a method to be built. It contains an internal table directly linked to the statistics of the *SoudMachine* object of each cell. These objects are used to model the welding times. We can have results such as in Fig. 106.

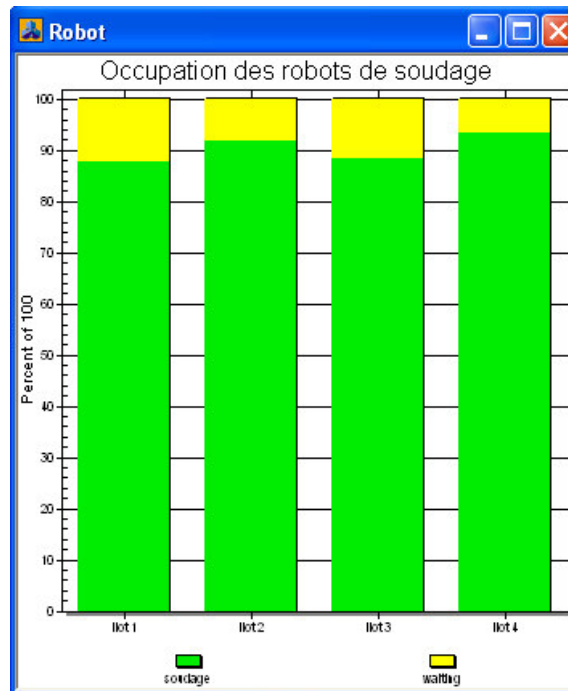


Fig. 106: Occupation of the welding robot for each cell

- The *Cree\_YL\_SEQUENCE* method

The method creates the *YL\_SEQUENCE* table (from the *Liste\_Kit\_par\_demillot* table). The table can be used to create a starting half-cell sequence. The *Liste\_Kit\_par\_demillot* table is obtained after a simulation run with a half-workshop sequence. We are thus sure that the obtained table respects all the restrictions – no long kit on small cells, etc. – which is not necessarily obvious manually. This is thus a tool to create an initial *YL\_SEQUENCE* table that can of course still be modified after. This method is not really used in the simulation.

- The *CreeSeqAtelier* method

The method creates the *SeqAtelier* table depending on two tables: *Groupe* and *Groupe\_kit*. In the table *Groupe* we have two columns: the first one contains the list of groups – remember that kits belong to a group – to be built in side A and the second one for side B. The table *Groupe\_Kit* contains a column by group with the list of all kits that constitute it. We see that both these tables are thus sufficient to establish the production sequence for each side. The method analyses the two tables and creates a production sequence and inserts it into the *SeqAtelier* table.

- The *CreeXML* method

This method is used to create an XML file that contains the operation times of each studied kit. The file created will be read by software that can automatically create a PERT<sup>13</sup> diagram. That coupling is a very important task of this thesis and will be explained more in detail in the Chapter 4.

### 3.2.5.18 The “Result\_Log” Frame

This frame has been created in order to manage external files – txt, xls, etc. The first goal was to add the possibility of creating log files of each simulation run. These logs files have several goals:

- to have a detailed view of each importation step during the simulation run;
- to facilitate the detection and correction of eventual bugs in the software.

The second goal of the frame is to carry out the interface between the BDRO<sup>14</sup> and the PHL<sup>15</sup> files. All the objects of this frame could be located in the *Tables* frame because the importation of the Access Database is done there but the number of elements in the *Tables* frame was already very high. For more clarity it has been chosen to create a new specific frame.

The structure of the frame can be seen in the following figure:

---

<sup>13</sup> *Program Evaluation and Review Technique*

<sup>14</sup> *BDRO: Resource planning tool used by the Aker Yards France shipyard*

<sup>15</sup> *“Programmation Hors Ligne” – meaning programming of the Welding Robot before the real production*

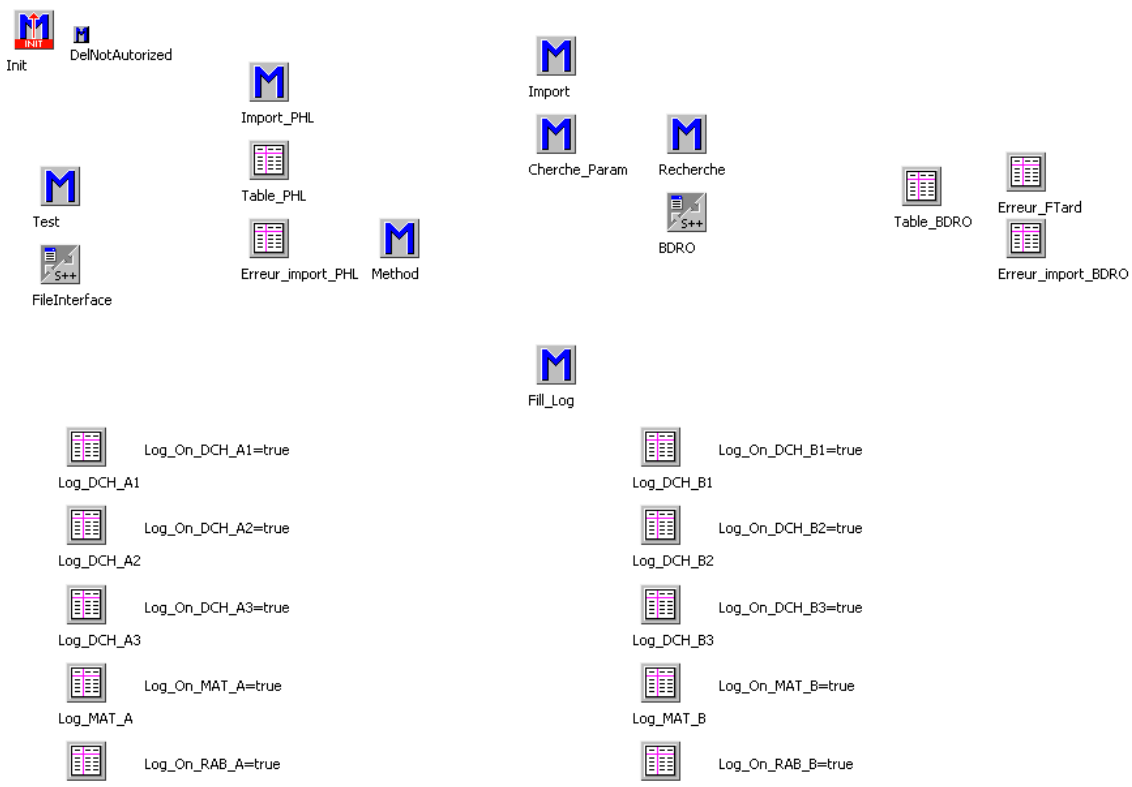


Fig. 107: The "Result\_Log" Frame

- The *Import* Method

This method carries out the importation of the BDRO into the simulation. The BDRO is a database that contains dates information about a sub-panel. In fact we always work with an extraction of the BDRO database – not all the database. The extraction result is an Excel file. Each line of the file corresponds to a Sub-panel. Columns correspond to different fields. The most important one is the “FTard” field that gives the latest starting date of the sub-panel. For each Sub-panel this is in fact the only field that we will import. If a sub-panel is in the BDRO file but not in the Access Database – the main database with all geometrical characteristics – we do not import data. That could happen because the Access Database is also an extraction and we do not treat all the ship in this workshop. Consequently, in case of a totally new simulation we first have to carry out the Access Database importation before the BDRO importation!

The method is launched automatically when the user clicks on the importation button in the main interface.

To carry out the importation the BDRO file must be in the same repertory as the eM-Plant file.

- The *Import\_PHL* method

This method treats the PHL importation. The PHL – “Programmation Hors Ligne” – is the programming of the welding robot before the real production. Indeed robots need to have very detailed programming to know which movements to make in order to weld these assemblies. This programming is done with specific software developed for the robot itself. A lot of very important results can be given from this programming. For example we can get the total welding length and an estimation of the welding time by the robot! That last information is really important and of course much more precise than the welding estimation time given by the simple formula found in section 3.2.3.7! Furthermore the programming can also give total the welding length that cannot be done by the robot. Consequently we can also have a much more precise estimation of the finishing time for the assembly. These data are thus really useful in improving the quality of the simulation model.

The PHL programming furnishes *.txt* or *.xls* files – to be chosen by the user. Interface developments have been done for *.xls* files. We have one PHL file by week of production. The name of the file is:

PostGenerationLog-Year-Number of weeks.xls

Each line of the file corresponds to an assembly and the columns correspond to different attributes. The attribute *TotalTime* is the most important and gives us the total time necessary to weld the assembly with the robot.

First of all the method checks the earliest and the latest *FTard* of the Sub-panel imported from the BDRO. For each of them we look to see if we have the corresponding *PostGenerationLog* file. If so we check in the file if we can find interesting data for some assemblies. Notice that by default we can select in the interface of the software if we will use (or not) these data from the PHL. If the user decides to use it is nevertheless possible not to have all the data for all assemblies. If an assembly does not have the PHL information in that case we simply use the old formula to evaluate the total welding time for this assembly.

To carry out the importation PHL files must be in the same repertory as the eM-Plant file.

- The *Fill\_Log* method

This method is run at any important event of the simulation. In input we must specify the name of the area, the name of the piece, the kit number, the sub-panel identification and the simulation time. The method inserts this information into the corresponding table – *Log\_DCH\_A1*, or *Log\_DCH\_MAT\_B*, etc. To minimize CPU-time it is possible to not record

some information by deactivating some events. This is done with different variables – such as *Log\_On\_DCH\_A1*, or *Log\_On\_DCH\_MAT\_B*, etc. – that can be switched on or off.

### 3.2.5.19 The “Resultats” Frame

For a simulation run most of the results are given in the Tables frame. Nevertheless it is interesting to run several simulations and to calculate mean results because the model is highly stochastic. The multi-run simulation can be done thanks to the *ExperimentManager* object. The results obtained are stored in this frame.

Some graphics cannot be obtained, mainly if we use a half-workshop sequence. This is the case for the graphic that shows the operation time of each kit by half-cell. Indeed due to stochastic effects kits will not be built each time on the same half-cell! Consequently it is not possible to establish means and to create the graph.

With the *ExperimentManager* we can execute several “observations”. An observation is just a complete run of the simulation. Several observations always have exactly the same configuration: the same sequence, same data, etc. The user just has to specify the number of observations to make and eM-Plant will execute the same number of simulations. The only change between each observation is the Stream. Results will be different only due to random numbers used in the simulation – almost each time is linked to a random distribution. We can specify for the *ExperimentManager* which results must be kept at each simulation and afterwards calculate means, variance, etc of these results.

Nevertheless we can also ask the *ExperimentManager* to run several “Experiments”. An experiment has its own starting characteristics – for instance its own sequence of kits. Two experiments are also two simulation runs but the results will be different not because of stochastic effects but because initial configurations are different. We can specify for the *ExperimentManager* a number of experiments to run and for each one specify the initial configuration. In our case, experiments will generally be different because of their sequences of kits. Furthermore for each experiment we can specify the number of “Observations” in order to minimize the stochastic effects on results. At the beginning and the end of an Experiment the *ExperimentManager* modifies the configuration of the workshop. Specific initialization can be carried out by the user with special methods – *Configuration* and *Evaluation* methods. These last methods are internal to the *ExperimentManager* object but can be changed manually.

The structure of the frame can be seen in the following figure:

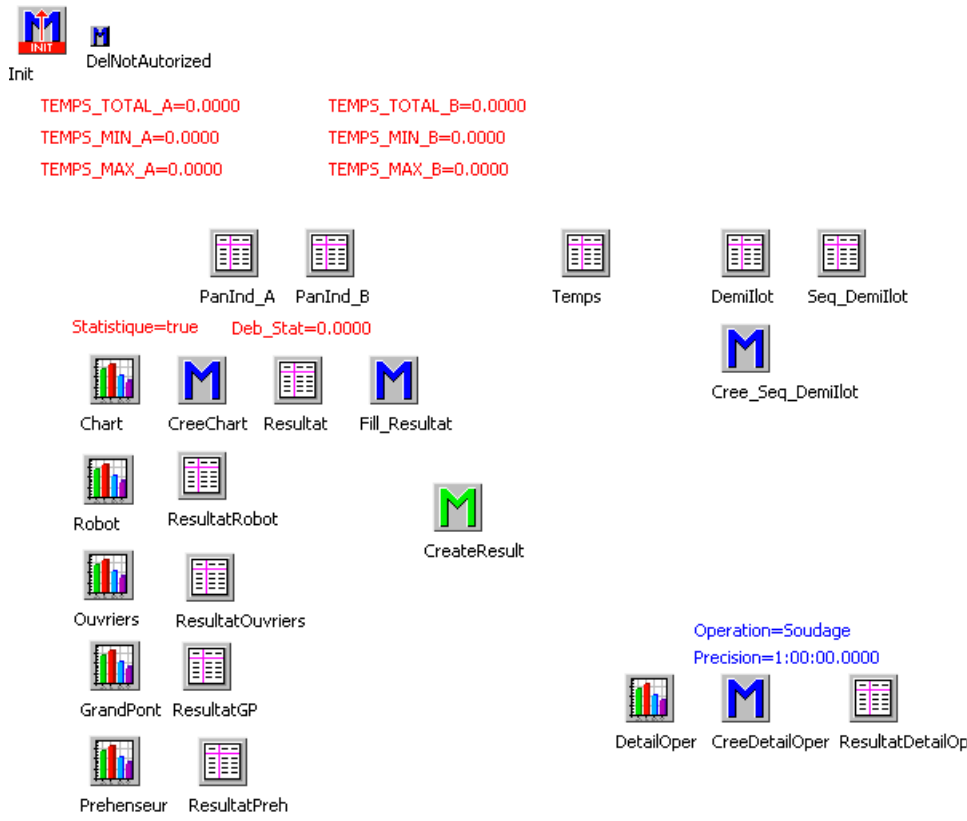


Fig. 108: The "Resultsats" Frame

We find in the frame a list of objects almost identical to the *Tables* frame but with results that correspond to several simulation runs.

### 3.2.5.20 The "Optimisation" Frame

This frame is used to carry out the optimization of the workshop with a genetic algorithm. Explanations of the optimisation are given in the next chapter and detailed explanations of the frame will be given there. The structure of the frame will also be explained in the chapter.

## 3.2.6 Analysis of a realistic problem

### 3.2.6.1 Introduction

This section treats a realistic case to show the potential results of the simulation. A list of assemblies grouped in kits is defined and simulation is performed with the developed tool. In previous sections all the developments that lead to the creation of the simulation model have been explained.

We explain here the methodology to follow as if we were in charge of the workshop schedule. The resulting values obtained are not the main points of this section. To get realistic

results we need a very precise calibration between the model and the real workshop. Unfortunately this “very precise calibration” has not been carried out in the available timeframe of this thesis. A “timing study” called “pointage” has been carried out during one complete week in the workshop to get unitary times. But that data collection is not complete enough to guarantee the accuracy of figures. To have a very good calibration simulation, developers must be near the workshop to have a direct return from workers of their developments. This was unfortunately not possible for the thesis; the workshop is located at about 700 kilometres from the University of Liege! Adjustments and calibration must be done day-by-day. The goal was not to get realistic values of the workshop but to demonstrate the possibilities of simulation in the shipbuilding industry. The methodology used is more important than the values obtained. This point is important and must be kept in mind during the analysis of this section.

As explained before, data which come from different databases are required. First of all we of course need the list of assemblies to be built. We make the hypothesis that they are already grouped by kits. In fact in reality there is a scheduler that is in charge of “creating” kits. The rule used is simple and is based on the assemblies’ geometry and on a rough estimation of their welding time. Note that the goal is to try to balance out the working load so this scheduler also chooses the side of each assembly. Remember that each assembly also belongs to a sub-panel and we are not allowed to have assemblies of the same sub-panel on both sides of the workshop. In conclusion we get two lists of assemblies grouped by kits. These two lists are given in one table: the KIT table – given in Microsoft Access© format.

The available list is not enough for the simulation because we need information about individual pieces and also other information related to the simulation. This data is contained in the corresponding tables STEELM and STEPRT – see section 3.2.3 for explanations. Thus there are three new tables: KIT, STEELM and STEPRT. To treat these tables we have to execute Access requests described in section 3.2.3. In fact we have created a simple interface – see Fig. 109 – to manage all the treatment. If we have a new kit table, we just have to push on the “Kit” Button. If we have a new STEELM or a new STEPRT, we have to click on the “New STEELM or STEPRT” button. Of course the data of these last tables should contain information related to the KIT table. This treatment creates the YL\_KIT\_TABLE Access table that will be imported into the simulation model.





**Fig. 109: Simple interface to manage the Access database treatment**

This importation is done by running the *Import\_Access* method in the *Tables* frame. An easier way is to use the interface developed – see Fig. 79 – and to click on the “*Importation des données*” button. During the importation an automatic checking of data consistency is carried out – for instance each assembly must have at least one main plate.

Relating to the list of assemblies we need the date information. Indeed there are restriction dates due to the workshop following on from the PrePreFabrication workshop. These dates give us the sequence order to respect. This kind of information is given in the BDRO database. We assume we will get an extraction of the database under an Excel format – BDRO.xls.

The first step is to check if data coming from the Access database and from the BDRO database are coherent. We must have data about the same assemblies. This verification is done during the importation of the BDRO. It means that it is important to first do the Access importation before the BDRO one. The BDRO importation is done with the interface – see Fig. 79. This importation automatically defines a sequence of production for the kits based on the *FTard* on assemblies.

### 3.2.6.2 Results of a simulation

In our benchmark there are a total number of 494 assemblies. They are constituted of 2910 individual pieces or sub assemblies. We do not want to simulate all these assemblies and we limit our choice to a certain period of time according to the BDRO table. After these treatments we finally get a list of 140 assemblies to simulate which are grouped in 61 kits for a total of 595 individual pieces. They belong to 20 different sub-panels. Among these assemblies we make the hypothesis that some are currently in the workshop and will thus constitute our initial situation. Some kits are supposed to be already tacked, others already welded. We also have plates that are already joined in the RAB areas. In our initialization dialog box we have defined the following parameters as in Fig. 110.

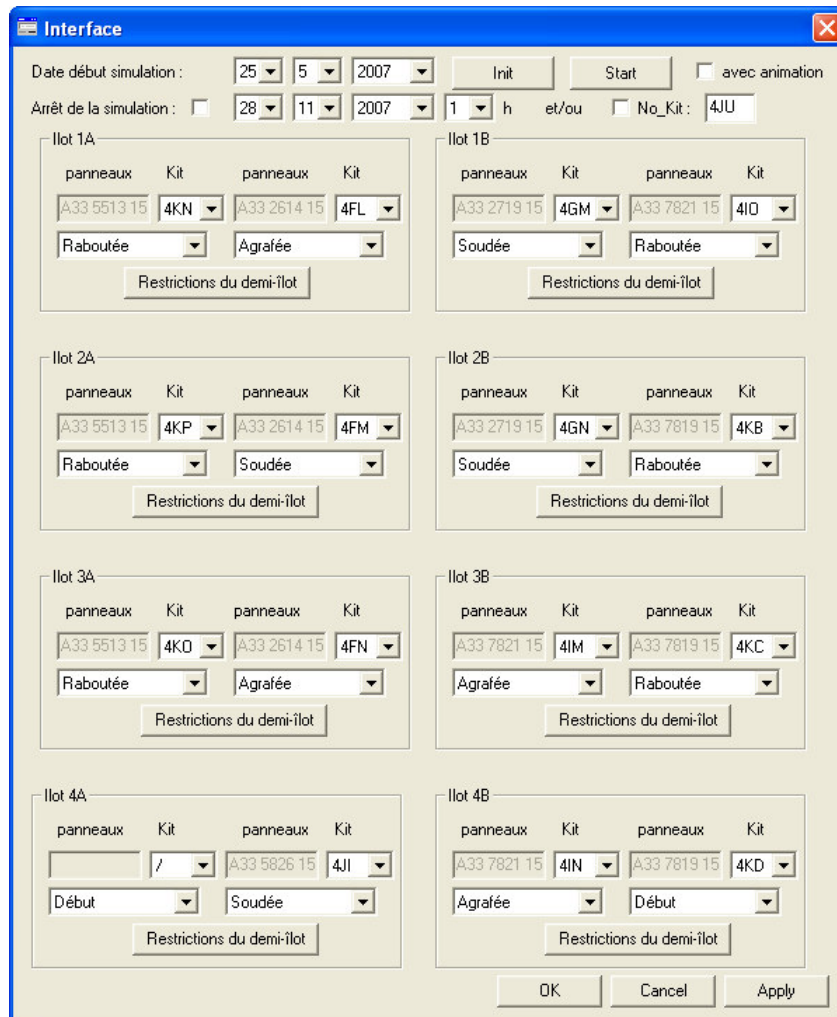


Fig. 110: Parameters of the Initialization

From a simulation point of view the state of the workshop is as in Fig. 111.

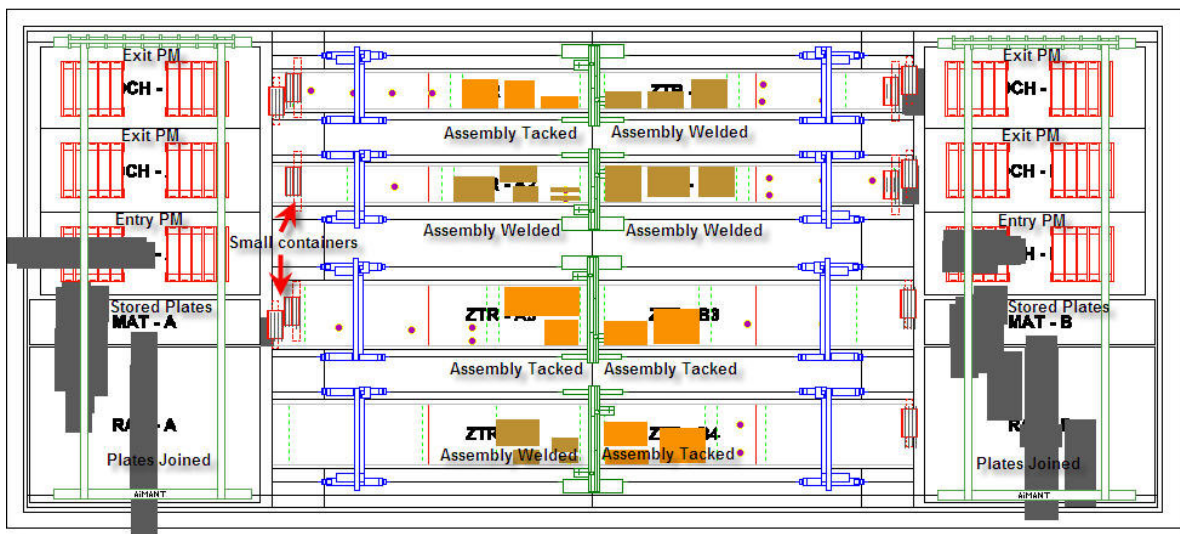


Fig. 111: Initial situation of the workshop

Internal working areas are already occupied by assemblies at a finished stage: some are tacked and others are welded. Notice that in reality the initial situation is not at a finish stage

– the operation is going on – but due to the system discrete-event methodology it is not possible to start in the middle of an operation. However it is possible to block the half-cell during a time specified by the user. In other words if we know that the kit will be welded in two hours we can specify that the kit is welded but we block the half-cell for two hours. In our benchmark we will not use this option and we will suppose that the next operation can start immediately.

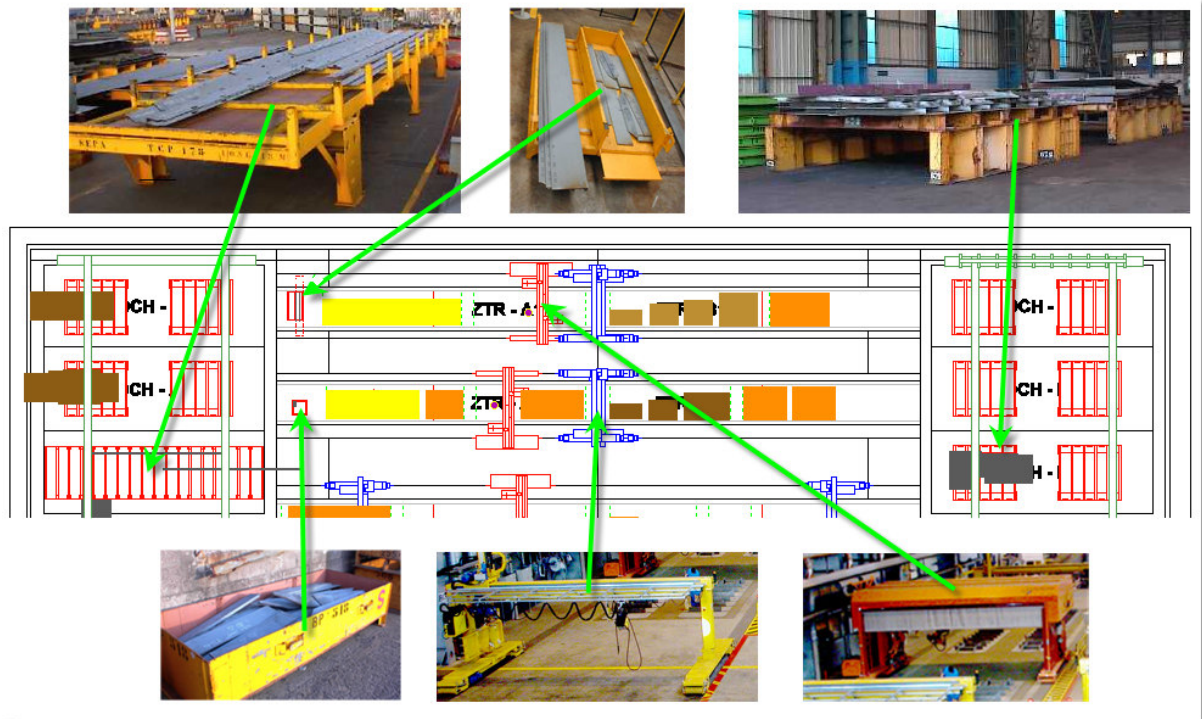
Outside working areas have been assigned to a kit – except for 4A – so corresponding containers have already been brought. We have also defined that for these kits the joining is already done.

What will be our fabrication sequence? By default we have explained that when we do the BDRO importation kits are automatically sorted by their *FTard*. This sequence will be our initial and reference sequence for this benchmark.

Parameters of the workshop are set by default – unitary times, speeds of handling tools, etc. We have imported a list of kits with their characteristics and we have a production sequence → the simulation can be launched. When the simulation is running we can choose to visualize what happens at different speeds or just to simulate without visualization. If we visualize the simulation at a certain speed we have to keep in mind that what we see is not just an acceleration of real time. Indeed the simulation functioning is based on discrete-event modelling. It means that we have a list of events to carry out and once an event is done we go to the next step and execute the next event. One event is for example: the exit of a piece from a machine, or a movement by a crane bridge, etc. The real time between each event can vary widely – from milliseconds to several hours or even longer. But the simulation time between each event is constant and depends on the speed chosen by the user. It is a very important point to take into account when we visualize the simulation! Because when we look at the simulation we will always see something happening – the execution of an event – and that does not correspond to reality!

Nevertheless there is also the possibility of simulating at real speed – which is not in practise really used. In that case the software automatically calculates the time to wait between each event to correspond to reality. If the simulation model is a production model we of course understand that it is not really interesting to simulate at real speed.

During a simulation several things can be observed. Different objects of the simulation have already been explained with their representation in the model. Fig. 112 shows again what we can see during the simulation in recapitulative way.



**Fig. 112: View of a part of the workshop during the simulation**

As explained before the colour of the assembly gives an indication of its state: yellow when the basic plate has arrived, orange when the assembly is tacked, light brown when welded and dark brown when finished. The handling equipment also has different colours depending on their activities.

For that benchmark the list of sub-panels to be built is the following one:

Side A	Side B
2614 15	2719 15
5826 15	7821 15
5513 15	7819 15
5510 15	6704 04
5304 01	5304 16
5305 01	8206 15
5517 01	6704 16
8106 15	8108 15
6704 15	5305 16

5610	15	5804	20
------	----	------	----

**Fig. 113: List of sub-panels to be simulated**

For each side, the first three sub-panels are used for the initialisation of the workshop as specified in Fig. 110. The order is directly linked to the BDRO. Each sub-panel contains at least one kit so we have the corresponding kits building list. And again for each kit we have at least one assembly that also gives an assemblies building list. The first sub-panel – highlighted in green – will not be treated in the results.

The rule used for the simulation is called “Half-Workshop sequence”. That means that we have fixed the sequence of assemblies but we do not know in which working area they will be built. The working area chosen is simply the empty one! Notice that the sequence is not rigid but can slightly change. Indeed there are some cells restrictions: the biggest assemblies cannot be built on the smallest working area. If a small working area is empty and the next kit is a big one we skip it and do the kit after. It is the reason why the sequence is not a rigid one.

If we run the simulation with the default sequence we can get our first results. The simplest one – and probably the most important – is the total production time necessary to finish the list of assemblies. As explained before we cannot just look at the simulation time at the end, because we have to keep in mind that one side of a workshop could finish much earlier than the other. The parameter to study is thus the time to produce the kits of side A plus the time to produce the kits of side B. For this test we get a time of:

4 days, 12 hours and 46 minutes;

In this case we do not take into account the timetable of the workers. They are supposed to work 24 hours a day. We know this is not the case in reality but results are easier to compare between different sequences when we do not take into account pauses etc. Indeed, imagine that the workshop is working in shifts. We suppose that the working day starts at 6 am and finishes at 6 pm. For a first sequence we run the simulation and see that the work is finished after 8 days at 5 pm. If we run a second sequence that in fact lasts only two hours more, we will finish this time after 9 days at 7 am! Clearly comparisons between production sequences are more interesting when we do not take into account workers’ shifts.

A PERT diagram can directly be obtained after a simulation run. This is possible thanks to the link created between the simulation model and the PERT tool developed – see Chapter 4.

Once we get the results for one sequence it is important to run again and again the same sequence to see the influence of random effects. We can immediately see the variance of the total production time.

### **3.2.6.3 Statistical analysis**

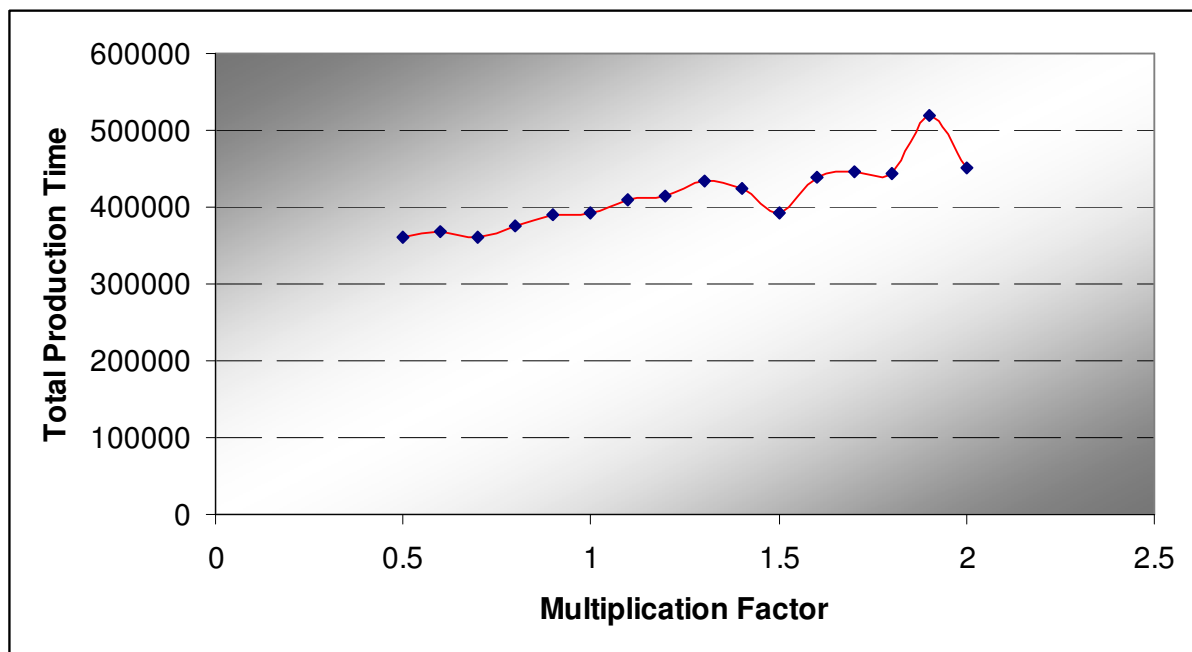
An important interest of the simulation is to do a statistical analysis of the process. It involves studying the impact of each parameter on the total production time. This analysis is also essential in improving the quality of the simulation. Indeed if a parameter has no major impact on the simulation we know that we do not need to refine the simulation model related to this parameter. On the other hand, if a parameter has a big influence it indicates that we must be careful about the modelling! A lot of importance must be accorded for the calibration of these parameters.

The default values of main parameters have been given by the Shipyard. For each of them we will change the mean value to see the effects on the total production time. We will change them from half the initial value to twice that with a step of ten percent. Afterwards we can class parameters from the most important ones to the least important.

With a simulation model developed we see that we can immediately get a huge amount of statistical data. In our case we just have changed values one by one but what happens if we change two or three or more values at the same time? Very complex and complete statistical studies can be carried out and the simulation can create lots of results. Consequently, with a complex study we can find a correlation between each parameter. We can see the influence of one parameter on another parameter. This point is one of the simulation's big attractions. Results derived from the simulation can be linked to neural network software to execute analyses and to give us these correlations between each variable. In our case – because it is an academic approach and because the calibration is not perfect which means that we have no interest to go further in results – we will not carry out such a detailed analysis. But this is a possibility of the simulation and one important aspect that we have to take into account when we have to choose if we will develop a simulation model or not. If the system is correctly modelled it can be understood in a very detailed way! These statistics analyses could also be done if we have collected all the data directly from the real workshop. In practice this is rarely done and we do not necessarily have enough statistical values to get reliable results. With a simulation model, we can generate hundreds and hundreds of more results than in reality. It means that results will be more reliable!

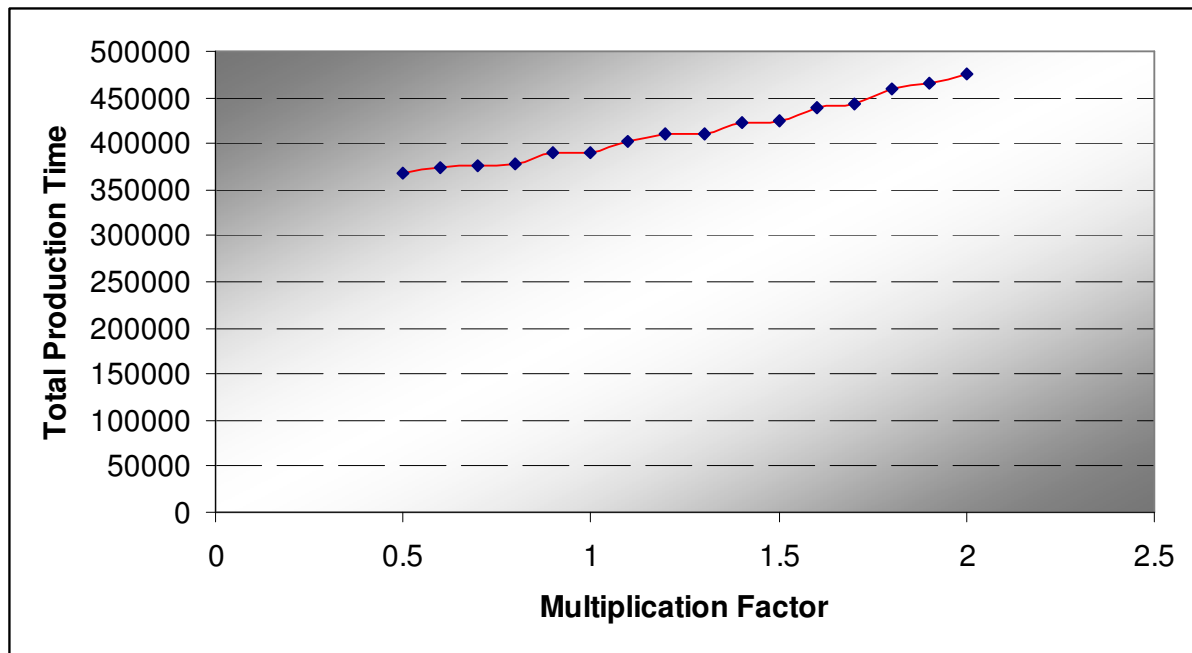
We will now show some examples of the results that can be obtained. First of all we have to select the parameter to analyze. In this case we will try to see the impact of the finishing time on the total production time. The finishing time has been defined as a percentage of the welding time of the assembly but it is linked to a distribution. In other words the mean is indeed proportional to the welding time but the process is nevertheless stochastic. We will multiply the mean by a factor varying from 0.5 to 2 with a step of 0.1 and observe the influence on the total production time. Note that we have to adapt other parameters to have the same distribution. For instance we cannot change the average value without adjusting variance and so on.

Results obtained are shown in Fig. 114.



**Fig. 114: Impact of the Finishing time on the Total Production time**

Some results could seem strange like the ones with a factor of 1.5 or 1.9. We have to keep in mind that the process is strongly stochastic and to get this graphic only one simulation run has been done for each multiplication factor. It is far better to run several simulations and to take the mean. In this case we get Fig. 115 with ten simulations per factor.



**Fig. 115: Impact of the Finishing time on the Total Production time with several runs**

Now we see that the influence between finishing and total production time is really highlighted. If we divide the average value of the finishing time by two the reduction of the total time is 6%. If we multiply the finishing time by two the total time will increase by 22%! This kind of information is important for two reasons:

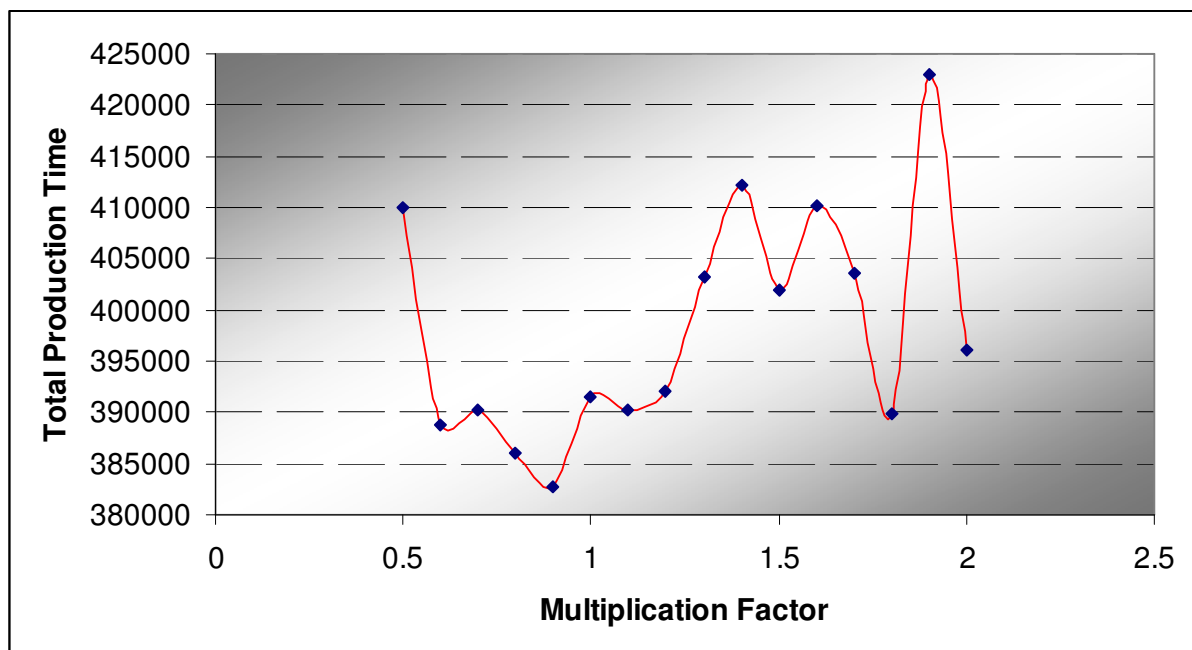
- First we can see which operations must be carefully modelled. Indeed it is not useful to spend time developing in detail operations that do not have a key impact on the total production time;
- Secondly it gives clues to the scheduler about which operations must be studied as a priority to increase the productivity of the workshop.

When we develop the model and are carrying out the calibration we must continuously make that kind of analysis. For instance we develop a first version of the simulation and we see that the tacking time does not really have an effect, contrary to the finishing time. Consequently for the second version of the model we develop a much more precise model for finishing operations. Now the average value of this operation could have changed a lot – because it is more refined and we have maybe seen that our old version overestimated the time. The average finishing time is now half of the first version. We must then again make a statistical analysis of the tacking operation because due to these changes this operation now maybe has an impact on the total production time.



For the second point it is important to study the impact of decreasing and increasing the mean. If we are sure that the current mean chosen is realistic, such an analysis provides very important information. We see that if we divide by two the finishing time – in increasing the number of workers – the total time will only decrease by 6%. The percentage is quite large but schedulers have to put in the scale the benefits drawn from such a reduction in comparison with the costs of supplementary workers. Depending on the case such investments could be useful or not. We can maybe conclude that we do not have to pay too much attention to the finishing and try rather to optimize other operations. On the other hand if the finishing time doubles, the impact on the total time is strong – 22%! The conclusion is that we still have to pay attention to this operation; not necessarily to reduce the time but to be sure that we will never increase it – for example in giving workers priority to other operations. In fact this phenomenon could be observed for each individual operation: below a certain value these operations have no real impact but above a critical value the impact becomes more and more crucial. The object of the statistical analysis is thus to determine which parameters are above their critical value!

In the following figure we see the same study but for the manual tacking time – not with the mechanized gripper; values are different in the simulation.



**Fig. 116: Impact of the Manual Tacking time on the Total Production time**

In this example we have only done one run with a multiplication factor but we see clearly that the impact on the total time is really slight. In this case it is even unnecessary to make a more precise analysis with several runs with factors. If we nonetheless do it we can

see for this example that the impact is effectively negligible. Of course if we raise continuously the mean time the impact on the total production time will start to become significant. In the first version of the simulation this manual tacking really had an influence. But now with the new methodology – the next kits are already tacked before the taking away of the previous one – the tacking is no longer a bottleneck within the system. This was of course the goal of this major change.

As has been shown, statistical analysis is a very important point when developing a model, once it is developed and calibrated.

## **3.3 Optimization of the Model**

### **3.3.1 Introduction**

We explained in previous chapters the clear interests of a simulation model and all the potential results that can be obtained. All these analyses provide information about the real workshop and where bottlenecks and problems might be. This is an important point for workshop schedulers, who can thus take adapted solutions to solve these problems. Nevertheless these solutions are not always obvious to find manually. Consequently it could be very precious if a tool is linked to the simulation model to automatically improve its productivity! In other words a very interesting possibility is to link the simulation model with an optimization tool!

What can be optimized in a simulation model? There are numerous answers to this question and depend strongly on the kind of simulation. We obviously want to optimize the parameters that can easily be changed in reality. In a production model an obvious solution to increase the productivity is to raise the number of machines in the workshop, or to increase the number of workers. But each workshop has budget limits and not all solutions are possible in practice. Consequently we do not have the possibility to optimize everything without considering the feasible limits.

In our simulation model we will try to optimize the production sequence. This sequence could have an important impact on the total production time. Indeed a situation to be avoided is waiting for the welding robots. This can be done if we have a good balance in the workload between each side of the workshop! To get this balance, one possibility is to change the production sequence. To do this optimization an appropriate algorithm has to be chosen. The

system is very complex and this point must be taken into account when choosing the optimizer.

What is the objective of the simulation and what are design variables? The goal is simple: to maximize the productivity of the workshop. As explained above three different solutions remain:

- To change the configuration of the workshop: to add a new production line, to add new machines, etc;
- To change available resource: sometimes resources could be quite flexible – for instance the number of workers can be changed.
- To change the production sequence in order to balance the workload.

The first solution is a strategic choice. This solution could be used only if the simulation is used several months before the production. This is a workshop optimisation but that kind of optimisation is done rarely – only before to build the workshop or if a big investment could be done. In this thesis this optimisation will not be treated.

The second solution is possible and interesting but more constraints must be known. Indeed cost factors must be involved: for instance we must know what the impacts on cost are if we hire more workers. On the other hand we must also know which delays are authorized for the production and the cost of these delays. With this knowledge an optimisation can be done to find the best compromise. Again this optimisation will not be treated here.

The third solution is an operational choice: only the production sequence is changed. Here the goal is to balance the workload to have a better use of the equipment. In other words we have a list of assemblies to build and we want to finish them in the shortest time. This is the optimization that we will do.

How to choose the good algorithm? The objective function is obviously the production time. The variable is the sequence to optimize. Notice that we also have some restrictions on the sequence: the assemblies can not be done whenever we want because the workshop depends on previous workshop.

The optimizer chosen is a genetic algorithm. Genetic algorithms are indeed solutions to use when we do not have the solution to a problem in a “classic” way. A “classic” way to solve a problem could be to have the equation that will give the solution. In this precise case the exact solution could generally be found after a known time. A typical problem is the

“Travelling Salesman Problem” which is a good case in which to use genetic algorithms. They are also often used when we have a function with many variables because the exploration of all the design space becomes prohibitive although a genetic algorithm can stay competitive. In the case of a sequence problem the attractions of that optimization is obvious due to the correspondence between sequence and chromosome. Furthermore genetic algorithms are in most of the cases used when we want to have a good solution but not necessarily to get the best solution. For our problem with such a huge number of solutions there is no hope of finding the best solution. But the goal is to increase the productivity of the workshop and a gain of 10 percent is still a good solution, even if the best one could offer a gain of 20 percent. Genetic algorithms could guarantee to obtain easily relatively good solutions!

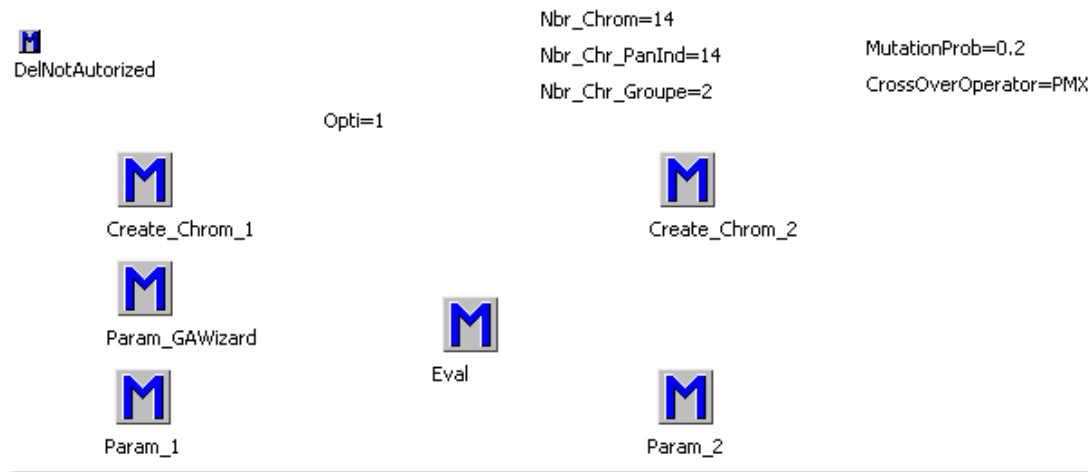
The section 2.3 gives a short reminder of genetic algorithms. The next section describes the way to link the algorithm to the simulation and results obtained.

### **3.3.2 Applications to the Workshop**

#### **3.3.2.1 Methodology**

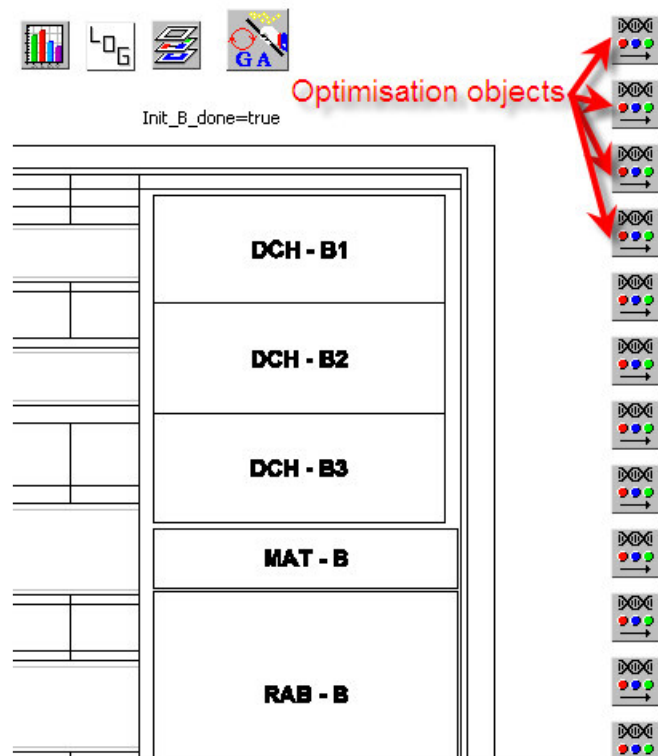
In order to optimize the workshop an efficient methodology is required to create chromosomes and individuals. A major difficulty in genetic algorithms is to code individuals of the population. The quality of the code drives the success of the algorithm. Historically binary coding was used very much at the beginning. Now real coding is widely used notably in applicative fields for the optimization of problems with real variables. The different objects and programming done to perform the optimization are now detailed.

The structure of the optimisation frame is the following one:



**Fig. 117: The "Optimisation" Frame**

The frame contains very few objects. Methods are used to create objects needed for the optimization. Unfortunately these objects cannot be created in this frame but must be put in the main frame of the model. In our case Optimisation objects must thus be put in the root frame. This is not very convenient for the visualisation and we create them to the right of the workshop – see Fig. 118. For more clarity it would be better to create them in a specific frame but because of software restrictions we have to put them into the root frame.



**Fig. 118: Optimisation objects in the root frame**

As we will see, the number of objects depends on the list of assemblies/kits that we have to optimize.

How are a population's chromosomes – and genes – created? Two different methods are implemented but they have the same philosophy. An individual is constituted of chromosomes and each chromosome has its own genes. During mutation operations the genes of a chromosome can be modified. In a cross-over generation to create new individuals, chromosomes are changed.

The first methodology is to create one chromosome for each sub-panel. A chromosome contains a sequence of genes: it will be the production sequence of each kit of the sub-panel. So a specific chromosome will differ for each individual by the sequence of genes that constitute it. New individuals will be created in interchanging chromosomes – thus in modifying orders of kits inside a sub-panel. In concrete terms it means that a child will have for each sub-panel exactly the same kits order as one of its parents. Except that mutation operations could change a part of the chromosome – a part of the sequence. Because the order of chromosomes is the same for each individual it means that the order of sub-panel will also be the same in the workshop. With that methodology we thus have the following constraints:

- Assemblies always belong to the same kit;
- Kits always belong to the same sub-panel (that is an inviolable constraint of the workshop);
- Order of kits inside the sub-panel will be optimized;
- **Order of sub-panels is fixed.**

The second methodology has exactly the same philosophy. The basis is the same but we add to each individual two chromosomes. The first chromosome contains the production sequence of the sub-panel of side A. The second one contains the sequence of side B. Consequently, in that case we have:

- Assemblies always belong to the same kit;
- Kits always belong to the same sub-panel (that is an inviolable constraint of the workshop);
- Order of kits inside the sub-panel will be optimized;
- **Order of sub-panels will be optimized.**

In the second case we strongly increase the degree of freedom of the optimizer: the solution space rise exponentially!

The results obtained will be given in the next chapter. Some theoretical details will also be given in the chapter because it is easier to explain the functioning of the algorithm on a practical case rather than purely theoretically.

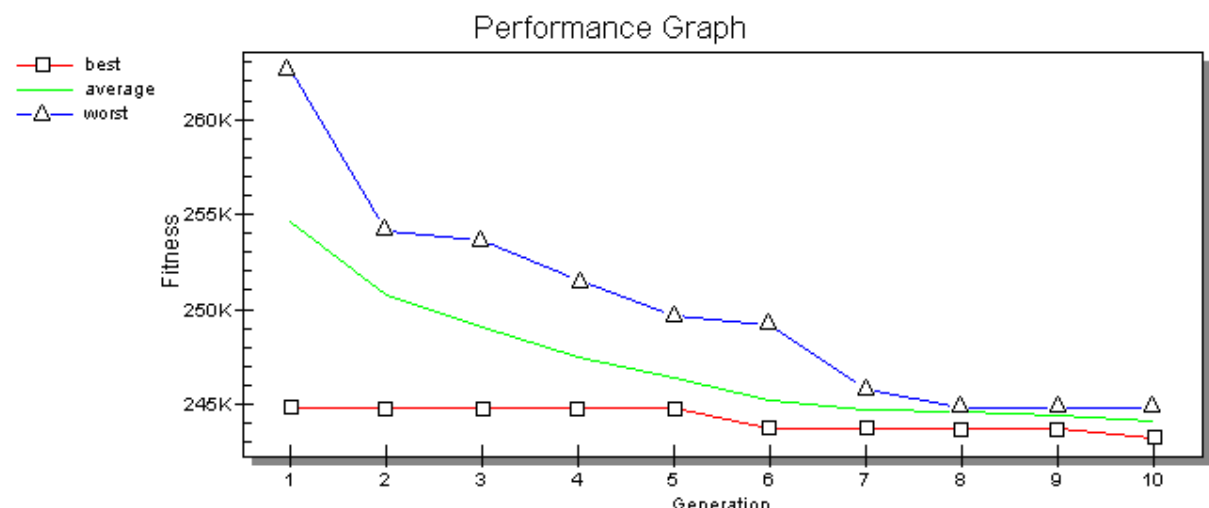
### 3.3.2.2 Results

In this section results of the benchmark indicated in section 3.2.6.2 are reported. Of course these results depend on the studied sequence: for a huge number of assemblies to be built with a high flexibility in the sequence, and gains will be greater than for a small sequence with little flexibility! It is why, again, we do not really pay too much attention to the value but on the possibilities of the optimization.

Different parameters have been tested:

- Choice of the selection;
- Choice of fitness (relative or absolute);
- Etc.

As was explained concerning the disadvantages of the GA, the difficulty is to define good values for these parameters and lots of experiments are required. We very often encountered a homogenization of the population after some generations. Fig. 119 shows that phenomenon: in red the evolution of the best individual of each population, in blue the worst individual and in green the average.



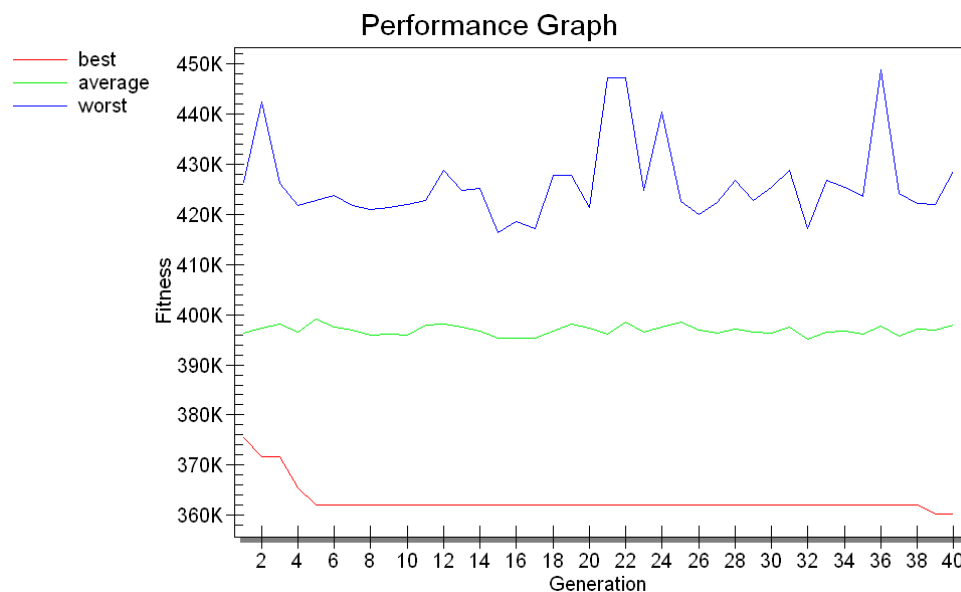
**Fig. 119: Homogenization of the population**

In this situation there is no interest in increasing the number of generations. A possible cause is that the number of individuals is too small or a mutation factor is not high enough.

On the other hand we cannot have a too high number of individuals because the evaluation of the fitness takes time. A compromise should then be found between the total CPU time available to solve the problem and the quality of the solution.

The optimization results obtained are good but there is a feeling that we are almost each time converging towards a local optimum and not a global one. This fact is in discrepancy with the main advantage of the genetic algorithm, in other words avoiding converging towards a minimum local. However an optimization gives each time better results than the best result obtained if the same number of sequences are generated randomly. On the one hand the optimization seems not to work as well as hoped but on the other hand the results are good. This paradox will be explained.

The convergence seems to be towards a minimum local. What are the reasons that let us think that when we change the parameters of the optimization – we saw that they were numerous – we get two tendencies of graphics results? The first tendency is a rapid homogenisation of the population – as in Fig. 119. If we select parameters to avoid this phenomenon we can get a situation such as in Fig. 120.



**Fig. 120: Non homogenisation of the population**

These results have been obtained with the following parameters:

- Size of generation: 100
- Number of Generations: 40
- Fitness reference: relative
- Parent Selection: Deterministic



- Cloning Best solution: Yes
- Offspring selection: Prob
- Cross Over: PMX
- Mutation
- Random Mutation: 0.1

The optimisation is the first one described i.e. without the group order optimization. What can be drawn from this graphic? The fitness is the total production time – of side A and of side B – in seconds. We see that the mean of each individual for the first generation is about 395000 seconds. In fact this value is the real mean of random sequences. The first important result is that at the end the best sequence takes 360181 seconds i.e. a gain of 9%! This gain is far from insignificant and tends to prove the efficiency of the genetic algorithm. On the other hand we see no improvement of the mean of the population! Now the functioning of the genetic algorithm implies that the best individuals are kept and consequently we must have an improvement of the mean. So in one way we have a good solution – much better than random sequences – but the general behaviour of the population seems not so good.

Why do we have this general behaviour? A big difficulty for the optimizer is that the effects of chromosomes are strongly related. It means that one chromosome could be very well conceived for an individual but could be completely badly conceived for another individual. For instance almost half the chromosomes are relative to the sequence of one side and the other half for the opposite side. So if a chromosome is “good” in one individual a change in another chromosome could completely modify the “quality” of the first one. This is one reason why it is so difficult to keep good sequences from one generation to another one. Naturally we can guess that we need to have a smaller mutation parameter for chromosomes related to the sub-panel’s order. But even with few mutations the chromosomes are too related together to allow the genetic algorithm to use its full efficiency!

Nevertheless the best solution obtained is quite satisfying. An explanation could be that we clone the best solution and this solution is at each generation used to be a parent to create new individuals. Even with the problem mentioned above we still have a probability to obtain a child better than the parent. If we have this case we clone it for the next generation so we still have an improvement of the best individual curve. But due to the non optimal efficiency of the genetic algorithm there is little chance that the best solution is created from “normal”

parent individuals. It is almost each time a new version of the best cloned solution. In other words, once we have an individual with a high potential, the genetic algorithm will improve it – by mutation, cross-over operation, etc. – but this could be a local optimum and the optimizer will have great difficulties to leave this local optimum behind.

The first optimization tests were done on a previous version of the PrePreFabrication workshop simulation model. In that version two important factors were different from the current version:

- There were only two working areas by cell – not four;
- We had the possibility to use the Half-Workshop sequence or the Half-Cell sequence.

These differences have an important influence on the optimization and reduce the problems explained. The consequences of the first point were that the chromosomes were less related even if this factor is not really predominant. The second point is more important: we could use a half-cell sequence. In that sequence kits were allocated in advance to a specific cell: we know that kit number 3 will be done on half-kit 3A, and straighter afterwards we will create on this half-cell kit number 7, etc. In practise we do not allocate a kit before the production in the real workshop. The main reason is that it is too difficult to maintain the sequence – due mainly to production hazards. Consequently, in the simulation in the new version we could only use the half-workshop sequence – the sequence is fixed but no allocation was made and when a working area is free we put on it the next available kit. From an optimization point of view the half-cell sequence was better because we imposed kit sequences on each working area and we knew that this sequence would be respected! Consequently the algorithm had the ability to keep the best sequences and to transmit them to the future generation. In a half-workshop sequence the problem is that the same initial sequence can lead to different production sequences in practice. Indeed once a cell is free we call the next available kit. But the production is stochastic and if we run the same simulation, working areas could become free in different orders and thus kits will not be done each time on the same cell although the sequence is the same! That point again makes the optimisation much more complex.

We have explained why the genetic algorithm works, but without 100% efficiency. However we can still obtain a sequence that gives an improvement of the production with only one click on the optimization button! For both optimizations we obtain similar gains

without a real advantage for the second one. The reason is probably that even if the solution space is greatly increased we strongly reinforce the phenomenon described here above. Consequently the optimizer does not really take the benefits from this increase of the solution space.

Gains depend strongly on input data. Indeed if we have a list of one hundred kits to optimize we will obtain better gains than if we have only twenty kits. But in that case the CPU-time to get the fitness value is about five times longer. For Fig. 120 we optimized 61 kits and the total duration was 11 hours<sup>16</sup>. Even if the computer used was not adequate for huge calculations we see that CPU time could become very large. The time of a simulation is directly proportional to the number of kits.

In order to use the genetic algorithm at its full efficiency we have tried another optimization than the two previously described: the optimization of the sequence of kits of only one side of the workshop! So, the sequence is fixed for side B, and we only optimize the first sequence of side A. Here the solution space is greatly reduced! But if we do not allow the optimization of the group sequence we eliminate the problem described in this chapter. In that case we have one chromosome for each group – sub-panel – of side A. If the sequence of genes in a chromosome is good we know that it will still be good if this chromosome is used in another individual! We can now use the maximum efficiency of the genetic algorithm. The problem is that we have fixed the sequence of one side so we also do not optimize perfectly the workshop.

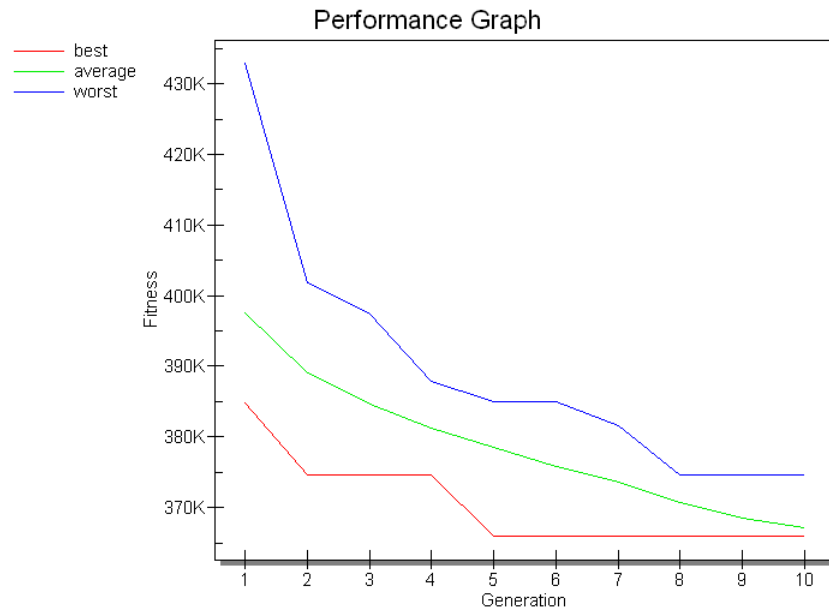
For this optimization we still have the homogenisation problem – see Fig. 121. To get the figures the following parameters have been used:

- Size of generation: 30;
- Number of Generations: 10;
- Fitness reference: absolute;
- Parent Selection: Random;
- Cloning Best solution: NO;
- Offspring selection: 1of4;

---

<sup>16</sup> On a Fujitsu Siemens Computer Mobile Intel(R) Pentium(R) M, processor 1.60GHz 591 MHz, 504 Mo of RAM

- Cross Over: PMX;
- Mutation;
- Random Mutation: 0.1



**Fig. 121: Results for a simplified optimization: only one sequence out of two is optimized**

In that case we do not have problem of the optimizer's general bad behaviour so we can better analyse the problem. We notice that the phenomenon of homogenisation occurred when we select the parameter "Offspring selection: 1of4". This parameter is a very important one because it determines which individuals of the population will be selected to create new individuals. To create new individuals the sequence is the following one:

- From the previous population we select individuals to use in the next generation – selection is done with the *Offspring selection* parameter;
- From these selected individuals we have to choose parents to create the new generation – the choice is done with the *Parent selection* parameter;
- Two parents will generate two children – by cross-over and mutation operations;
- The sequence can start again to create a new generation.

Note that the first population is created randomly. We will show the process for a simple example with a generation size of 10. First of all, 10 individuals are randomly created and each of them is evaluated by the simulation to get its fitness – see Fig. 122.

	Individual	Fitness
1	Individual1_1	380785.47
2	Individual1_2	391159.89
3	Individual1_3	381237.95
4	Individual1_4	392557.05
5	Individual1_5	393617.96
6	Individual1_6	400401.09
7	Individual1_7	394005.25
8	Individual1_8	381258.35
9	Individual1_9	389196.66
10	Individual1_10	406477.43

**Fig. 122: Random creation of the first generation**

For each individual we have thus a corresponding fitness. Each one of them has in that case seven chromosomes – because we have seven groups in the A side – as showed in the Fig. 123.

	Chromosome
1	Chromosome1
2	Chromosome2
3	Chromosome3
4	Chromosome4
5	Chromosome5
6	Chromosome6
7	Chromosome7

**Fig. 123: List of chromosomes for an individual**

Each chromosome contains a list of kits. The list is the same for each individual, the only change being the order of these kits. Fig. 124 shows some chromosomes of a specific individual. Each kit can be considered as a gene of a chromosome.

Chromosome 1			Chromosome 2			Chromosome 3		
	Define Set			Define Set			Define Set	
1		1099	1		1217	1		1445
2		1100	2		1214	2		1447
3		1101	3		1216	3		1444
			4		1215	4		1446
						5		1443

**Fig. 124: Examples of chromosomes**

These ten individuals will be the parents to create the new generation. Some individuals will be selected several times, others will never be. The selection depends on the Parent Selection parameter. In this case we have chosen *random* so we do not take into account the fitness to select parents. If we take *Deterministic*, parents with the highest fitness have more chance of being selected – proportional to the fitness value. Fig. 125 shows a *Family table* where we see for each line the parents selected and the two children created. Children have

the chromosomes of their parents – some are from parent 1, others from parent 2 – eventually modified by a mutation.

	Parent 1	Fitness	Parent 2	Fitness	Child 1	Child 2
1	Individual1_9	389196.66	Individual1_2	391159.89	Individual2_1	Individual2_2
2	Individual1_4	392557.05	Individual1_3	381237.95	Individual2_3	Individual2_4
3	Individual1_1	380785.47	Individual1_8	381258.35	Individual2_5	Individual2_6
4	Individual1_5	393617.96	Individual1_3	381237.95	Individual2_7	Individual2_8
5	Individual1_5	393617.96	Individual1_6	400401.09	Individual2_9	Individual2_10
6	Individual1_4	392557.05	Individual1_4	392557.05	Individual2_11	Individual2_12
7	Individual1_10	406477.43	Individual1_10	406477.43	Individual2_13	Individual2_14
8	Individual1_2	391159.89	Individual1_3	381237.95	Individual2_15	Individual2_16
9	Individual1_2	391159.89	Individual1_7	394005.25	Individual2_17	Individual2_18
10	Individual1_5	393617.96	Individual1_5	393617.96	Individual2_19	Individual2_20

**Fig. 125: Family table with parents and children individuals**

We now have a list of twenty children and each one of them is evaluated by a simulation run. Fitness values are given in Fig. 126.

	Individual	Fitness
1	Individual2_1	401826.68
2	Individual2_2	380517.11
3	Individual2_3	388100.95
4	Individual2_4	388042.77
5	Individual2_5	394951.28
6	Individual2_6	398099.87
7	Individual2_7	393190.41
8	Individual2_8	398045.29
9	Individual2_9	395668.39
10	Individual2_10	389848.51
11	Individual2_11	368213.83
12	Individual2_12	413781.14
13	Individual2_13	395024.59
14	Individual2_14	384527.68
15	Individual2_15	403003.63
16	Individual2_16	416108.98
17	Individual2_17	400070.68
18	Individual2_18	387511.93
19	Individual2_19	378158.65
20	Individual2_20	395078.71

**Fig. 126: Fitness values of all children**

We see that the best individual already has a better fitness than the parent's best. The total population of the generation 2 is thus composed of 20 parents – even if some are identical – and 20 children – see Fig. 125. Among these 40 individuals we have to select individuals that will be used to create the third generation. At this point we use the *Offspring Selection* parameter. This is probably the parameter that has the most important impact on results. We have already explained the parameter but as a reminder here are the four different values that it could take:

- Only use the two child solutions and select the solution with the best fitness value (*Iof2*);
- Use parent and child solutions and select the best solution (*Iof4*);

- Use parent and child solutions. It employs a stochastic selection. Selection probabilities are proportional to the quality of the solution (*Prob*);
- Select the individual taken for the new generation regardless of the fitness value (*Random*).

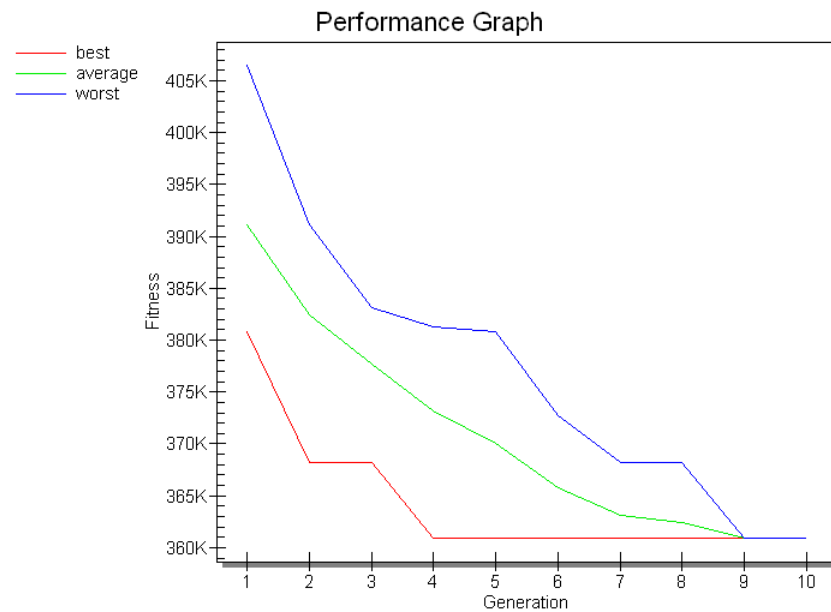
For each line of the Family table we will keep only one individual to get again the ten individuals used to create the next generation. In our example we have chosen the *Offspring selection Prob* that gives the selection given in Fig. 127.

	Individual	Fitness
1	Individual2_11	368213.83
2	Individual2_19	378158.65
3	Individual2_2	380517.11
4	Individual1_3	381237.95
5	Individual2_4	388042.77
6	Individual1_2	391159.89
7	Individual2_5	394951.28
8	Individual2_9	395668.39
9	Individual2_17	400070.68
10	Individual1_10	406477.43

**Fig. 127: Individuals selected to create the third generation**

We can see that we have individuals of the first and of the second generation. With an *Offspring selection 1of2* we would have obtained only individuals of the second generation, losing individuals of previous generations – except if we have selected the option *Cloned best solution*. Now the process can restart finally creating the next generations.

With this methodology we can easily see why we tend to get a homogenisation of the population, especially when we select an *Offspring selection 1of4*. In this case, for each line of the *Family table* we keep the best element. A characteristic of the process is that we can have the same parent several times – see Fig. 125. If this parent is really good it will be selected for the next generation and consequently we can have exactly the same individuals several times in the population. For instance if we have a very good element selected twice as a parent we have a strong probability of finding it twice in the list of the ten selected individuals. And we can thus have 4 (or even more) times this element chosen as a parent for the next generation and so on. We can rapidly get a completely homogeneous population! Fig. 128 shows an extreme case. Notice that we still have a final solution with a fitness of 360933 – which is a gain of 8.6%! This result has been obtained very rapidly because we had a population size of 10 elements only. So the genetic algorithm has still improved the solution.



**Fig. 128: Radical homogenisation of a population**

The optimizer could be greatly improved if we avoided selecting the same individuals several times for the next generation. Normally the homogenisation of the population occurs because the same chromosomes are copied in most individuals and not because of a simple copying and replication of individuals! Although we have access to lots of parameters – access to parametric methods, evaluation methods, etc – we do not have access to the entire algorithm. Consequently we cannot correct the algorithm that could have certainly been better designed. To minimize the effect we recommend thus avoiding cloning the best solution and choosing a *Parent Selection “Random”* rather than *“Deterministic”* when we select *Offspring Selection “1of4”*. With an *Offspring Selection “Prob”* or *“Random”* the same phenomenon could be observed but it is really less marked. We completely avoid the problem with *“1of2”*. But in that last case we lose the best individuals by eliminating the individuals of previous generations. That reduces strongly the potential interest of the optimizer.

One of our goals concerning the optimization was to optimize the kits’ creation! In other words kits are not conceived in advance but are created by the optimizer. Here again we strongly increase the solution spaces that allow us to hope to obtain better gains concerning the total production time. However it could only work if the genetic algorithm is working completely efficiently and we had seen it was not the case because of this chromosome interdependency problem. Furthermore this problem is still reinforced if we allow the tool to optimize the kit’s creation! If we want to automatically create kits we must act differently. A good methodology could be to start with a (double) list of assemblies with no kit allocation. First we make kits with a welding time estimated at about a time T1. We run the simulation



and find the best sequence to minimize the total production time. We keep the best solution. Then we start again by creating new kits with a welding time of around T2. We optimize and find the best solution. We do the same operation for different welding times. We can also do the same with half kits with T1, and half kits with T2 with a huge difference between T1 and T2. At the end we can observe what the trends are. Is it preferable to have kits that have approximately the same workload? If yes, is it preferable to have huge kits or smaller ones? Or is it better to have a mix of kits? This methodology can give solutions to these questions. This study is much more complex and requires lots of data to be validated. But the study has to be done only once and helps us to find the best way to build kits. Once they are built we still can do an optimization sequence as done in this thesis.

In conclusion we see that we can have benefits with the optimization – the total production time could be reduced by ten percent – but the gains are limited for two reasons:

- The problem is really unusual and in coding chromosomes it is very difficult to eliminate interactions between them: a good chromosome for one individual could be very bad for another one. We cannot by-pass this problem!
- The genetic algorithm used could probably be improved to be more efficient. We do not have access to all its structure and again cannot by-pass the problem!

Nevertheless the conception of this workshop is really untypical and we will rarely come across workshops that have the same problem as the first problem described!

### **3.4 Conclusion**

A complete model of the PrePreFabrication has been developed. The chosen workshop has many unusual characteristics that make the modelling more difficult:

- The sequence production is very unusual (not a classical production line);
- We encounter space allocation problems in choosing kits;
- Each product is different making production optimisation harder;

Of all the workshops of the shipyard, this one is probably the most difficult to model. We have shown in this chapter that the simulation is possible. The results that we can get are very varied and can really help the planner to improve the productivity of the workshop.

The results obtained can be ranked in three different categories.

First, we can simply have information about the general behaviour of the workshop. We can have access to average operation time, duration times for a sub-panel (or a simple piece) in the workshop, relative occupation of workers, etc. Of course all of these times can be obtained with measurements in the real workshop. But we cannot imagine that a worker will make a note each time he takes a piece, start his tacking, uses the mechanized gripper, takes a break and so on. Furthermore, here we can simulate lots of different production sequences and collect a lot of data in a shorter time than in reality. To have a better understanding of this general behaviour of the workshop we have developed a Gantt interface in Excel© and a PERT tool in Java© - see Chapter 4. This last tool is not only a new way to visualize results but is also a powerful tool to optimize the workshop in another way than with a genetic algorithm. This first analysis gives us important information about bottlenecks within the system.

Secondly we can make a deeper study with a statistical analysis of the workshop. By modifying their value we can observe the impact of each individual operation on the total production time. This second analysis gives us important information about how to modify bottlenecks within the system!

Thirdly we can do an optimization of the workshop. For a given configuration of the workshop we showed that we can increase the productivity only with a different production assembly sequence. The workshop was not really designed in advance for an optimization but we have shown that the link between a complex simulation model and an optimization algorithm is possible. As explained in this chapter it is very difficult – almost impossible – to be sure of getting the best solution. But the algorithm used gives better solutions than a random approach and the increase of productivity could still be interesting. A possible perspective could be to improve the quality of the genetic algorithm itself.

In conclusion, the simulation of a complex workshop can provide very interesting information for the scheduler that no other tool could offer.

# Chapter 4

## Development of a Specific PERT Tool and Application to the PrePreFabrication Workshop

### 4.1 PERT: Theoretical background

#### 4.1.1 Introduction

The *Program Evaluation and Review Technique* (PERT) is a network model that allows randomness in activity completion times. PERT was developed in the late 1950s for the U.S. Navy's Polaris project, which had thousands of contractors. It has the potential to reduce both the time and cost required to complete a project. In that particular case the time scale of the project was decreased from 7 years to only 2 years solely with the use of the PERT diagram – see <http://www.netmba.com/operations/project/pert>.

This method is one of the most rigorous to be used but also one of the most powerful. Thanks to such a method it has been possible to do the planning of very a huge project – an example is the manufacturing of huge ship such as « Queen Mary 2 ».

Firstly we will describe what a PERT network is and its terminology. Secondly we will outline the tool developed in the framework of this thesis in order to easily represent a project with a PERT network. The main characteristics of the tool are a high level of easiness for the user, a low CPU-Time consumption and the possibility of optimizing a project to minimize the production cost. The theoretical developments in this chapter are not new – no theoretical developments have been made – but the tool developed in this thesis has the main advantage of being very flexible and can thus be easily linked with our simulation model. Furthermore, existing PERT tools rarely contain an optimization option as in this tool – see further on in this chapter for more information about the optimization. Finally this chapter contains the link between the PERT tool developed and the virtual simulation model developed in Chapter 3. The PERT tool will be automatically created for a given sequence of production and will provide interesting supplementary information.

The possibility to represent a simulation of the building of a list of kits with a PERT diagram is really useful. Generally we use PERT diagrams to model projects at a strategic or operational level. Here the objective is to use it at a tactic level! We can run several

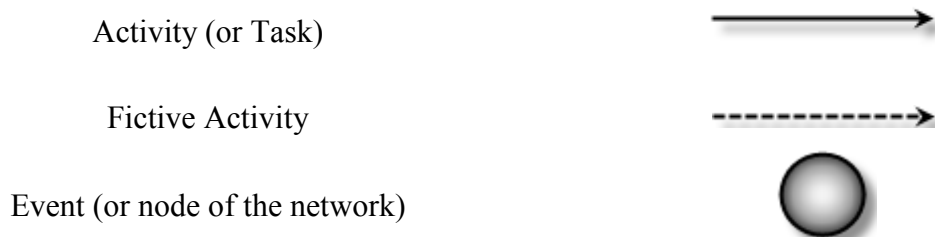
simulations – with different parameters – and for each of them we will have a PERT diagram. This view is – for this workshop – one of the best ways to represent at a glance what will happen in the workshop. As it is explained in the chapter we will directly see which operations is critical and which can be in late without rise the total production time. This is very important for that kind of workshop. Indeed resources – workers for instance – are the same for different working areas. If two – or more – of these working areas require these resources at the same time, how do we have to allocate these resources? It is impossible to have the answer without a simulation and its PERT diagram. Because the PERT will tell directly: this work is not critical at all and then it is better to allocate workers on the other job! Tactical choices can be done very efficiently. That kind of information can be drawn directly from a simulation run – without the PERT diagram – but a deeper analysis is require in order to get the same conclusion. View of the reality is much more highlighted with this PERT, above all if the PERT is automatically created.

#### 4.1.2 The Network Diagram

In a project, an *Activity* is a task that must be performed and an *Event* (or node) is a milestone marking the completion of one or several activities. An activity consumes time, it requires resources (such as labour, materials, space, machinery), and it can be understood as representing the time, effort, and resources required to move from one event to another. Before an activity can begin, all of its preceding activities must be completed.

Project network models represent activities and milestones by arcs and nodes. PERT originally was an activity *on arc* network, in which the activities are represented on the lines and milestones on the nodes. A node is a point that marks the start or completion of one or more tasks. It consumes no time, and uses no resources. It marks the completion of one or more tasks, and can not be “reached” until all of the activities leading to that event have been completed.

Schematically we use the following convention:



Fictive activities are used to model interdependence between nodes – and not to represent a real task.

In the PERT method, for each node we calculate two values:

- The *early start time*: this is the earliest date at which we can start the following activities of this node in taking into account time needed to realize predecessors activities;
- The *late start time*: this is the latest date at which we can start the following activities of this node if we don't want to make the project late.

The PERT chart may have multiple pages with many sub-tasks. The following figure is a very simple example of a PERT diagram with four nodes and five activities:

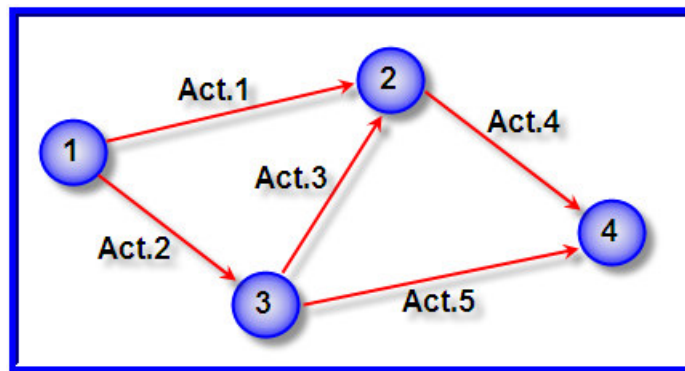


Fig. 129 : Example of a simple PERT network

To determine the *early (earliest) start time* of a task, we have to follow the diagram **from left to right** and to calculate the longest way from the start of the project until this task. If there is more than one way, we do the calculation for each one and we choose the **latest** date.

To evaluate the *late (latest) start time* of a task, we have to follow the diagram **from right to left** and to subtract from the late start time of the next task the duration of the activity that we calculate. If there are different ways, we do the same calculation for each and choose the **earliest** date.

The difference between the early start time and the late start time of an activity is called the total float (margin). We can also define the free float by the amount of time that a task in a project network can be delayed without causing a delay on a part of the network – not on the final total time.

An activity from A to B is called *critical* if the difference between the late start time of B and the early start time of A is equal to the duration of the activity. The set of critical activities is the critical path. Any delay T on one activity of the critical path will delay the total project time by at least T. The critical path determines the total calendar time required for the project.

### 4.1.3 Terminology

The main terms have been described before but sometimes we also used the following terminology – ref <http://www.netmba.com/operations/project/pert>:

- *A predecessor event*: an event that immediately precedes some other event without any other events intervening. It may be the consequence of more than one activity.
- *A successor event*: an event that immediately follows some other event without any other events intervening. It may be the consequence of more than one activity.
- *Optimistic time (O)*: the minimum possible time required to accomplish a task, assuming everything proceeds better than is normally expected.
- *Pessimistic time (P)*: the maximum possible time required to accomplish a task, assuming everything goes wrong (but excluding major catastrophes).
- *Most likely time (M)*: the best estimate of the time required to accomplish a task, assuming everything proceeds as normal.
- *Expected time (T<sub>E</sub>)*: the best estimate of the time required to accomplish a task, assuming everything proceeds as normal (the implication being that the expected time is the average time the task would require if the task were repeated on a number of occasions over an extended period of time).

$$T_E = (O + 4M + P) \div 6$$

- *Lead time*: the time by which a *predecessor event* must be completed in order to allow sufficient time for the activities that must elapse before a specific PERT event manages to be completed.
- *Lag time*: the earliest time by which a *successor event* can follow a specific PERT event.
- *Slack*: the **slack** of an event is a measure of the excess time and resources available in achieving this event. **Positive slack (+)** would indicate *ahead of schedule*; **negative slack** would indicate *behind schedule*; and **zero slack** would indicate *on schedule*.

- *Fast tracking*: performing more critical activities in parallel
- *Crashing critical path*: Shortening duration of critical activities
- *Float* or *Slack* is the amount of time that a task in a project network can be delayed without causing a delay - Subsequent tasks – (**free float**) or Project Completion – (**total float**)

#### 4.1.4 Benefits of PERT

PERT is useful because it provides the following information:

- Expected project completion time.
- Probability of completion before a specified date.
- The critical path activities that directly impact on the completion time.
- The activities that have slack time and that can lend resources to critical path activities.
- Activity start and end dates.

#### 4.1.5 Limitations

PERT's weaknesses are the followings [Koteswara et Al., 2008]:

- The activity time estimates are somewhat subjective and depend on judgement. In cases where there is little experience in performing an activity, the numbers may be only a guess. In other cases, if the person or group performing the activity estimates the time there may be bias in the estimate.
- Even if the activity times are well-estimated, PERT assumes a beta distribution for these time estimates, but the actual distribution may be different.
- Even if the beta distribution assumption makes sense, PERT assumes that the likely distribution of the project completion time is the same as that of the critical path. Because other paths can become the critical path if their associated activities are delayed, PERT consistently underestimates the expected project completion time.

The underestimation of the project completion time due to alternate paths becoming critical is perhaps the most serious of these issues. To overcome this limitation, Monte Carlo simulations can be performed on the network to eliminate this optimistic bias in the expected project completion time – ref. <http://www.netmba.com/operations/project/pert>.

## 4.2 Tool developed

### 4.2.1 Introduction

Our goal is to have a powerful and very flexible tool to use to improve the productivity of the PrePreFabrication workshop. The idea was not to develop a specific tool, but a more general one and, after that, using it in our specific case.

The tool will not only be a simple representation of a project's activities. It will automatically assess different information that will really help the planner to improve the scheduling or improve the workshop itself. It will be possible to use the tool at **three** different levels:

#### *1) Simple Study of the project*

The tool will be very user-friendly to easily model a workshop or a process. In other words we will be able to quickly add/remove activities, fictive activities, nodes and easily change the attributes of each object. The only parameters to be introduced are the time (duration) of **each** activity.

The software will automatically evaluate for each node the earliest start time and the latest start time. The critical path will be detected automatically and be observable by the user. For each node, we will also be able to see the critical path from the start of the project until the node selected!

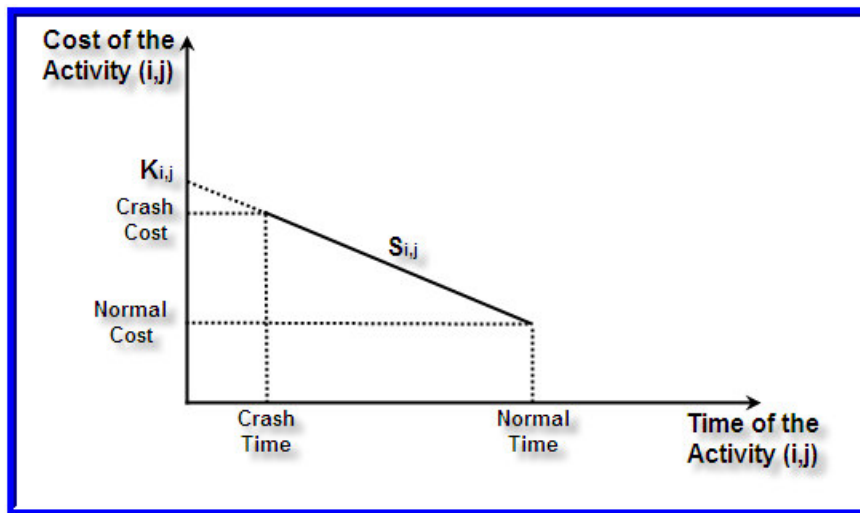
The total margin will also be directly assessed and can improve the planners' knowledge of the project studied.

#### *2) Optimization of the project (minimization of cost)*

For each activity, there is the possibility to give an interval of duration to realize the activity itself. The user can introduce a normal time and a crash time – sooner than the normal time. To each duration of an activity we link a cost. Doing an activity in  $x$  hours will cost  $y$  Euros (or man hours, months or other resources) and if we decrease the time the cost will obviously increase because doing the same activity in a shorter time requires more resources. In our software it is assumed that the cost is directly proportional to the time.

In short, each activity  $(i, j)$  is described by a crash time, a crash cost, a normal time and a normal cost, where “ $i$ ” is the number of the start node and “ $j$ ” the number of the end node – see Fig. 130.





**Fig. 130: Characteristics of an activity**

If we want to model an activity whose time is not directly linked to the cost we have to split this activity into smallest ones to approach the curve. The curve is replaced by a piecewise linear function.

Generally the approximation by a straight line is enough and we will avoid doing the approximation explained before for two reasons: we rarely have this curve – it requires detailed information about the activity – and a high increase of activities can decrease the speed of the calculation of the software – even if in practice we can model projects of more than hundred activities without having a too high calculation time.

If the user fixes a total time to realize the project, we will have different possibilities to change activities between their limits to obtain the time imposed. But each possibility will give a different total cost for the project. This total cost is the sum of the cost of each activity.

For a given total time of the project, the software will calculate the time of each activity in order to minimize the total cost of the project.

The total time fixed should of course be high enough to give a feasible solution, but the tool will immediately calculate and indicate the minimum time required.

As a conclusion, if the duration of each activity is not fixed but can change – because the resources can be increased or decreased – the tool will straightaway optimize resources and minimize the total cost! This level is thus particularly interesting.

### *3) Parametric study to optimize Total Time/Cost of the project*

Level three is only an improvement of level two: we can do a parametric study on the total time of the project to see its influence on the total cost! As a result we will thus have a graph to show this link between these two factors.

Practically it means that we can see if it worth increasing the total time of the project, or to see if a decrease of this total time will not have a too excessive impact on the cost.

Like the minimum total time, the maximum total time of the project is automatically calculated by the software. The software will not simply do multiple calculations of level 2 with different increase of T – for example by a delta given by the user. The software will first evaluate the cost for a minimum total time and take the results of this optimization to calculate faster the network with other times. In fact it will calculate if there is any change in the critical path when we increase the total time. If there is no change, the total cost increases exactly like the cost of the activity on the critical path that increases the cost the least. Consequently, the results of the parametric study will only be some linked Time/Cost characteristics of the network.

The parametric study is done in such a way that the total calculation time is almost the same as for only one optimization of a total time.

For the workshop studied, using only the first level of the software we will obtain very interesting results. For instance we will see straightaway if the welding robot is really the workshop’s bottleneck or not. Joined to an optimizer in order to try different sequences of assemblies, the tool is already very powerful in using only the first level!

**4.2.2 Theoretical approach**

**4.2.2.1 Introduction**

Our unknowns are the times of each activity  $x_{ij}$  (= design variables). Let’s introduce now the variable  $y_i$  for each event “i”:  $y_i$  is the earliest time that activities can start following this event. For instance,  $y_1$  will of course be zero and  $y_4$  (see Fig. 129) the total time of the project (a time that we will now call “T”). An activity obviously has the following constraints to respect:

$$\left\{ \begin{array}{l} x_{ij} \geq \text{crash time} \\ x_{ij} \leq \text{normal time} \\ y_i + x_{ij} \geq y_j \end{array} \right.$$

We can also calculate very easily the total cost which is given by:

$$\text{Total Cost} = \sum K_{ij} + x_{ij} * S_{ij}$$

where the sum is done on all activities and  $S_{ij}$  is the slope of the activity  $(i, j)$  (see Fig. 130). To take into account the fact that the total time of the process ( $T$ ) is fixed, we have to add another constraint:

$$y_{\text{end}} \leq T$$

where “end” is the index of the last event.

For computational reasons, it is better to replace  $x_{ij}$  by  $x'_{ij} = x_{ij} - \text{crash time}$ . To keep things easy we will not write the symbol ‘ in the following part of this report, but keep in mind that it is not the true variable design. The total cost is the objective function to be minimised and we can delete all  $K_{ij}$  for the optimization and add them at the end of the process. With all these points, the optimization problem can now be written:

$$\begin{aligned} & \text{Max } \sum -x_{ij} * S_{ij} \\ \text{Subject to } & \left\{ \begin{array}{l} \text{Act.}_1 \left\{ \begin{array}{l} x_{ij} \leq \text{normal time} \\ y_i + x_{ij} - y_j \geq \text{crash time} \end{array} \right. \\ \dots \\ \text{Act.}_{\text{num\_ac}} \left\{ \begin{array}{l} x_{ij} \leq \text{normal time} \\ y_i + x_{ij} - y_j \geq \text{crash time} \end{array} \right. \\ \\ y_{\text{end}} \leq T \end{array} \right. \end{aligned}$$

with  $x_{ij} \geq 0$  for each activity;  $y_i \geq 0$  for each node.

#### 4.2.2.2 Revised Simplex Method

This problem is solved by using the simplex method. The simplex method is a general procedure for solving linear programming problems. It has proved to be a remarkably

efficient method that is used to solve huge problems on today's computers. Apart from its use on tiny problems, this method is always executed on a computer. Extensions and variations of the simplex method are also used to perform post optimality analysis (including sensitivity analysis) on the model (Hillier *et. Al*, 2001). The theory of the Simplex Method will not be described here (see references for more details) but we will explain the Revised Simplex Method which is the procedure used in this program.

This method was designed to accomplish exactly the same things as the original simplex method, but in a way that is more efficient for execution on a computer. Thus, it is a streamlined version of the original procedure. A typical linear programming problem can always be written by using matrices. The standard form is:

$$\text{Maximize } Z = \mathbf{c} \mathbf{x}$$

Subject to

$$\mathbf{A}\mathbf{x} < \mathbf{b} \text{ and } \mathbf{x} > \mathbf{0}$$

where  $\mathbf{c}$  is the row vector

$$\mathbf{c} = [c_1, c_2, \dots, c_n]$$

$\mathbf{x}$ ,  $\mathbf{b}$ , and  $\mathbf{0}$  are the column vectors such that

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \dots \\ b_n \end{bmatrix}, \quad \mathbf{0} = \begin{bmatrix} 0 \\ \dots \\ 0 \end{bmatrix}$$

and  $\mathbf{A}$  is the matrix

$$\mathbf{A} = \begin{bmatrix} A_{11} & \dots & A_{1n} \\ \dots & \dots & \dots \\ A_{m1} & \dots & A_{mn} \end{bmatrix}$$

To obtain the augmented form of the problem, introduce the column vector of slack variables

$$\mathbf{x}_s = \begin{pmatrix} x_{n+1} \\ \dots \\ x_{n+m} \end{pmatrix}$$

so that the constraints become

$$[\mathbf{A}, \mathbf{I}] \begin{pmatrix} \mathbf{x} \\ \mathbf{x}_s \end{pmatrix} = \mathbf{b} \quad \text{and} \quad \begin{pmatrix} \mathbf{x} \\ \mathbf{x}_s \end{pmatrix} > \mathbf{0}$$

where  $\mathbf{I}$  is the  $m \times m$  identity matrix, and the null vector  $\mathbf{0}$  now has  $n + m$  elements.

You can see next an example of an augmented form (in fact, it is simply an equivalent model without the inequality constraint):

*Original form of the Model*

$$\text{Maximize } Z = 3x_1 + 5x_2$$

Subject to

$$(1) \quad x_1 < 4$$

$$(2) \quad 2x_2 < 12$$

$$(3) \quad 3x_1 + 2x_2 < 18$$

And

$$x_1 > 0, x_2 > 0$$

*Augmented Form of the Model*

$$\text{Maximize } Z = 3x_1 + 5x_2$$

Subject to

$$(1) \quad x_1 + x_3 = 4$$

$$(2) \quad 2x_2 + x_4 = 12$$

$$(3) \quad 3x_1 + 2x_2 + x_5 = 18$$

And

$$x_j > 0, \text{ for } j=1, 2, 3, 4, 5.$$

Although both forms of the model represent exactly the same problem, the new form is much more convenient for algebraic manipulation. We call this the *augmented form* of the problem because the original form has been augmented by some supplementary variables needed to apply the simplex method. An augmented solution is a solution for the original variables (the decision variables) that has been augmented by the corresponding values of the slack variables. For example, augmenting the solution (3, 2) in the example yields the augmented solution (3, 2, 1, 8, 5) because the corresponding values of the slack variable are  $x_3 = 1$ ,  $x_4 = 8$  and  $x_5 = 5$ .

In a linear problem, each constraint boundary is a line that forms the boundary of what is permitted by the corresponding constraint. The points of intersection are the corner-point solutions of the problem. The points that lie on the corners of the feasible region are the

corner-point feasible solutions (CPF solutions). A basic feasible (BF) solution is an augmented CPF solution. The only difference between basic solutions and corner-point solutions (or between BF solutions and CPF solutions) is whether the values of the slack variables are included. For any basic solutions, the corresponding corner-point solution is obtained simply by deleting the slack variables.

One of the key features of the revised simplex method involves the way in which it solves each new BF solution after identifying its basic and non basic variables. Given these variables, the resulting basic solution of the  $m$  equations

$$[\mathbf{A}, \mathbf{I}] \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_s \end{bmatrix} = \mathbf{b}$$

in which the  $n$  nonbasic variables from the  $n + m$  elements of

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{x}_s \end{bmatrix}$$

are set equal to zero. Eliminating these  $n$  variables by equating them to zero leaves a set of  $m$  equations in  $m$  unknowns (the *basic variables*). This set of equations can be denoted by

$$\mathbf{B} \mathbf{x}_b = \mathbf{b},$$

where the vector of **basic variables**

$$\mathbf{x}_b = \begin{bmatrix} x_{b1} \\ \dots \\ x_{bm} \end{bmatrix}$$

is obtained by eliminating the non basic variables from

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{x}_s \end{bmatrix}$$

and the basis **matrix**

$$\mathbf{B} = \begin{pmatrix} \mathbf{B}_{11} & \dots & \mathbf{B}_{1m} \\ \dots & \dots & \dots \\ \mathbf{B}_{m1} & \dots & \mathbf{B}_{mm} \end{pmatrix}$$

is obtained by eliminating the columns corresponding to coefficients of non basic variables from  $[\mathbf{A}, \mathbf{I}]$ . (In addition, the elements of  $\mathbf{x}_b$  and, therefore, the columns of  $\mathbf{B}$  may be placed in a different order while the simplex method is executed).

The simplex method introduces only basic variables so that  $\mathbf{B}$  is *non singular*, so that  $\mathbf{B}^{-1}$  will always exist. Therefore, to solve  $\mathbf{B}\mathbf{x}_b = \mathbf{b}$ , both sides are premultiplied by  $\mathbf{B}^{-1}$ :

$$\mathbf{B}^{-1} \mathbf{B} \mathbf{x}_b = \mathbf{B}^{-1} \mathbf{b}.$$

Since  $\mathbf{B}^{-1} \mathbf{B} = \mathbf{I}$ , the desired solution for the basic variables is

$$\mathbf{x}_b = \mathbf{B}^{-1} \mathbf{b}.$$

Let  $\mathbf{c}_b$  be the vector whose elements are the objective function coefficients (including zeros for slack variables) for the corresponding elements of  $\mathbf{x}_b$ . The value of the objective function for this basic solution is then

$$Z = \mathbf{c}_b \mathbf{x}_b = \mathbf{c}_b \mathbf{B}^{-1} \mathbf{b}.$$

For the original set of equations, the matrix form is

$$\begin{pmatrix} 1 & -\mathbf{c} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \mathbf{I} \end{pmatrix} \begin{pmatrix} Z \\ \mathbf{x} \\ \mathbf{x}_s \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{b} \end{pmatrix}$$

The algebraic operations performed by the simplex method (multiply an equation by a constant and add a multiple of one equation to another equation) are expressed in matrix form by pre-multiplying both sides of the original set of equations by the appropriate matrix. In particular, after any iterations,  $\mathbf{x}_b = \mathbf{B}^{-1} \mathbf{b}$  and  $Z = \mathbf{c}_b \mathbf{B}^{-1} \mathbf{b}$ , the right-hand sides of the new set of equations have become

$$Z = \begin{pmatrix} 1 & \mathbf{c}_b \mathbf{B}^{-1} \\ \mathbf{0} & \mathbf{B}^{-1} \end{pmatrix} \begin{pmatrix} 0 \\ \mathbf{b} \end{pmatrix} = \begin{pmatrix} \mathbf{c}_b \mathbf{B}^{-1} \mathbf{b} \\ \mathbf{B}^{-1} \mathbf{b} \end{pmatrix}$$

Because we perform the same series of algebraic operations on both sides of the original set of operations, we use this same matrix that multiplies the original right-hand side to premultiply the original left-hand side. Consequently, since

$$\begin{pmatrix} 1 & \mathbf{c}_b \mathbf{B}^{-1} \\ \mathbf{0} & \mathbf{B}^{-1} \end{pmatrix} \begin{pmatrix} 1 & -\mathbf{c} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} & \mathbf{I} \end{pmatrix} = \begin{pmatrix} 1 & \mathbf{c}_b \mathbf{B}^{-1} \mathbf{A} - \mathbf{c} & \mathbf{c}_b \mathbf{B}^{-1} \\ \mathbf{0} & \mathbf{B}^{-1} \mathbf{A} & \mathbf{B}^{-1} \end{pmatrix},$$

the desired matrix form of the *set of equations after any iteration* is

$$\begin{pmatrix} 1 & \mathbf{c}_b \mathbf{B}^{-1} \mathbf{A} - \mathbf{c} & \mathbf{c}_b \mathbf{B}^{-1} \\ \mathbf{0} & \mathbf{B}^{-1} \mathbf{A} & \mathbf{B}^{-1} \end{pmatrix} \begin{pmatrix} Z \\ \mathbf{x} \\ \mathbf{x}_s \end{pmatrix} = \begin{pmatrix} \mathbf{c}_b \mathbf{B}^{-1} \mathbf{b} \\ \mathbf{B}^{-1} \mathbf{b} \end{pmatrix}$$

The elements of these matrices are very important and used in the optimization process. A summary of the Revised Simplex Method is:

1. *Initialization:*

Determination of the initial BF solution: slack variables will be the basic variables.

2. *Iteration:*

- Step 1: Determine the entering basic variable. For each non basic variable, we calculate its coefficient in the expression of  $Z$  and the variable with the higher value (and strictly positive) will be the entering basic variable

- Step 2: Determine the leaving basic variable. For each equation where the coefficient of the entering basic variable is strictly positive ( $> 0$ ), we have to calculate the ratio of the right-hand side to the coefficient of the entering basic variable. The basic variable in the equation with the minimum ratio is the entering basic variable.

- Step 3: Determine the new BF solution: Derive  $\mathbf{B}^{-1}$  and set  $\mathbf{x}_b = \mathbf{B}^{-1} \mathbf{b}$ .

3. *Optimality test:*



It is directly linked to step 1. If no entering basic variable is found, the optimum is reached.

For a parametric study, rather than to start a completely new optimization when we change the parameter, it is possible to use the results of the last optimization in order to find the new optimum quickly.

#### 4.2.2.3 Adaptation of the method to our problem

To use this revised simplex method, some modifications must be made to our model. This method doesn't accept inequality and thus slack variables and artificial variables have to be added. Our model, described in the introduction to this chapter, must be modified.

The first constraint of each activity can be modified as following:

$$x_{ij} + x_{ij}'' = \text{normal time}$$

where  $x_{ij}''$  is a slack variable with  $x_{ij}'' \geq 0$ .

The second constraint of each activity becomes:

$$-y_i - x_{ij} + y_j \leq -\text{crash time}$$

$$\rightarrow -y_i - x_{ij} + y_j - x_{ij}''' = \text{crash time}$$

where  $x_{ij}'''$  is a surplus variable with  $x_{ij}''' \geq 0$ .

Unfortunately, the simplex method requires in each constraint a slack variable with a unitary coefficient in each constraint. Thus, the constraint should finally be:

$$-y_i - x_{ij} + y_j - x_{ij}''' + \bar{x}_{ij} = \text{crash time}$$

where  $\bar{x}_{ij}$  is an artificial variable with  $\bar{x}_{ij} \geq 0$ . In order to force this variable to tend towards zero, we have to subtract it from the objective function:

$$\text{Objective function} = \sum (-x_{ij} * S_{ij} - M\bar{x}_{ij})$$

where M is a positive constant with a high value. Because we want to maximize this objective function,  $\bar{x}_{ij}$  will obviously tends towards zero. This variable is in the objective function and thus is not really a slack variable (as defined in the simplex method), but its role is the same and will now be considered as a slack variable.

For the last constraint, we simply add a slack variable  $y'$ :

$$y_{\text{end}} + y' = T$$

with  $y' \geq 0$ .

Now the problem (which is the dual problem) can be solved by the simplex method and its new formulation is:

$$\text{Max } \sum (-x_{ij} * S_{ij} - M \bar{x}_{ij})$$

Subject to

$$\left\{ \begin{array}{l} \text{Act.}_1 \left\{ \begin{array}{l} x_{ij} + x_{ij}'' = \text{normal time} \\ -y_i - x_{ij} + y_j - x_{ij}''' + \bar{x}_{ij} = \text{crash time} \end{array} \right. \\ \dots \\ \text{Act.}_{\text{num\_act}} \left\{ \begin{array}{l} x_{ij} + x_{ij}'' = \text{normal time} \\ -y_i - x_{ij} + y_j - x_{ij}''' + \bar{x}_{ij} = \text{crash time} \end{array} \right. \\ y_{\text{end}} + y' = T \end{array} \right.$$

with  $x_{ij} \geq 0$ ,  $x_{ij}'' \geq 0$ ,  $x_{ij}''' \geq 0$  and  $\bar{x}_{ij} \geq 0$  for each activity;  $y_i \geq 0$  for each node;  $y' \geq 0$ .

The number  $n$  of design variables (without slack variables) is

$$n = 2 * \text{num\_act} + \text{num\_event}$$

The number  $m$  of constraints (which is also the number of slack variables) is

$$m = 2 * \text{num\_act} + 1$$

The system can now be solved by using the revised simplex method previously described.

### 4.2.3 Development of the tool

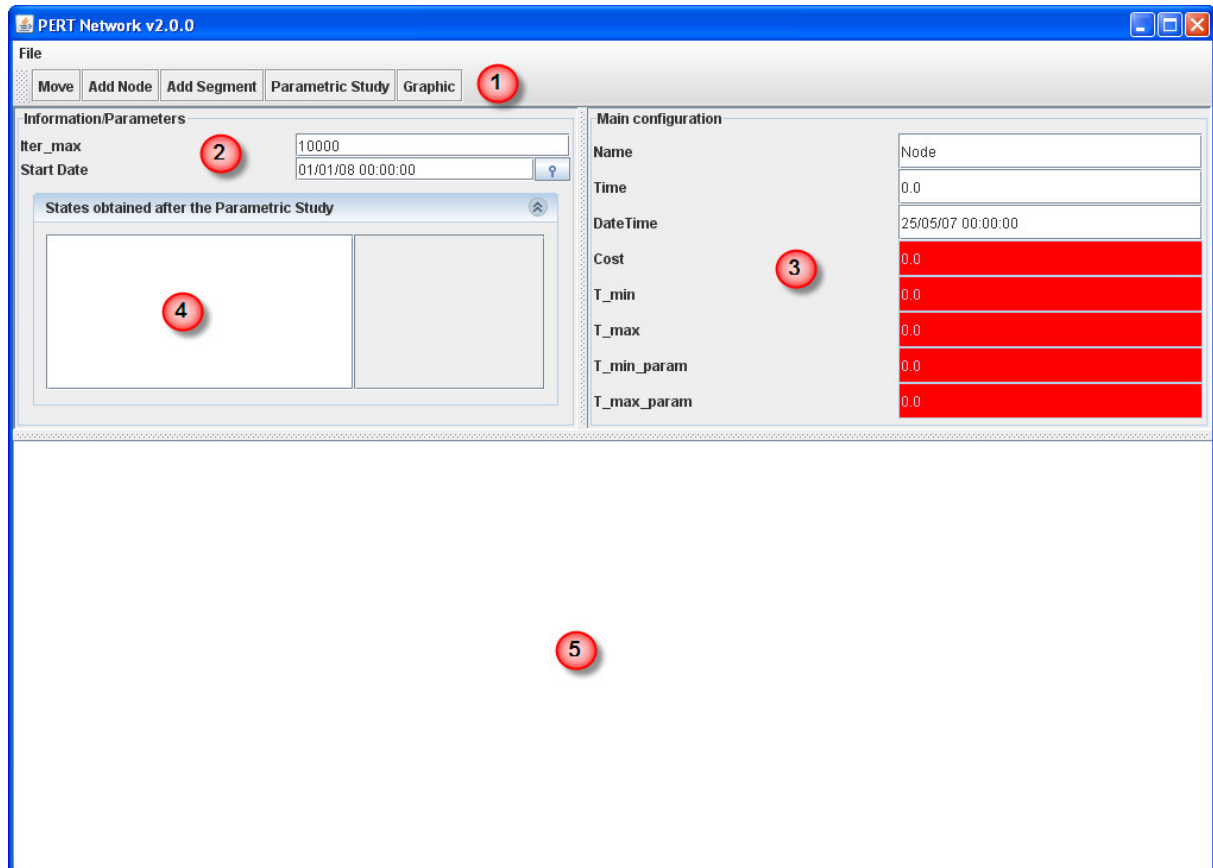
The software is made of two different parts:

- The optimization code ;
- The interface.

In the first version, the optimization code has been developed in C++ and the interface in *Visual Basic Applications* – Excel. In order to gain in flexibility it has been chosen to

rework the interface and to develop a new one in Java. For reasons of convenience the optimization has also been translated into *Java*.

The optimization code is only the computerized version of developments explained before. In this chapter we will present the interface and the way to use the software.



**Fig. 131: Interface of the PERT Network Software**

- ① : buttons to create the model (add nodes/activities), do the *Parametric Study* and obtained graphical results;
- ② : Iter\_max : choice of maximum number of iterations for the optimization;  
Start Date : date of the beginning of the project;
- ③ : gives different information for the selected node/activity (name, duration, earliest start date, latest start date, etc.);
- ④ : gives all the different states (values of each activity) of the network for each characteristic total time. Clicking on a state put all values in the model.
- ⑤ : area to draw the PERT network. Classical operations are available (selection, pan, zoom, etc.).

On the following figure we have a small network done with the interface. The user has only to:

- Add nodes (events) and link them by activities;
- Enter duration of each activity;

- Enter limit of duration (crash time and normal time);
- Enter corresponding cost (crash cost and normal cost);
- Enter names of events and activities (*facultative*).

Points 3 and 4 are only necessary if we want to do an optimization.

In Fig. 132 we see a small example of a network with 6 nodes and 8 activities. The attributes of each activity have been encoded and the critical path is automatically drawn in red.

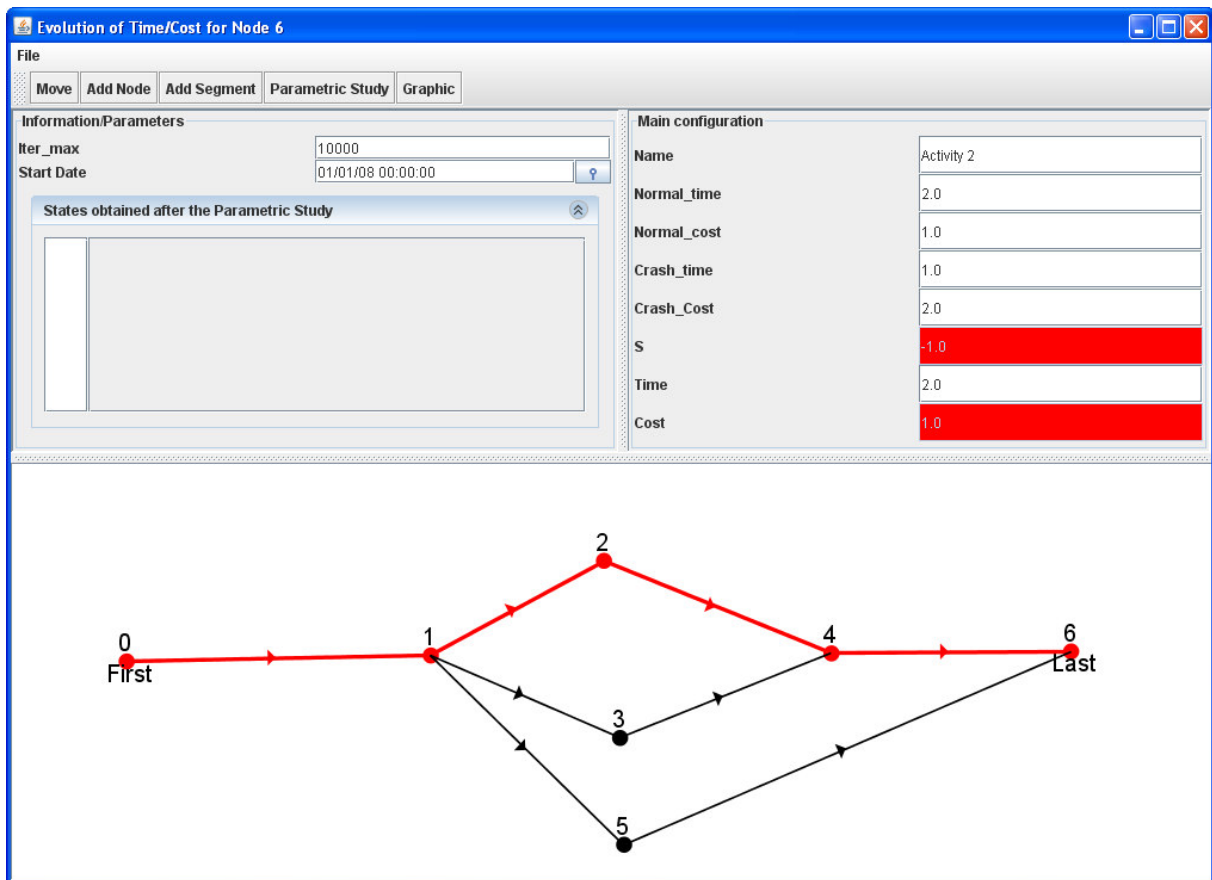


Fig. 132: Small network created with the software

The selected activity appears in blue and its attributes can be seen in the upper right part of the window:

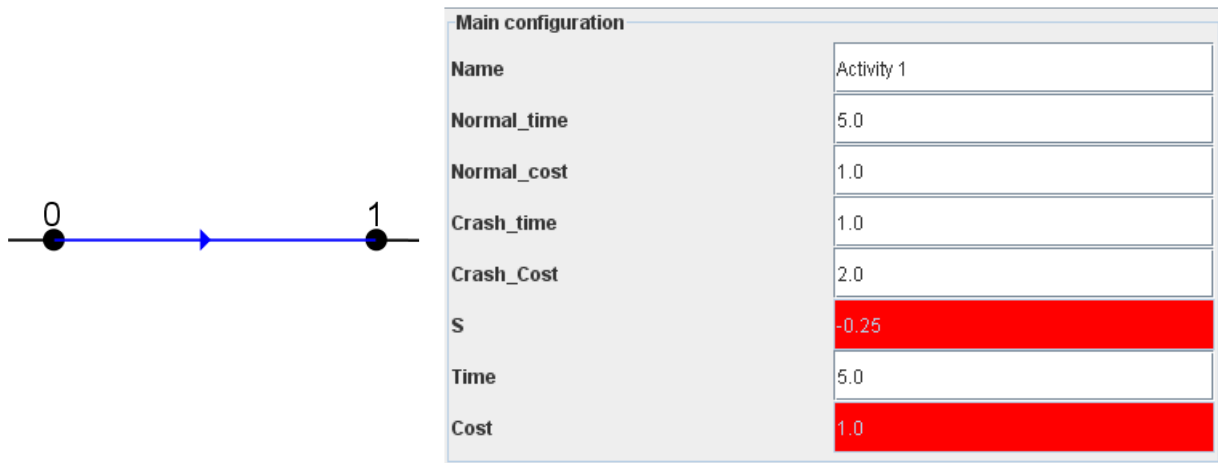


Fig. 133: Activity attributes

We straightaway see the main parameters:

- *Name*: by default the name is “Activity” + the number automatically generated, but a user can change it – to “Welding” for instance;
- *Normal\_time*, *Normal\_cost*, *Crash\_Time* and *Crash\_cost*;
- *S*: the slope of the diagram – see Fig. 130;
- *Time*: the duration chosen by the user (in second);
- *Cost*: cost of the activity for the selected Time.

Attributes in red are automatically calculated and cannot be directly changed by the user.

When we select a node (the colour becomes yellow), we have the following information:

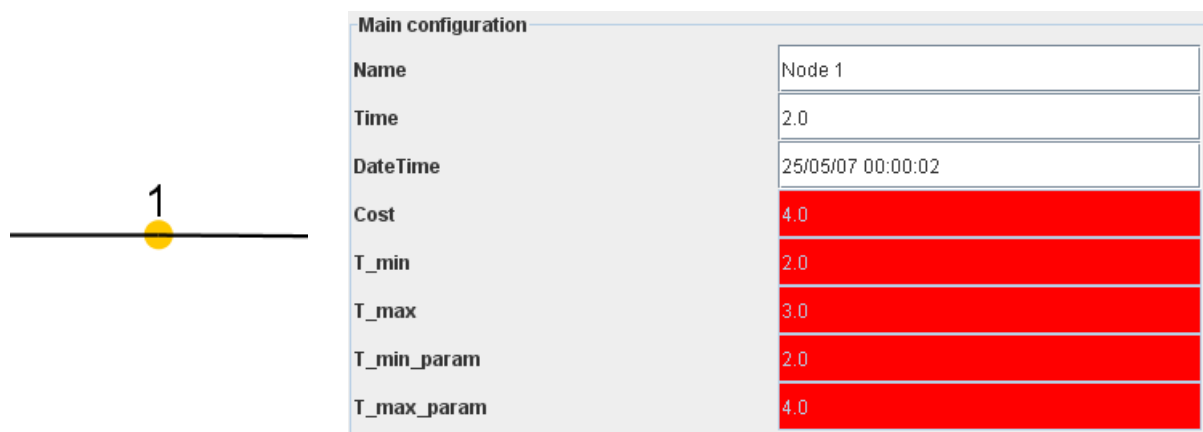


Fig. 134: Node attributes

Here again we directly see the main parameters:

- *Name*: by default the name is “Node” + a number automatically generated, but the user can change it – to “Welding done” for instance;

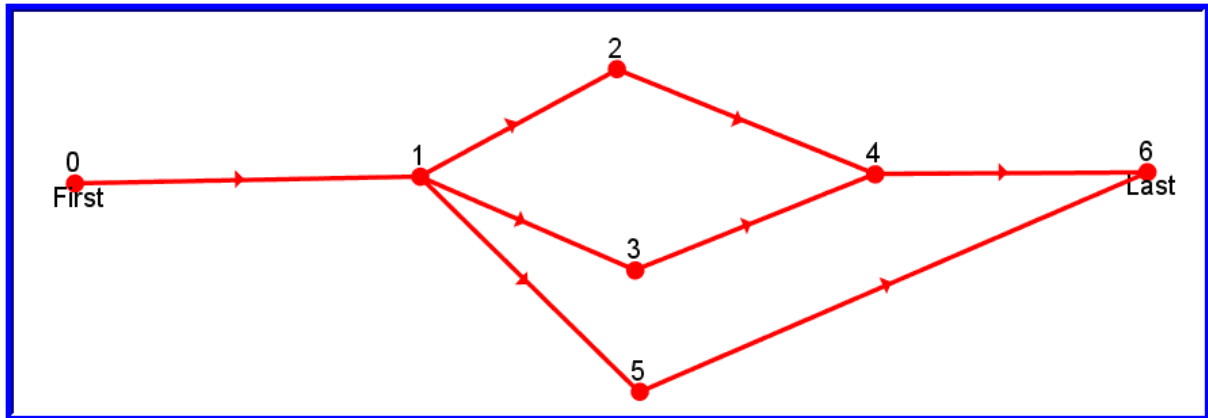
- *Time*: Time necessary to reach this node (in second);
- *DateTime*: Time necessary to reach this node in a more readable format than *Time*. For the project, we have to give a real start date (precise time and a date). The *DateTime* is the start date plus the time needed to reach this node. The start date can be selected with a calendar (see Fig. 135)
- *Cost* : Cost necessary to reach this node;
- $T_{min}$  and  $T_{max}$ : Earliest and latest start time;
- $T_{min\_param}$  and  $T_{max\_param}$ : Time to reach this node if all activities are set to respectively the *crash time* and the *normal time*.



Fig. 135: Calendar to choose the start date

For the last node,  $T_{min\_param}$  and  $T_{max\_param}$  thus give the real margin time needed to carry out the project. Here again the attributes in red cannot be modified directly by the user but is information automatically generated by the software. The *Time* is not in red because it can be modified for the last node! If we change it we directly run the optimization that will modify all the activities duration in order to reach the last node at this time and at the minimal cost! If we do not want to optimize the network but just see the total time with the duration of each activity chosen by the user, we just have to look at  $T_{min}$  (or  $T_{max}$  because they are identical for the last node).

In our example, if we change the *Time* of the last node we get the following figure:



**Fig. 136: Optimized network**

All the activities are now critical! This is due to the algorithm: if an activity is not critical, the optimization will increase its time in order to minimize its cost! Nevertheless if an activity has reached its normal time, the software supposes that if we still increase the time the cost will remain constant. Because of its “limitation” it is possible to have some activities that are not critical even after an optimization.

If we now launch a Parametric Study we will have the following results:

Etat	TimeOfActivity	CostOfActivity
Etat 0	24.0	
Etat 1	8.0	
Etat 2		
Etat 3		
Etat 4		
Etat 5		
Etat 6		
Etat 7		
Etat 8		

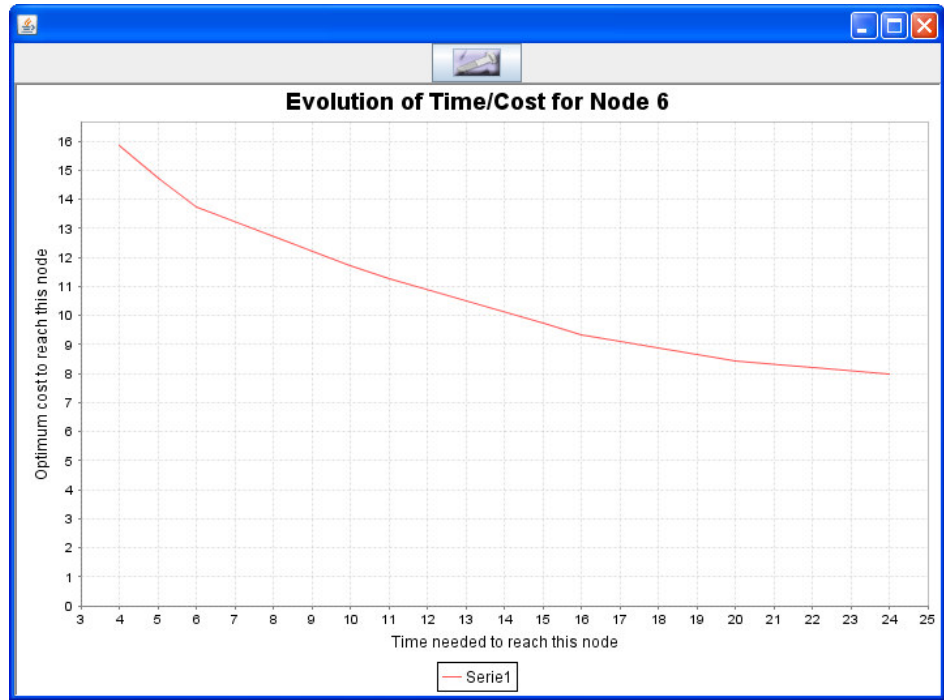
Activity	Etat 0	Etat 1	Etat 2	Etat 3	Etat 4	Etat 5	Etat 6	Etat 7	Etat 8
Activity 0	2.0	5.0	10.0	7.0	10.0	2.0	10.0	8.0	1.0
Activity 1									1.0
Activity 2									1.0
Activity 3									1.0
Activity 4									1.0
Activity 5									1.0
Activity 6									1.0
Activity 7									1.0

**Fig. 137: Results of a Parametric Study**

All the different characteristic states of the network are listed in a window. Each state is defined by a total time (and an associated optimized total cost) and by the duration of each activity. If we left click on a state we get the corresponding data and if we double left click we put all this data into the network.

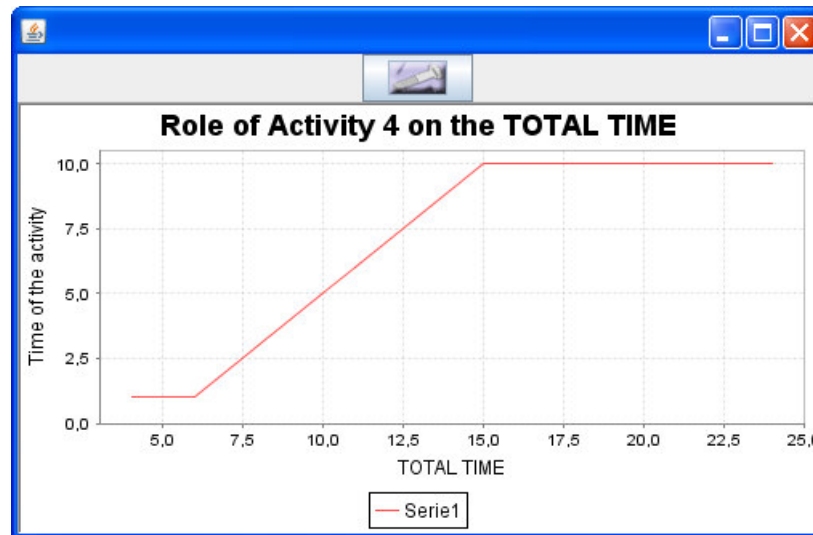
For each node and each activity, we can also have graphical results:

*For a node:*



**Fig. 138: Evolution of Time/Cost for a selected event**

*For an activity:*



**Fig. 139: Influence of an activity on the total time**

For the graphic of the last node, the first point (T, Cost) is necessarily the point (T\_min\_param, Cost) and the last one the point (T\_max\_param, Cost). The function will always be a piecewise linear function.

Be careful in analysing these previous graphs – mainly the graphs of activities. For the example shown before, it means that to get a total time of less than 6.75, we have to select the crash time for this activity. Between 6.75 and 15, it is that activity that we have to change to have the corresponding total time with a minimum cost. Beyond 15, the activity has reached its normal time and others activities are increased to reach the right total time.

*Remarks:*



- The normal cost should be strictly inferior to the crash cost.
- Crash time can be zero, but for numerical reasons at least one activity must have a crash time greater than zero (which is obviously always the case for a practical problem).

## 4.3 Application to the PrePreFabrication Workshop

### 4.3.1 Methodology

The PERT Tool developed can be very useful to analyse the productivity of the PrePreFabrication Workshop and to get similar and complementary results. How can we determine a PERT diagram for such complex workshop?

On a half-cell, the task is always the same for a kit:

**TACKING ON THE EXTREMITY OF THE HALF-CELL**



**DISPLACEMENT TOWARDS THE CENTER WHEN THE PREVIOUS KIT IS FINISHED**



**WELDING**



**FINISHING**



**EVACUATION**

The first step – preparation – is not taken into account because it is negligible compared to the others tasks – maybe except the displacement but this task is important to take into account correlation between others tasks.

These five tasks can easily be represented by a PERT diagram:

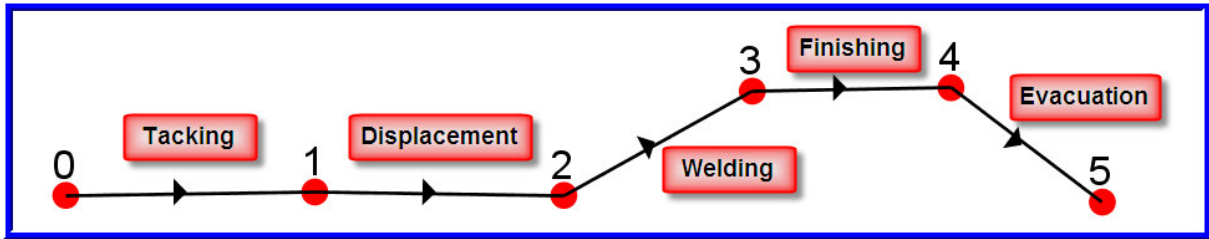


Fig. 140: Fabrication of a kit in a PERT network

On a half cell, as soon as the first kit has been moved to the centre area to be welded, the next kit can be brought. But the next kit can only be moved when the first one has been evacuated. For two kits the PERT diagram becomes thus:

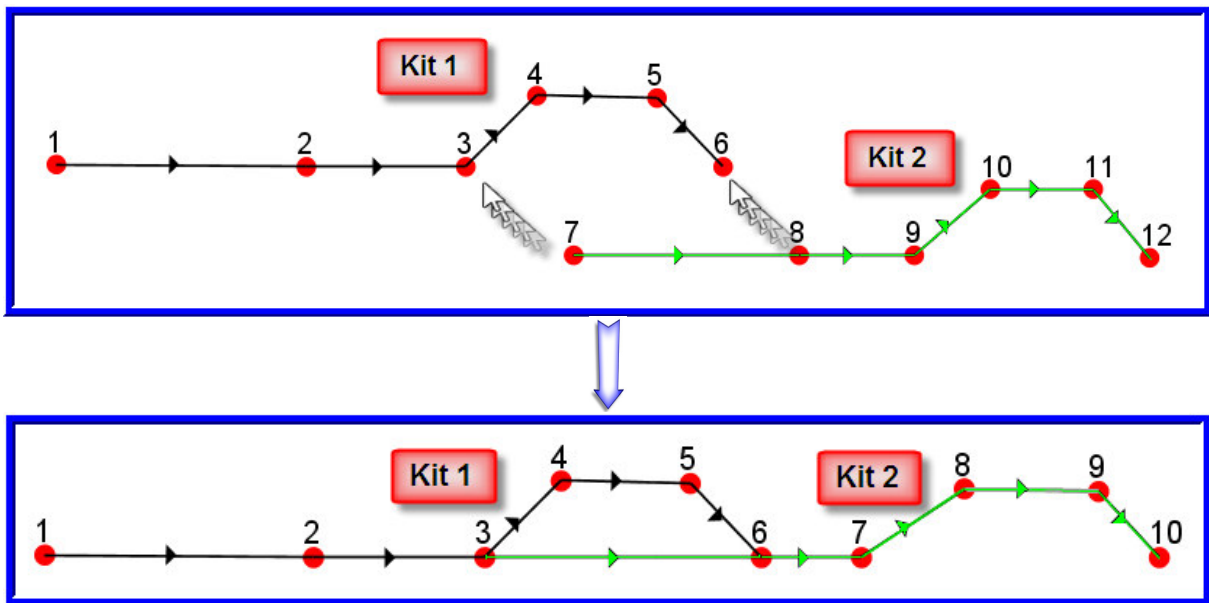
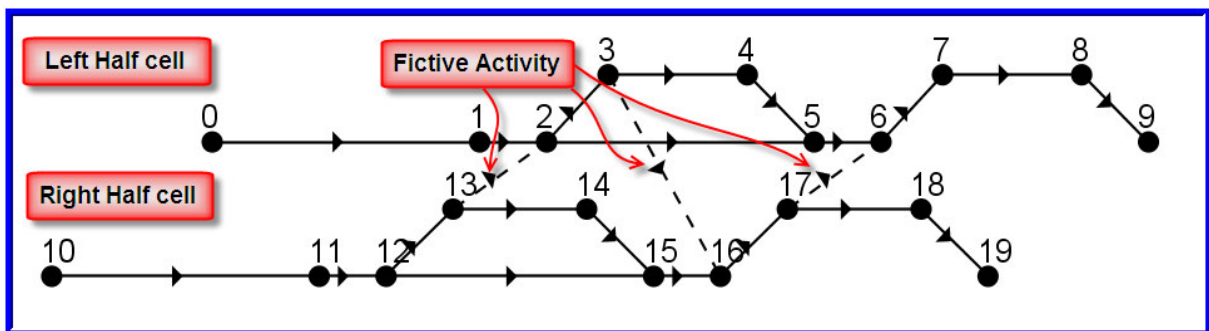


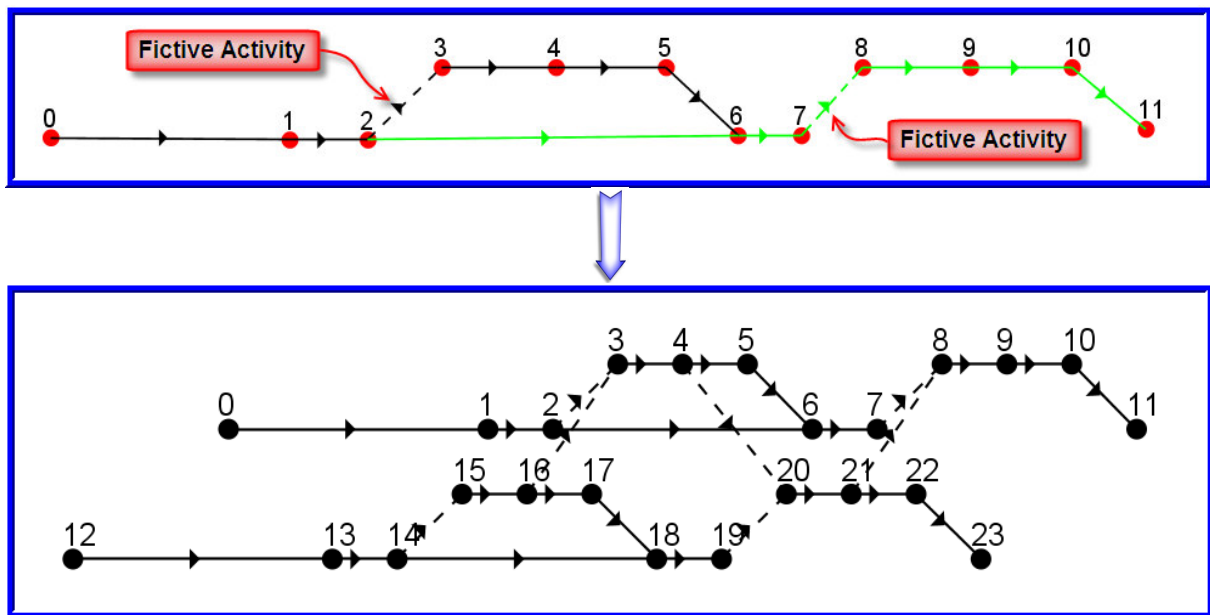
Fig. 141: Representation of 2 successive kits

Node 3 and node 7 are now the same. The operation is identical for nodes 6 and 8. Of course we can add as many kits as we want to this half cell. For the entire cell we have two PERT diagrams of the same kind. The only correlation is that we must wait for welding before welding on the other cell. So we have to add fictive links between welding operations: the beginning of the welding of the second half cell must start after the welding of the first half cell. That gives us Fig. 142.



**Fig. 142: Representation of a complete cell**

Nevertheless this PERT diagram is wrong. Indeed we see on the upper half cell that the second kit can only start (= activity from the node 2 to the node 5) when welding is finished on the first of the second half cell (= activity from the node 12 to the node 13). That does not correspond to reality. We can by-pass this problem by adding a new fictive activity between the Displacement and the Welding:

**Fig. 143: Fictive activities of a complete cell**

The obtained diagram is still not a correct PERT diagram because we do not have only one start node and one start end. Thus we have to create a start event and link it to the first event of each half-cell. This is not a fictive activity because we can use it to model the warm-up of the workshop! The workshop is not empty when we start and we know that it is not possible to start directly taking on the first kit: we have to wait for the displacement of kit that is already in the workshop. If we can estimate the time before this displacement we can put this value in the duration of the “warm-up” activity. The activity Displacement of the first kit can also be done only if the kit in the central area is moved away. Another warm-up activity from the start point to the first node of Displacement must thus be added. For the end node we just have to add a fictive activity from the last node of the half-cell to the end node.

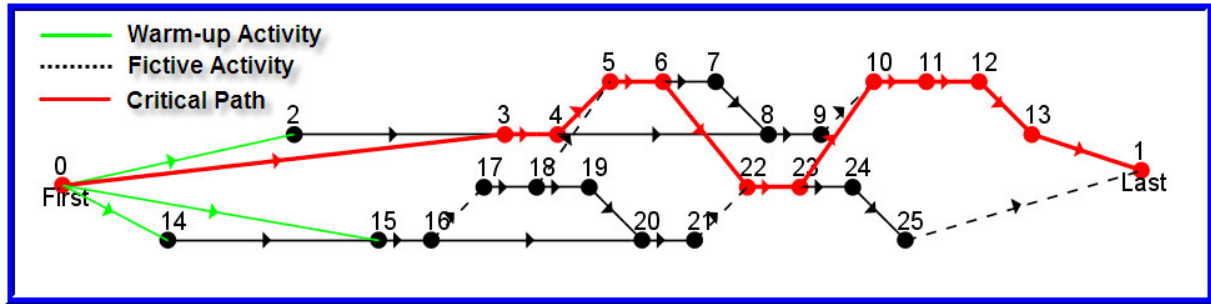


Fig. 144: A complete representation of a cell (4 kits)

On Fig. 144 we can see that the critical path has been automatically calculated. This is the PERT of the production sequence of four kits – two on each half cell. By convention the half cell that starts first is slightly shifted to the left. More precisely this is the start of the welding of the first kit that determines which half cell has to be shifted. Depending on the warm-up of each side, one half cell could even be shifted to more than one kit! The software automatically determines the right shift and we can thus have a configuration as seen in Fig. 145. In this figure the first fictive activity “between cells” is between the first kit of half-cell A and the second one of half-cell B – equals Activity from node 23 to node 5.

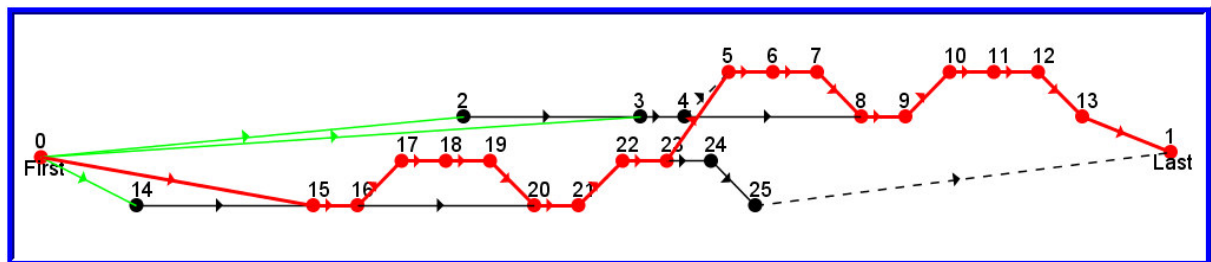


Fig. 145: Shift of a half-cell due to high difference of warm-up activities

When we have this situation we must be careful if we change the time of an activity that is a predecessor of the first fictive activity between welding operations. A large change could lead to a change in the order of the welding sequence → the PERT diagram could become false. We recommend never changing these times and consequently not to allow the optimization of these times.

For each cell we can have the same approach and thus have the representation of the entire workshop on only one PERT diagram. Obviously it rapidly becomes clear that it is demanding to draw the PERT diagram manually and to enter all the required information. Another option has thus been added to the tool: the possibility to import data from another tool by an XML file.

### 4.3.2 Automatic creation of the PERT

Due to the huge number of nodes and activities needed to model the production of a couple of kits we have to automate the creation of the PERT. The solution could be to write automatically all the needed data in a format that can be readable by the PERT tool. We have chosen to use an XML file.

The structure of the XML file is very simple and intuitive:

```
<Data>
  <Parameters dd="25" MM="5" yyyy="2007" HH="0" mm="0" ss="0" />
  <Ilot name="Ilot 1" X="100" Y="100">
    <DemiIlot name="DemiIlot1A" WarmUpAgr="34873" WarmUpTransf="75545">
      <Kit name="Kit 1445" Agraf="4126" Transfert="282" Soud="16440" Parach="10675" Evac="722" />
    </DemiIlot>
    <DemiIlot name="DemiIlot1B" WarmUpAgr="29174" WarmUpTransf="47873">
    </DemiIlot>
  </Ilot>
  <Ilot name="Ilot 2" X="100" Y="200">
  </Ilot>
  <Ilot name="Ilot 3" X="100" Y="300">
  </Ilot>
  <Ilot name="Ilot 4" X="100" Y="400">
  </Ilot>
</Data>
```

Fig. 146: XML structure to automatically generate PERT network

First of all we enter the starting date of the project. Then for each cell (*Ilot*) we give the *name* and its position in the window (*X* and *Y*). One cell necessarily has two half cells (*DemiIlot*) with attribute names, two warm-up times (see earlier explanations) and a number of kits. The order of kits in the XML is the order of construction in the PERT network. Attributes for kits are the *name*, Tacking time (*Agraf*), Displacement time (*Transfert*), Welding time (*Soud*), Finishing time (*Parach*) and finally Evacuation time (*Evac*).

Parameters relative to the cost are not specified in this file because at this point we do not have precise correlation between cost and time for these activities. The automated welding robot will always do the same welding at the same time except if we change the robot in the workshop. But the goal of the PERT in that case is not to find the best configuration of the workshop but to optimize “flexible” resources in order to minimize the production time/cost. By “flexible” resources we mean the workshop’s workers. Thus we can suppose that if we add workers to the workshop the average *Tacking* time and *Finishing* time will decrease. For the crane bridge the situation is the same as for the welding robot: only one is available for each half workshop with inflexible efficiency and we cannot consider the possibility of

changing the time taken for operations *Displacement* and *Evacuation*. To summarize, if we want to use the PERT optimization for the PrePreFabrication workshop:

- *Tacking*: optimization possible (resources are workers);
- *Displacement*: time “constant”;
- *Welding*: time “constant”;
- *Finishing*: optimization possible (resources are workers);
- *Evacuation*: time “constant”.

If the optimization of the PERT is necessary it is always possible to add parameters related to the cost to the XML file.

How to create this XML file? Actually it is not really easier to fill the XML than to introduce all the data in the model. We have two possibilities to automatically create the file:

- Create it directly from the Simulation Model;
- Create it directly from the database of assemblies.

#### 4.3.2.1 Creation from the Simulation Model

All the needed data (times) can be obtained by a simulation run. In fact, with the simulation of the production of a set of kits we get for each half-cell:

- The sequence of kit;
- Start date and end date of each operation.

We can thus have all the times needed for each operation. Nevertheless some points should be made. First of all *Tacking* and *Finishing* are done by workers that are shared between two cells. Duration times obtained for these operations thus strongly depend on the sequence and on the workload of others cells. These times are thus an approximation of the real necessary times. For the *Displacement* and the *Evacuation (Moving Away)*, the calculated times are well estimated and are independent of the sequence of kits. But the problem is that we have only one crane bridge for four half-cell and waiting times for these operations are not taken into account with this PERT model. The solution is to artificially increase the duration times of these activities. For the *Welding* the problem is the same as that of the crane bridge except that here we have fictive links between half-cells and the waiting time before welding is thus hidden thanks to these fictive activities!

If we keep the duration time of simulation and create, with eM-Plant, an XML file we can straightaway have a PERT as in Fig. 147.

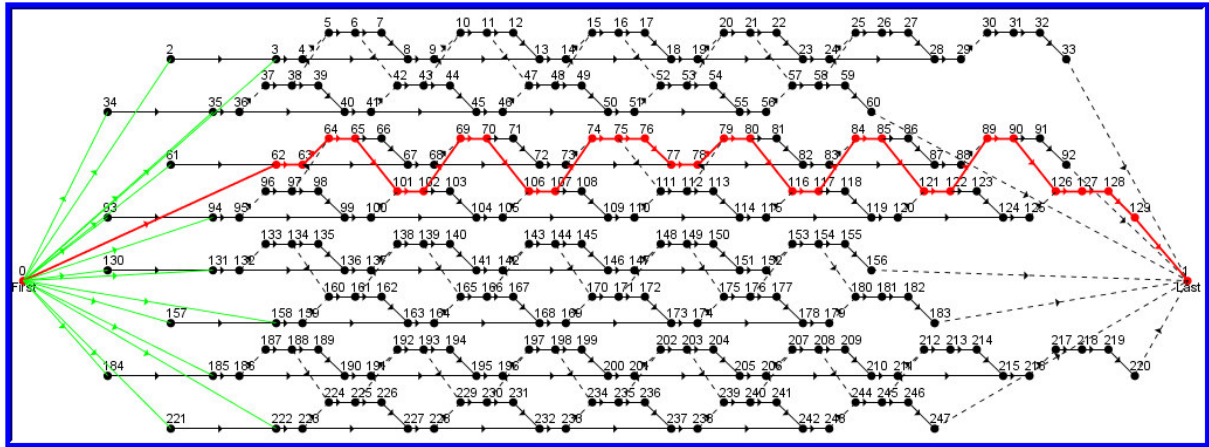


Fig. 147: Example of a PERT of a sequence of kits for the PrePreWorkshop

In the figure the critical path is clearly highlighted. The first two “lines” are the representation of the first cell of the Workshop (cell number 1), the two following are number 2, then number 3 and 4. The upper part of a cell is the left side of the workshop (side A) and the other one the right side (side B).

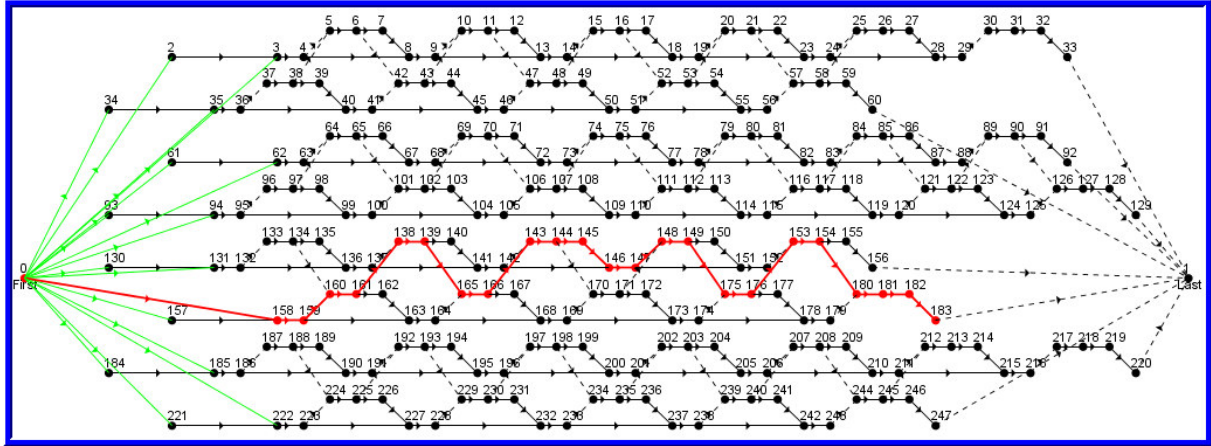
As described before we have two “Warm-up Activity”s by half cell to model the fact that we do not start with an empty workshop. Duration of the first one (from node 0 to the first node of the half-cell’s first kit) is simply the difference between the start date of the kit and the start date of the simulation. The duration of the second one (from node 0 to the second node of the first kit of the half-cell) is the difference between the start date of the Displacement operation of the kit and the start date of the simulation. As a consequence, for each half-cell the branch constituted of the first virtual activity and the first tacking activity is not really necessary in the diagram.

For each cell it is important to know which half-cell will start the welding first. In Fig. 147 we see that half-cells that start welding are respectively side B for cell 1, side B for cell 2, side A for cell 3 and side A for cell 4. The tool automatically shifts to the right half-cells that start later and consequently adjusts fictive activities. To determine which half-cell has to be shifted a *pre-calculation* is done based on the duration of the preceding activities of the first welding operation.

Even without the optimization option a figure such as in Fig. 147 is really interesting for the planner. We see immediately which operation cannot be delayed! Before launching the production workers thus know which kit must have priority! For the other activities we also

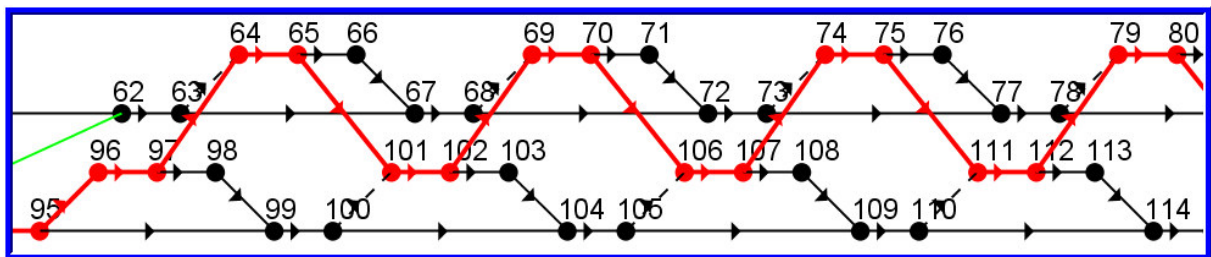
have the float straightaway and we can class each activity according to this parameter to create a priority list of activities!

Furthermore we can also have the critical path for a specific cell chosen by the user – see Fig. 148.

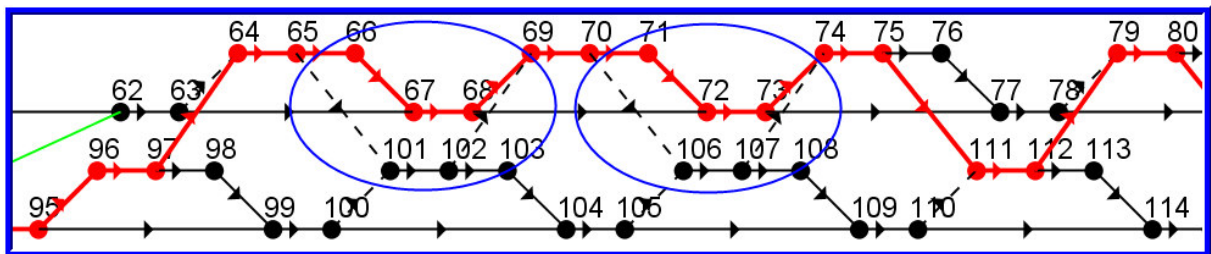


**Fig. 148: Critical path for the Cell 3**

An important problem of the PrePreFabrication workshop is to try to maximize the use of the welding robot. By observing the critical path of a cell we see straightaway if the welding robot is saturated or not. We have to remember that for two half-cells each welding operation is linked by fictive activities. If the critical path contains these fictive activities, the welding robot is correctly used and is not waiting between two kits. Fig. 149 and Fig. 150 illustrate that point.



**Fig. 149: Saturated Welding robot for a cell**



**Fig. 150: Not saturated Welding robot for a cell**



#### 4.3.2.2 Creation from the database

All the data needed to create this PERT is available in the database. For the *Tacking* we have the total welding length for each activity and we can thus estimate the total tacking time for the kit. Obviously this is still an approximation because we do not know how many workers will be free when the tacking will start. Here again we thus have to overestimate the tacking time calculated to include into the PERT.

For the welding the operation is not simulated but is obtained directly from the database: either from the PHL (see section 3.2.3) if information is available or by a simple calculation based on total welding length.

For the *Finishing*, the method is the same as for the *Tacking*. We just have an approximation and must overestimate the time to take into account the availability of workers.

For the operation of the crane bridge (*Displacement* and *Evacuation*) we know the unload and load time of the crane bridge. The simulation also takes into account the setup-time and movements of the crane bridge. A (small) overestimation of the calculated time is thus required.

To get correct overestimations of these operations a statistical analysis for a huge quantity of assemblies can be carried out. We have to simulate these assemblies several times and compare simulated times to calculated times. For instance we can make the link between the welding length of an assembly and its average time that the simulation gives. That will give us a good idea of the weight to use in the calculation of these times!

In the simulation model we have all the required information, which comes directly from the database – geometrical characteristics of assemblies, etc. There is the possibility to either use the time obtained from a simulation run or directly from the database with a sequence chosen for each half-cell. There is no obligation to use eM-Plant to create the XML file but if we want to create it directly from the database – for instance from Microsoft Access© – we have to build a new interface. The interface is used just to see available kits and to select the sequence for each half-cell. Rather than creating a complete new interface (and also some programming code to generate the XML file) we have chosen to use the one developed for the simulation in eM-Plant.

As a consequence eM-Plant is used each time to create the XML file but with two possibilities:

- Using results obtained from a simulation run;
- Using data obtained directly from the database with a calculation based on statistical analysis.

#### 4.3.2.3 Comparison with the simulation on the total production time

A comparison between the total production time given by the simulation and the PERT diagram is still interesting. Even if the PERT is created from the simulation results there remains a difference for the reasons mentioned before. If the duration times of each activity come from the simulation run, the differences are due to waiting times before tacking, displacement, finishing and moving away. The waiting time before welding is taken into account thanks to the fictive activity added. Notice that we cannot create fictive activities for other operations, as is done for welding, because we do not know the order of use of – for instance – the crane bridge. This is not the case for the welding: robot goes to one side to weld, then to the other side when the corresponding kit is ready.

With the example of Fig. 147 we have 45 kits. The results are given in the following table:

	Results of the Simulation run	Results of the PERT
Total Production time (s)	249912 (s)	241420 (s)

**Tableau 1: Comparison of results between Simulation and PERT**

The difference between these results is 3.4%. As expected the total production given by the PERT tool is lower than the one given by the simulation. One reason is that the smallest operation (preparation of kits) is not modelled in the PERT diagram. Notice that there is no difficulty to add this operation but its duration is very low when we compare it with the other activities.

#### 4.3.3 Conclusion

Although PERT diagrams of the workshop do not give new results than those obtained by the simulation we can now visualize results in a better way. The main advantage is seeing straightaway which kits and which operations cannot suffer any delay without having an influence on the total production time: bottlenecks are clearly highlighted.

Nevertheless the optimization of the cost is a real contribution that the simulation is not able to make. Of course this optimization is only possible if the planner can assess the link between production time and resources – data that we didn't have in the frame of this thesis.

Another advantage is that CPU-Time to get the total production time is really faster than for a simulation if we create the PERT directly from the database. In this case we have to approximate some operations because the PERT cannot manage all the interactions between the activities – share of resources. This gain in CPU-time is a real benefit because we can more easily link it with an optimization algorithm! Here optimization means optimization of the sequence – like the simulation does – and not an optimization of resources. Once the “best” solution (sequence) is found it is still possible to run the simulation to validate (or not) the solution.

As already mentioned, the main problem of the PrePreFabrication workshop is the shared resources such as workers and crane. This affects greatly *Tacking – Finishing – Displacement* and *Evacuation (Moving Away)* times. It is also not possible to take into account workers’ shifts, breaks, breakdowns and so on. Geometrical constraints are also completely avoided: cells have a limited size and we thus cannot have any pair of kits on the same half-cell at the same time! This problem is a sequence constraint that is managed in the simulation but not with the PERT tool if we create the diagram directly from the database! An important point not discussed is the joining operation. This operation is not modelled in the PERT diagram and is supposed not to be a bottleneck. *A priori* this is always the case but the virtual simulation could warn the planner if that unexpected situation occurs contrary to the PERT tool.

To summarize, the Advantages/Disadvantages are:

#### 1) Advantages

- Optimization of production Cost available;
- Easy to visualize bottleneck and float time for any activity;
- CPU-Time weak → very useful to link to an optimization algorithm.

#### 2) Disadvantages

- Problems of shared resources such as workers and cranes;
- Not possible to take into account workers’ shifts, breaks, etc.;
- Problems with geometrical constraint of kits;
- Joining is not modelled.

The final conclusion is that we have fewer details than in a simulation run but this PERT tool linked to the PrePreFabrication Workshop gives supplementary information and good knowledge of bottlenecks in less time.

# Chapter 5

## Development of a tool to solve Space Allocation Problem

### 5.1 Introduction

A common problem met in most industries – and especially in shipbuilding – is the space allocation problem. Building one or several ships requires a space that can be very large and one major goal of any shipyard is to reduce storing areas. Nevertheless these storing areas are indispensable and can never be totally avoided. The bigger the production workshops are, the easier it is to manage production hazards and to control these storing areas. Available space for all the shipyards is in most cases limited and a shipyard designer has to find a compromise between workshop size and storing area size. Consequently the available space for workshops is most of the time smaller than the real need. That point makes schedulers work harder and they really have to optimize the production to respect deadlines fixed by ship-owners.

Fig. 151 illustrates the lack of space that some shipyards can encounter. Even if ships become bigger and bigger, once the shipyard has been built it is generally very difficult – or even impossible – to modify workshop sizes.



**Fig. 151: Examples of space limitation for some shipyards**

Solutions must be found to improve the productivity taking these space constraints into account. This chapter will detail a tool developed to help schedulers face this common problem in the naval industry. The tool contains two modules:

- An graphical user-friendly interface (GUI) to manage the workshop and the scheduling;
- An optimization tool to optimize this scheduling.

In the first version the interface has been developed in Microsoft Excel© and the optimizer in C++. In order to improve greatly the interface we have chosen to make a new interface, developed this time in Java. To keep homogeneity in the software on this occasion we also have translated the optimizer into Java. Only the last version will be presented in the chapter.

What kinds of problems are treated by the software? As mentioned before the tool solves space allocation problems but we have to specify the limitations of the tool. Hypothesis of the problem are the following ones:

- A specified workshop has a list of products to build;
- The workshop could be divided in working areas – with fixed dimensions – where products will be built;
- Each product could be different but must have a parallelepiped form. Dimensions must be given;
- Each product has an earliest and a latest starting date. The real starting date can be chosen in between both these dates. Eventually they can coincide.

The goal of the scheduler is to determine the position of each product in a working area and its starting date.

The problem is similar to a nesting problem<sup>17</sup> but still has some important differences. In a way nesting problems are more complex because we do not necessarily have rectangular shapes. On the other hand our problem is much more complex because it has four dimensions: the three spatial dimensions and the time dimension! The differences are thus very large and we will not use same techniques. As will be explained in the next section the chosen method for the optimizer will be inspired by the 3D bin packing problem.

---

<sup>17</sup> An operation that consists in minimizing the amount of scrap raw material produced by cutting operations to get manufacturing parts

We will apply the tool to three different workshops of the Saint-Nazaire shipyard. Two are “real” workshops and one is not really one but is the area next to the dry dock where final blocks are built before being put in the ship. Its available space is also limited and requires specific attention for the scheduling. The two other workshops are used to assemble blocks but with a limitation in their weight: 120 tons for the first one – simply called the 120 tons workshop – and 180 tons for the second one – simply called the 180 tons workshop. These workshops are not fundamentally different but nevertheless they still have several specificities. The tool developed can manage these particularities.

## **5.2 Development of the tool**

### **5.2.1 Introduction**

What exactly is the problem to be solved? The situation is the following one. We have a workshop with one or more working areas where activities must be performed. An activity could be: building a block – this will be the case for the workshops we studied but it could be a completely different task. The duration is linked to the activity and most of all it will require ground space to do it. In other terms it means that in the workshop we have to book a limited space from a starting date to an ending date. Simplifications will be made in the software:

- In reality the block does not occupy the entire surface during the first days of the building. Nevertheless we will book the final surface from the first day to the last one. In fact this approximation is not far from reality. Even if we could maybe gain more surface with a more precise approach this scheduling is better because if we meet problems that avoid workers working on a block for one or two days, it is still possible to add workers the next days to exit the block at the right date. If we have a very precise scheduling, taking into account the evolution of space utilization day by day, we will loose a lot of flexibility in the management of resources;
- We book on the surface a rectangular shape no matter what the real shape of the products are. Here again the simplification does not take us too far from reality.

Each block also has a time window in which the work must be accomplished. Indeed we cannot start before the required pieces coming from a (or several) previous workshop are built. Suppliers give us the earliest starting date for each activity. The activity cannot start in the workshop before this date. On the other hand our workshop is also a supplier for the next one that requires us to supply finished products by a specific date. This date is our latest ending

date: the activity must be finished at this date. So we have for each block/activity an earliest starting date and a latest finishing date.

What is the duration necessary to execute the activity? In our problem duration is a data. It has been estimated previously and we cannot change this data for the planning. Consequently we can deduce a latest starting date and earliest finishing date.

For each activity our objective is thus to find its position in the workshop and its real starting date – between the earliest and the latest starting date. The difficulty could become very large when the number of activities to be allocated is very high. In the example of the workshops studied before to have a specific tool, the planning was done on a simple blackboard with cut paper to represent blocks!

The first objective is to build a visualisation tool to facilitate the manual scheduling of the workshop! Drag-and-drop functions must be available with tools to detect overlapping blocks. Indeed with a high number of blocks, overlaps are not easy to find each time due to this time dimension. The goal is to find a way to clearly visualise the workshop situation over time.

The second objective is to automatically optimize the scheduling. When we have a list of blocks to assign, the solution is not easily done manually. Automatic allocation is a real improvement and allows the planner to gain not insignificant time. Better solutions could also be found!

The next sections will detail firstly the interface developed and secondly the optimizer tool.

## **5.2.2 Interface of the software**

As mentioned in the introduction, the first version of the interface has been developed in Microsoft Excel©. When the interface has required more and more advanced functionality we have chosen to develop a new one in Java. The advantages are greater fluidity and flexibility. Several indispensable functionalities such as drag-and-drop were not possible in the first version.

How to represent in a clear visual view this fourth dimensional problem? We have chosen to create two different views of the workshop:

- The first one is a spatial view of the workshop;
- The second one is a timeline view.



The spatial view is simply the overhead view of the workshop. We simply see the situation at a specific date. Blocks are represented by their length and width. Unfortunately we lose the third dimensional information but a special filter has been added to the view. We can select the height and only the blocks that have an greater height will be shown in the workshop – see Fig. 152.

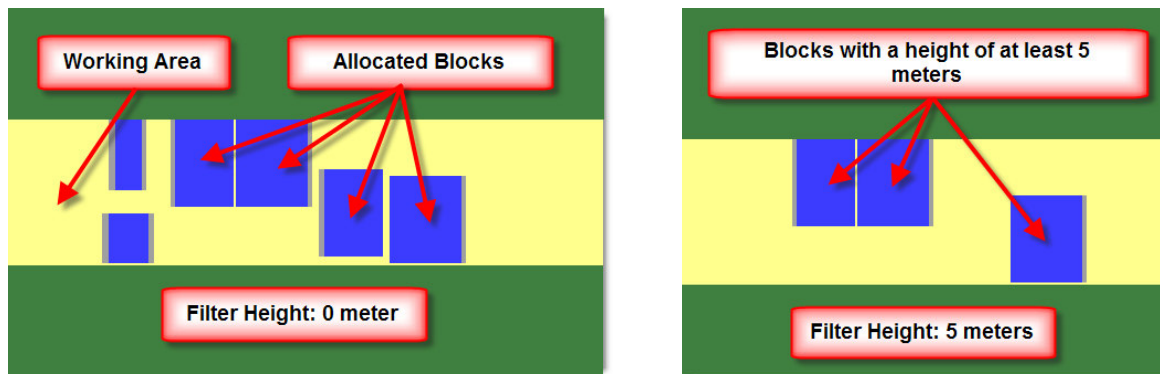


Fig. 152: Height filter of the spatial overhead view

The timeline view is the overhead view of the workshop with a dimension in space and the other in time. The chosen space dimension is the longest of the workshop. Fig. 153 shows this timeline view.

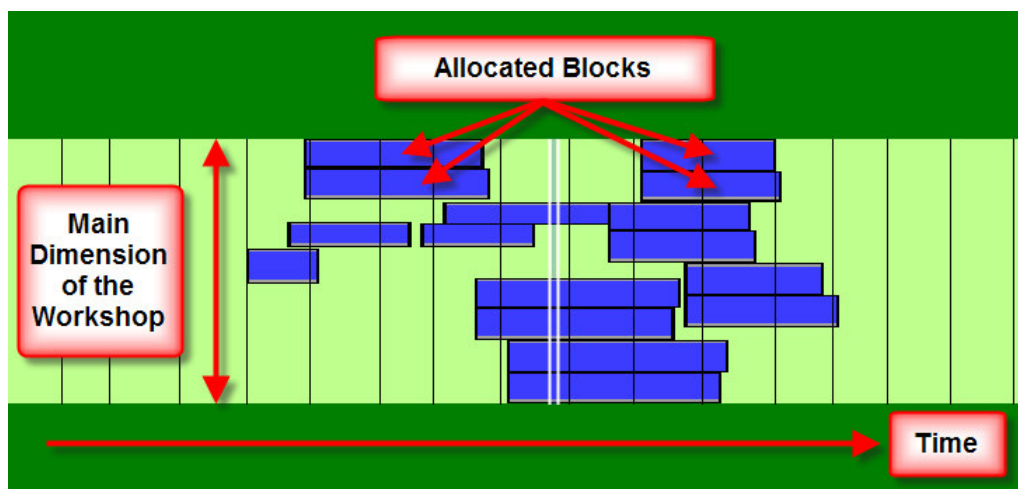
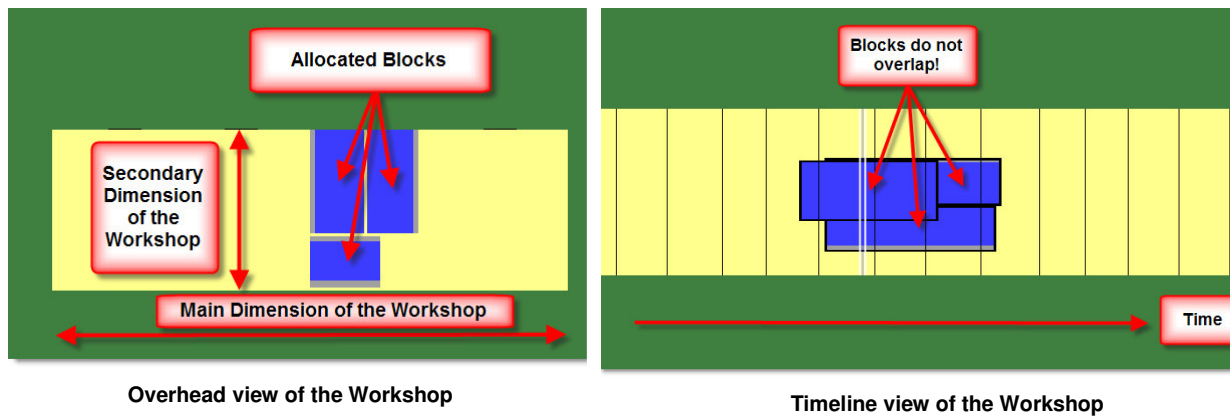


Fig. 153: Timeline view of the workshop

Consequently, in this view we can have blocks that seem to overlap although they do not in reality but are simply next to each other – see Fig. 154.



**Fig. 154: Fake overlapping blocks**

In this figure the spatial view shows the workshop's state at the specific date selected by the vertical white line on the timeline view. This vertical line can be moved over time with the mouse and instantly the update is made in the spatial view! This is a common method to easily see the evolution of the workshop over time!

Before continuing with the description of the software we will describe the format required for the data. The list of blocks to be allocated must be saved in a csv<sup>18</sup> file named DBREF – each line corresponds to a block/activity – with the following important fields:

- Name of the block. This field must be unique for each block to avoid identification problems;
- Earliest Starting date;
- Latest Ending date;
- Time required to build the block – duration;
- Geometrical dimensions: length, width and height.

This information is sufficient to run the software and carry out a planning. However some fields can be added depending on the specificities of the workshop. For instance for the 180 tons workshop, weight could play a major role in the way a block exits: if the block is too heavy for the handling systems it must be exited by a skid platform. These skid platforms cannot be used everywhere in the workshop so these blocks can be allocated only at special locations. In that case the field weight must be added in the DBREF.csv file.

---

<sup>18</sup> *Comma Separated Value. Specific file where data is divided into fields – without any format – divided into fields separated by commas*

Now the software has all the required information about the products but it needs to know the characteristics of the workshop. Developments have been focused on the flexibility of the tool. We did not want to restrict the software to only one or two workshops. To face this problem we have defined the workshop in a XML<sup>19</sup> file. The file is divided into different tags:

- The first information is related to the main workshop characteristics: name and the two main dimensions. As explained already, the timeline view needs one spatial dimension. Which dimension is chosen is also defined here;
- We have information about working areas. Indeed the workshop could be divided into cells that have specific equipment. For instance a special crane bridge to move away products. Consequently we must describe each one of these working areas: name, dimensions, relative position in the workshop and height of the crane bridge. The last parameter is important when we exit a block to see if other blocks are not in the way. We can also divide working areas into different preferential working areas. Sometimes a block could have a preferential working area. There is no obligation to build the block there, but if there is a choice it will be put there by preference. These sub-divisions are not indispensable but if they are defined we must specify: the name, dimensions and relative position in the workshop;
- Then we have a list of annotations: this is text information that will be written in the spatial view of the workshop. For instance we can write where exit ways are located. Notice that in the interface we have the possibility to visualize (or not) these annotations;
- Finally we have parameters related to specific rules of the workshop. As already explained we have used the interface for three different workshops. They each have specific rules. Specific constraints and/or rules must be noticed in the XML file.

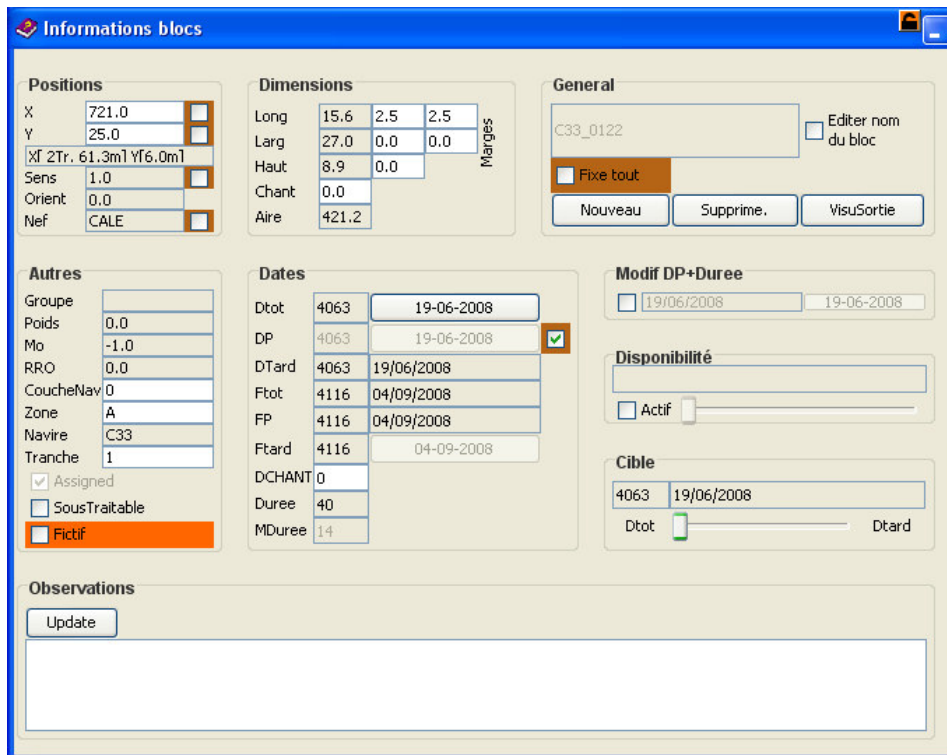
---

<sup>19</sup> *Extensible Markup Language. This is a generic framework for storing any amount of text or any data whose structure can be represented as a tree;*

When we describe working areas we always have to create one named *Corbeille*<sup>20</sup>. This is a virtual area where no allocated blocks are stored. In other words when we start a new scheduling all blocks are in this virtual area.

Now blocks to be allocated and the workshop are completely defined and there is sufficient information to run the interface. If we open a new file, the workshop is empty and all the blocks are in the bin.

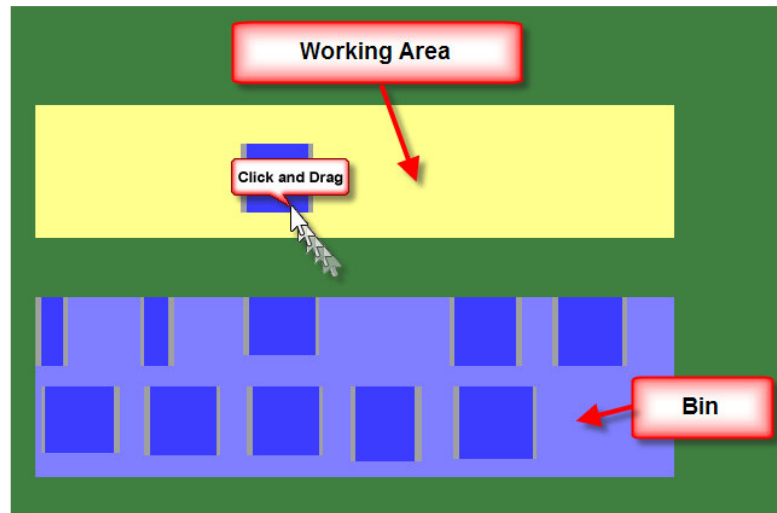
When we double click on a block we open a dialog box with its parameters – see Fig. 155. Some parameters can be changed, others not. Notice that we have the possibility of defining margins for each block. In practice we will rarely put two blocks next to each other for reasons of convenience. Furthermore we can add restrictions to each block. We can fix its x or y dimensions, or fix its working area or fix its starting date! These constraints are taken into account for the optimization but also if we want to do a manual allocation. Notice that a multiple selection has been implemented. We can select several blocks at the same time and change the parameters for each block. For instance we can select all the blocks and add an x margin of 2 meters and a y margin of 1 meter. This avoids repeating the operation for each block.



<sup>20</sup> Bin in French;

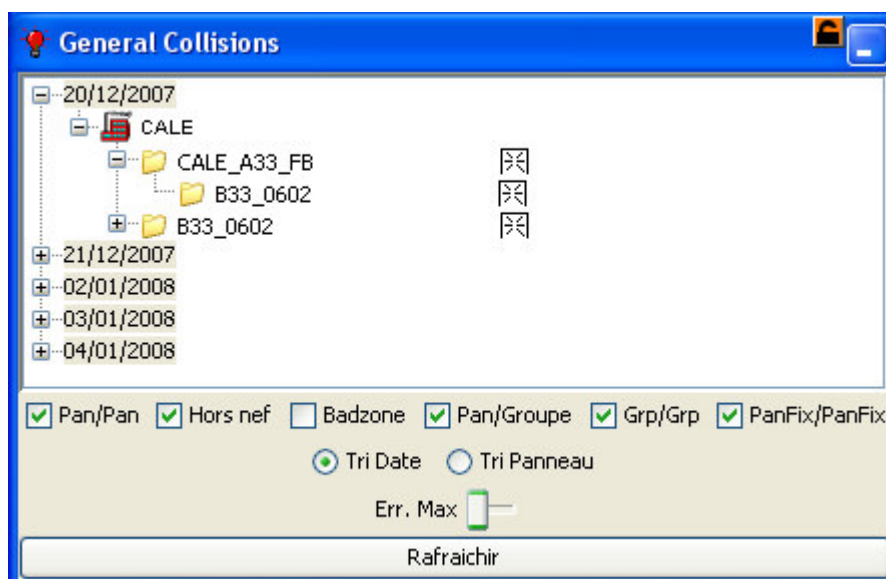
**Fig. 155: Block's information**

One interest of the software is that we can easily allocate blocks by a simple drag-and-drop – see Fig. 156. Simple checks are then automatically done. For instance if we have fixed the working area of the block, it will automatically go back to its initial position if we try to put it in another area.

**Fig. 156: Manual allocation of a block**

When an overlap is detected blocks are clearly highlighted – see Fig. 158.

Nevertheless these overlaps appear only in the spatial view. The problem is that this overlap may last only one day. Consequently if we have a list of blocks allocated to a time period of several months it could be difficult to find all the overlaps. To facilitate the detection a colliding tool has been developed – see Fig. 157.

**Fig. 157: Overlaps detection tool**

A list of colliding blocks is generated. If we click on it, the spatial view is automatically updated at the right date and the blocks are selected!

One important feature is the possibility of creating fictive blocks. Indeed for some reasons the space in a working area could be limited – because a machine is temporarily stored there, etc. We do not want to allocate blocks to this position. To avoid this we can create a fictive block – with the same parameters as for a normal one – to simulate this obstruction during the time we wish. These blocks have special colours in the software – see Fig. 158.

During a scheduling we cannot modify the position of the blocks that are currently in the real workshop. These blocks must be completely fixed. For more clarity they are represented by a special colour.

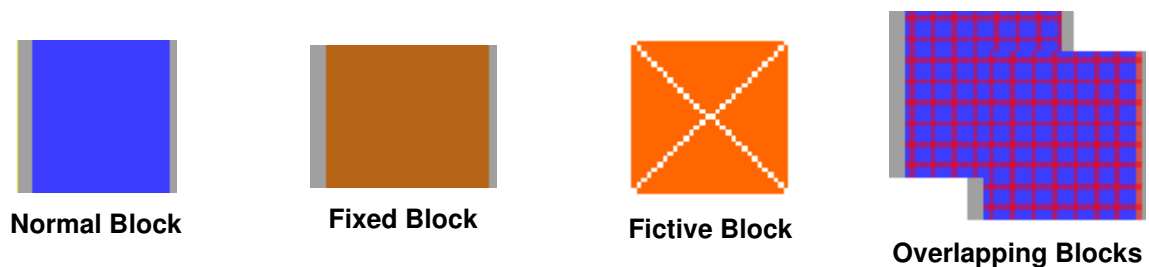


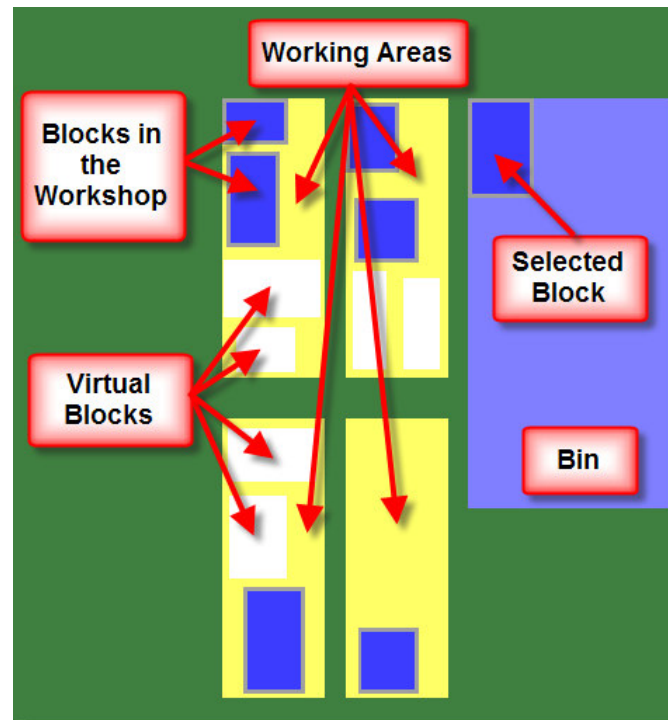
Fig. 158: Legend of blocks

Even with the developed interface, manual allocation of all the blocks is still difficult. Of course we can detect overlaps easily and some constraints are automatically checked. Imagine that we have already put many blocks in the workshop and that we still have some blocks to allocate. We select a block in the bin and view the spatial view at its earliest starting date. We have to put it between this date and its latest starting date. We see lots of space available in the spatial view so drag-and-drop the block to an empty location. The problem is that we are not sure that before the end of this block another block has not already been assigned! Of course we can look at the timeline view but this view lacks one dimension so all the information required is not clearly available! We have to place the block, check the collision tree, if there is a problem, then change its position, check again and so on. This method is not really useful. To solve the problem we have integrated a button in the block's information panel - Fig. 155 – named *Disponibilité*<sup>21</sup>. When we activate this area the spatial view is set automatically to the earliest starting date of the selected block. All assigned blocks

---

<sup>21</sup> *Disponibilité in French*

to this date appear but also new virtual blocks in white – see Fig. 159. These blocks are blocks that are not yet in the workshop at the current date but are blocks that will be assigned between this date and the end date of the selected block if we allocate it in the workshop at this date.



**Fig. 159: Assigned blocks and virtual blocks**

If we assign our block to a white area, we know that there will be a problem because another block is already assigned there in the future, before the end of our block! But if we can put our block in an empty location – without normal and virtual blocks – we are sure that the space is really empty and that we can assign it. If we do not find space we can modify the value of the cursor in the block information panel. This cursor can be changed from the earliest starting date to the latest starting date of the current block. When we change it the view is directly automated: normal and virtual blocks are updated! Of course we cannot go before the earliest starting date and after the latest one. In moving the cursor we can easily see if there is real space to allocate our specific block to. We have to keep in mind that the virtual blocks seen depend on the selected block. The functionality used on the 6th of March for a block will not show the same virtual blocks if we select another block at the same time. This tool is really useful – even essential – for the user!

Another difficult constraint to see is the exit problem. At the ending date of a block we can have other blocks between it and the exit door. This could be a problem if the height of the crane bridge minus the height of the block to be moved away is lower than the blocks

which are in the way! In this case it will not be possible to exit the block at its end date. With all the tools developed it is not yet easy to see this problem quickly. Another functionality has thus been added to facilitate the detection of this problem. Again the button named *VisuSortie*<sup>22</sup> is available in the block information panel - Fig. 155. When we select a block and click on that button the spatial view is set automatically to the ending date of the block. A change is also made to the height filter of the workshop: the chosen height is the crane bridge's height minus the height of the selected block. In other words, blocks that are still visible in the workshop restrict the exit of the block. The user can easily see if there is a possibility to exit the block or not.

Now we do not have an empty workshop but some blocks have been assigned – and maybe also some fictive and/or fixed blocks. If we want to save our work we cannot save it in our initial file *DBREF.csv*. Indeed for each assigned block we now have new information: its working area with its coordinates and its real starting date! These fields were not available in the initial file. In fact when we save we create a new csv file – with the name of our choice. This file has the same fields as the *DBREF* file with new fields for each block:

- The working area where the block has been assigned;
- Coordinates of the upper left block's position in this working area;
- The orientation of the block;
- The real starting date.

Note that fixed blocks are also saved in a separate file named *Fixes.csv*. This separation is again done for fictive blocks saved in the file *Fictifs.csv*. In fact when a scheduling is finished we can fix all blocks. They are thus saved in the file *Fixes.csv*. When we have a new list of blocks we have a new *DBREF* file. When we create a new scheduling the software takes files *Fixes*, *Fictifs* and *DBREF* and mixes them together to save a single csv file. Of course we can directly re-open a previously saved file.

An important point has to be mentioned. All given duration times by block correspond to the real time to finish the activity. This time does not include days off! If mounting a block takes fourteen days we can make big mistakes with that simplification. A solution has been provided by integrating it in the software calendars. We can specify for each day of the year

---

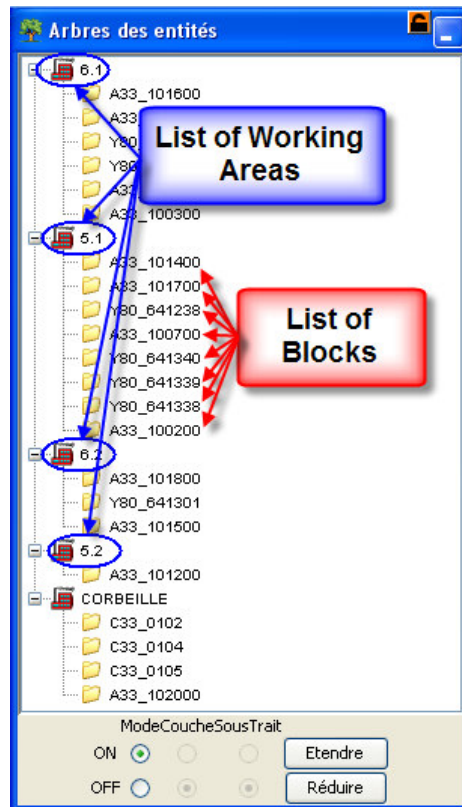
<sup>22</sup> Equivalent to « See at the Exit »



which day will be a working day and which will be a day off. In the timeline view we only have access to open days and this is always the case in the software: we work with a numerical system representing dates in a continuous way. It means that skipping a day in the timeline is equivalent to skipping a working day – but in reality it could be several days. However the user generally prefers to have access to real dates. A conversion is thus done between the dates of the software and real dates. This is done with the working calendar integrated in the tool. For instance in Fig. 155 we can see the real date each time date – in day/month/year format – and the tool's date – in a numerical format.

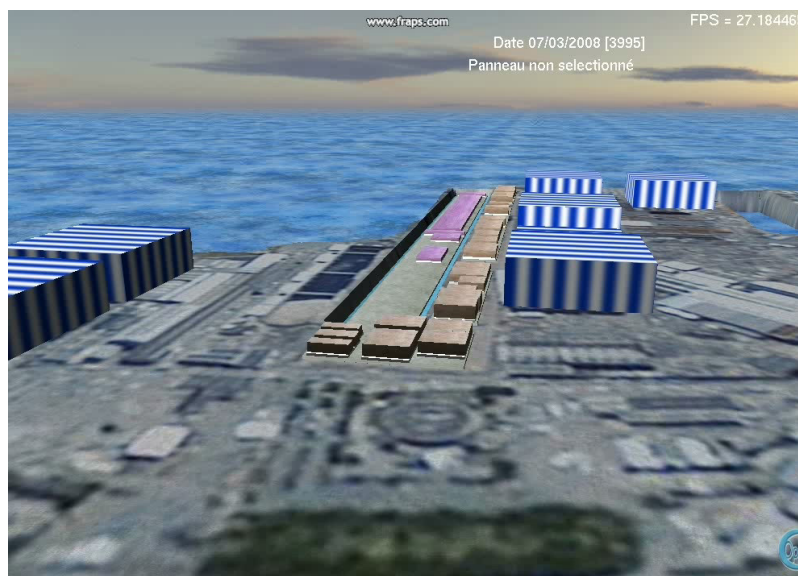
The tool has also other interesting functionalities:

- Research functions – for instance to find immediately a specific block;
- Management of blocks in a hierarchical tree structure – see Fig. 160;
- Refined selection tool – possibilities to select activities that have some parameters included between specified values;
- Possibility of group activities. This could be useful if two blocks must be built next to each other. Both blocks will then be considered as only one;
- Snap tool to facilitate positioning of blocks;
- Etc.



**Fig. 160: Hierarchical tree structure to manage activities**

Notice that there is the possibility to have a three-dimensional view of the workshop as we can see in Fig. 161.



**Fig. 161: 3D view of the workshop**

Even without sophisticated algorithms a well-designed interface can be a powerful tool to facilitate schedulers' work.

### 5.2.3 Theory developments of the optimisation

Up to this point we have shown all the necessary tools to retrieve the information and to simulate a certain production planning, but this planning was still set up manually.

The goal of this study was actually to find out automatically the best planning. For a list of given blocks we have to allocate the biggest amount of blocks – a solution with all the blocks does not always exist. A specific tool has been developed for this.

This chapter will first formalize the optimization problem and detail the technique used for this space allocation optimization problem from a more scientific point of view. After that, we explain more practically how to use the optimizer and give some suggestions.

Detailed explanations of the algorithm have already been given in the paper *Optimization of Surface Using Heuristic Approaches* by Langer, Bair & al (Compit 2005) but are given in the next sections for the sake of completeness.

#### 5.2.3.1 Optimization algorithm

##### *Objectives*

The aim of this section is to present the method developed to solve the scheduling problem of space allocation. Remember that the factory is divided into areas, the blocks produced in the factory are very large, and, once a building block is placed in the factory, it cannot be moved until all the processes on the building block are finished. The blocks cannot also overlap. The objective is to maximize the number of building blocks produced in the factory during a certain time window.

##### *Problem formalization*

More precisely, we are given a set of  $n$  rectangular-shaped blocks. Each block is characterized by its geometric dimensions (width  $w_j$ , length  $l_j$  and height  $h_j$ ) but also by processing information such as its processing time  $t_j$ , its ready time  $r_j$  and its due date  $d_j$  ( $j$  in  $\{1, \dots, n\}$ ). We are also given a number  $A$  of identical two-dimensional areas, having width  $W$  and length  $L$ . Time is considered as a third dimension. The areas are fully dedicated to the production of the blocks.

The problem we are faced with consists of orthogonally ordering the blocks into the areas, while respecting the time constraints, and with the objective to produce the largest

number of building blocks. In practical terms, this means that we have to assign six variables to each block  $j$ :

- $p_j = \{0,1\}$  indicating whether the block  $j$  is produced or not;
- the name  $a_j = \{1, \dots, A\}$  of the area where block  $j$  is to be produced;
- $x_j$  and  $y_j$  coordinates, representing the position of the upper left corner of the block  $j$  in the area;
- an orientation  $o_j = \{0,1\}$  (either horizontally or vertically) for block  $j$ ;
- a starting date  $s_j$ ;

A solution will be considered as feasible if the individual and the collective constraints are met. We call individual constraints those which focus on one block only, regardless of the other blocks. Major individual constraints are represented by the fact that<sup>23</sup>:

- blocks must fit within the width of an area ( $x_j \geq 0$  and  $x_j + [o_j w_j + (1-o_j) l_j] \leq W$ )
- blocks must fit within the length of an area ( $y_j \geq 0$  and  $y_j + [o_j l_j + (1-w_j) l_j] \leq L$ )
- blocks must fit in their time windows ( $s_j \geq r_j$  and  $s_j + t_j \leq d_j$ )

Collective constraints focus on the interaction between the positions of different blocks. As a first step, the only collective constraint considered is that we need to prevent the blocks from overlapping.

We will also assume in the first sections that there exists at least one feasible solution for the set of blocks initially given. In other words, this means that all the constraints can be satisfied when  $p_j=1$  for all  $j$  in  $\{1, \dots, n\}$ .

To explain the algorithm developed for the entire problem, we will show the method that leads to one of these feasible solutions. Our technique has been largely inspired by techniques initially developed for a very similar problem called the *Three-Dimensional Bin Packing Problem* – see Fig. 162. We consider the problem without the assumption of feasibility, using the first method to assess whether a subset of blocks is feasible or not. Then, we describe how additional real-life issues are integrated.

---

<sup>23</sup> We will show in a later section how additional individual constraints are easily integrated.

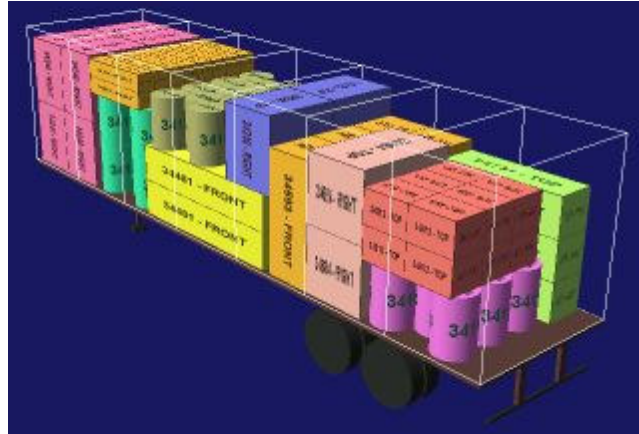


Fig. 162: The three-dimensional bin packing problem

### *Analogy to the 3D-BPP*

In the Three Dimensional Bin Packing Problem (3D-BPP), we are given a set of  $n$  rectangular-shaped items, each characterized by width  $w_j$ , height  $h_j$ , and depth  $d_j$  ( $j$  in  $\{1, \dots, n\}$ ) and an unlimited number of identical three-dimensional containers (bins) having width  $W$ , height  $H$ , and depth  $D$ . The three-dimensional bin packing problem (3D-BPP) consists of orthogonally packing all the items into the minimum number of bins.

The major difference between 3D-BPP and our initial problem is that, in the former, items/blocks must fit into the container height ( $z_j \geq 0$  and  $z_j + h_j \leq H$ ), whereas they must fit into their time window in the latter ( $s_j \geq r_j$  and  $s_j + t_j \leq d_j$ )<sup>24</sup>. The differences between minimizing the number of bins and maximizing the number of blocks will not complicate the formulation, since we will assume, at least for the first sections, that there exists at least one feasible solution for a fixed number of block and of areas/bins.

The 3D-BPP is strongly NP-hard<sup>25</sup>. Indeed, it is a generalization of the well-known one-dimensional bin packing problem (1D-BPP), in which a set of  $n$  positive values  $w_j$  has to be divided into the minimum number of subsets so that the total value in each subset does not exceed a given bin capacity  $W$ . It is clear that 1D-BPP is the special case of 3D-BPP arising when  $h_j = H$  and  $d_j = D$  for all  $j$  in  $\{1, \dots, n\}$ , and it has been proven<sup>26</sup> that the 1D-BPP is NP-

---

<sup>24</sup> One can compare a bin in the 3D-BPP to a timeline of the two-dimensional representation of an arc, which gives us a three-dimensional representation of the problem.

<sup>25</sup> See Garey and Johnson (1979) for more information about the complexity of combinatorial optimization problems.

<sup>26</sup> Coffman & al. (1997)

Hard. For such difficult problems, one way to prevent combinatorial problems flaring up is to allow algorithms to reach fairly good solutions, without guaranteeing that the best possible solution is reached. Some of the best known methods which use this strategy are local search heuristics.

Faroe & al. proposed in 2003 a "*New Heuristic for 3D-BPP*". Their method offers a huge degree of flexibility so that it can be adapted to various additional constraints. Therefore it fits perfectly to the way our problem presents itself, and to most of the additional real-life issues described in the section "*Additional real-life issues*".

### ***Finding feasible solutions***

#### **General approach**

The local search heuristic proposed to find a feasible schedule strictly enforces the individual constraints only; in other words, all the solutions generated by the heuristic respect the constraints associated with each individual block. Then, penalties linked to the collective constraints are summed up in an objective function that is minimized. With no additional real-life collective constraints, the objective function value of a given solution is the total pairwise overlap between the blocks. Therefore, with a randomly generated unfeasible solution where blocks can overlap, searching for a feasible solution is equivalent to minimizing the objective function, since an objective value of zero indicates that the collective constraints are also met. For any solution  $X$ , let  $overlap_{ij}(X)$  be the overlap<sup>27</sup> between blocks  $i$  and  $j$ . The objective function can now be formulated as

$$f(X) = \sum overlap_{ij}(X) \text{ with } i < j$$

Given a solution  $X$ , we can redefine the neighbourhood  $v(X)$  proposed by Faroe & al. (2003) as *the set of all solutions that can be obtained by translating any single block along the coordinates axes and the timeline, or by a move to the same position in another area, or by a +/- 90 degree rotation of a block around one of its four corners*. A neighbour of  $X$  is therefore constructed by assigning a new value to one of the variables  $x_j, y_j, s_j, a_j, o_j$ . It is clear that this definition of a solution space includes all feasible schedules and that there is a path of movements between every pair of solution.

---

<sup>27</sup> In square meters day.

A typical local search procedure proceeds by moving from the current solution  $X_p$  to a neighbouring solution  $X_{p+1}$  in  $v(X_p)$  whenever this move improves the value of the objective function. This may lead to two types of difficulty. First, the solution may settle on a local minimum<sup>28</sup>. Several standard methods, such as the Simulated Annealing (Aarts & Korst 1989) or the Tabu Search (Glover 1990), exist to avoid this well-known shortcoming of local search procedures. Secondly, the neighbourhood of any given solution may be quite large (even if continuous, variables like  $x_j$ ,  $y_j$  or  $s_j$  can be discrete for practical purposes). Therefore, exploring the neighbourhood to find an improved move can be very costly in computing time. To deal with these issues, we present here an application of the *Guided Local Search* (GLS) heuristic, and its accompanying neighbourhood reduction scheme called *Fast Local Search* (FLS).

### Guided local search

The Guided Local Search Heuristic (GLS) has its roots in a Neural Network architecture named GENET, developed by Wang and Tsang (1991), which is applicable to a class of problems known as *Constraint Satisfaction Problems*. The actual GLS version, with its accompanying FLS, was first demonstrated by Voudouris (1997) and Voudouris & Tsang (1997, 1999), and finally applied to the 3D-BPP by Faroe & al. (2003).

Basically, GLS increases the objective function of a problem to include a set of penalty terms and considers this function, instead of the original one, for minimization by the local search procedure. A local search is confined by the penalty terms and focuses its attention on promising areas of the search space<sup>29</sup>. Iterative calls are made to a local search procedure, denoted as *LocalOpt(X)*. Each time *LocalOpt(X)* gets caught in a local minimum, the penalties are modified and a local search is again demanded to minimize the modified objective function. To a certain extent, the heuristic may be classified as a Tabu Search heuristic; it uses memory to control the search in a similar way to Tabu Search.

GLS is based on the concept of *features*, a set of attributes that characterizes a solution to the problem in a natural way. In our adaptation of the model, the features are the overlaps between the blocks, and we denote by  $I_{ij}(X) = \{0,1\}$  the indicator whether blocks  $i$  and  $j$

---

<sup>28</sup> States which are better than all the neighbours but not necessary the best possible.

<sup>29</sup> Voudouris and Tsang (1999).

overlap or not. In a particular solution, a feature with a high overlap is not attractive and may be penalized. As a result, the value of  $overlap_{ij}(X)$  can measure the impact of a feature on a solution  $X$ <sup>30</sup>.

The number of times a feature has been penalized is denoted by  $p_{ij}$ , which is initially zero. Loosely speaking, we want to penalize the features with the maximum overlap that have not been penalized too often in the past. The source of information that determines which features will be penalized should thus be the overlap and the amount of previous penalties assigned to the features. For this purpose, we define a utility function  $\mu(X) = overlap_{ij}(X)/(1+p_{ij})$ . After each  $LocalOpt(X)$  iteration, the procedure adds one to the penalty of the pairs with maximum usefulness.

After incrementing the penalties of the selected features, they are incorporated into the search with an augmented objective function

$$h(X) = f(X) + \lambda \sum_{i,j} (p_{ij} \cdot I_{ij}(X)) = \sum_{i < j} overlap_{ij}(X) + \lambda \sum_{i,j} p_{ij} \cdot I_{ij}(X)$$

where  $\lambda$  is a parameter – the only one in this method – that has to be chosen experimentally. Thus, when local search has found a solution  $X^* = LocalOpt(X)$ , overlaps with maximum utility are penalized and become undesirable. In a sense, the search procedure is ordered to set a priority on these features and, for this reason, it jumps out of the local minimum.

### Fast local search

Let us now describe the so-called *fast local search* procedure<sup>31</sup>. FLS is used to transform a current solution  $X_{cur}$  into a local minimum  $X^* = LocalOpt(X_{cur})$ . It will help us to reduce the size of the neighbourhood with a selection of the moves that are likely to reduce the maximum utility overlaps.

We define the sets  $v_m(X)$  as subsets of the neighbourhood  $v(X)$  where all solutions in  $v_m(X)$  only differ from  $X$  by the value of the variable  $m$  ( $m = \{x_j, y_j, t_j, a_j, o_j\}$  with  $j$  in  $\{1 \dots n\}$ )<sup>32</sup>.

---

<sup>30</sup> Referred as “cost function” in Faroe & al. (2003)

<sup>31</sup> See Voudouris & Tsang (1997) and Faroe & al. (2003).

<sup>32</sup> In the case of  $m = \{o_1, \dots, o_n\}$ ,  $v_m$  also includes the particular change in  $x_j$  and in  $y_j$  that considers a rotation around the four corners of a block. To simplify the explanation, this technical issue is not detailed.



The neighbourhood  $v(X)$  is thus divided into a number of smaller sub-neighbourhoods that can be either active or inactive. Initially, only some sub-neighbourhoods are active<sup>33</sup>. FLS now continuously visits the active sub-neighbourhoods in random order. If there exists a solution  $X_m$  within the sub-neighbourhood  $v_m(X_{cur})$  so that  $f(X_m) < f(X_{cur})$ , then  $X_{cur}$  becomes  $X_m$ ; otherwise we suppose that the selected sub-neighbourhood will provide no more significant improvements at this stage, and thus it becomes inactive. When there are no active sub-neighbourhoods left, the FLS procedure is stopped and  $X_{cur}$ , the best solution found, is returned to GLS. From a less formal point of view, FLS selects at random a variable  $m$  within a list of active variables, as long as this list is not empty. Then, it searches, within the domain of  $m$ , any improvement in the objective function. If it does not exist, the variable  $m$  becomes inactive and is removed from the list. By doing so, we focus especially on variables open to improvement.

The size of the sub-neighbourhoods related to the  $a_j$  and the  $o_j$  variables is relatively small, therefore FLS is set to test all the neighbours of these sets. But, on the other hand, using an enumerative method for the translations along the  $x$ ,  $y$  and  $t$  axis would become very expensive in terms of computing time, if area and/or time windows are large. We may, however, show that only certain coordinates of such neighbourhoods need to be investigated. If  $m$  represents  $x_j$ , changes in the overlap function only depend on  $x_j$  ( $h(X) = h(x_j)$ ). Most of the terms of this function are constant, thus, since we want to compare values, only the few terms dependent on  $x_j$  should be computed<sup>34</sup>. Also, it is obvious that  $overlap_{ij}(X) = overlap_{ji}(X)$ , so that the computing time of one solution is linear ( $n$ ) instead of quadratic ( $n^2$ ). Additionally, all functions  $overlap_{ij}(x_j)$  are piecewise linear functions, and therefore the functions will reach their minimum in one of their breakpoints (or at the limits of their domains). As a result, FLS only needs to compute the values of  $f(x_j)$  with  $x_j$  at breakpoints or at extreme values. In fact, there are at most four breakpoints for each function, and only the first and the last one are evaluated. Indeed, in regard to the analogy with the 3D-BPP, a good packing intuitively supposes that the boxes touch each other.

---

<sup>33</sup> We show at the end of this section how the selection is made to focus on the maximum utility overlaps.

<sup>34</sup> Furthermore, an overlap is the product of four partial overlaps (three for the overlaps on each of the  $x$ ,  $y$  and  $z$  axis, and the fourth equals one if  $a_i = a_j$ ; zero otherwise). Since we know that only the partial overlaps for the  $x$  axis depend on  $x_j$ , computing efforts can be reduced to their smallest size.

We have shown that FLS represents a relatively fast procedure that leads to a local minimum, if the amount of active sub-neighbourhoods is relatively small. Let us remember that  $LocalOpt(X)$  is called iteratively by GLS, and that penalties are changed with the objective of escaping local minima. Activation of sub-neighbourhoods should therefore allow moves on penalized features. The following reactivation scheme is used (Faroe & al. 2003): first, moves on the two blocks  $i$  and  $j$ , corresponding to the penalized features, are reactivated. Secondly, we reactivate the moves on all blocks that overlap with blocks  $i$  and  $j$ . The latter reactivation is added to allow FLS to pay attention not only to the two overlapping blocks but also to the whole area around the penalized feature.

### ***Selecting the blocks***

In the previous chapter, we described a method that minimizes the collective constraints under the restriction of the individual constraints and we supposed that there exists at least one feasible solution for the set of blocks initially given. Let us denote this procedure by  $GlobalOpt(X)$ . If  $GlobalOpt(X)$  is efficient, it should find a solution with an objective function of zero after a certain time and this solution would be one of the feasible solutions. However, in the initial formulation of the problem, we do not know whether a set of blocks is feasible or not. The combination of GLS and FLS can be used anyway if we rely on the following heuristic assumption: there exists no feasible solution if none is found within a certain amount of computing time  $T$ . Consequently, the search heuristic  $GlobalOpt(X,T)$  is utilized as a test of feasibility and gives the corresponding schedules if a feasible solution is identified within  $T$ .

Several methods have been tested using this concept. The objective was to remain as close as possible to the working methods and habits used in the factory under study. From this point of view, an efficient approach for the industrial application is to start GLS with a randomly generated solution  $X_0$  that includes the entire set of blocks ( $p_j = 1$  for all  $j = \{1, \dots, n\}$ ). After a search of  $T$  seconds, the algorithm is stopped and returns  $X_1 = GlobalOpt(X_0, T)$ , the best solution found (in terms of overlap). One of the blocks with the highest overlap is removed from the set ( $X_1 \rightarrow X'_1$ ) and the heuristic  $GlobalOpt(X'_1, T)$  is restarted. The entire procedure ends if a solution  $X_n$  with zero overlap is found.

A variant procedure is to start with an empty set  $X_0$  ( $p_j = 0$  for all  $j = \{1, \dots, n\}$ ). At each iteration, if the solution  $X_{n+1} = GlobalOpt(X_n, T)$  is feasible, then an additional block is inserted into the set; otherwise an overlapping block is removed. This procedure is stopped after a

certain amount of computing time, or by any more sophisticated stopping criterion, and returns the solution with the largest collection of blocks.

Both approaches suffer from one major default: they are likely to have an aversion to the largest blocks. Indeed, we do not have an appropriate weighting scheme to evaluate the preferences between blocks, and, since small blocks generally provide smaller overlaps, they are preferred to larger ones. In the real-life situation, when the entire set of block cannot be produced, the person in charge of scheduling can either subcontract specific blocks to other factories, or change some temporal parameters (e.g. intensify the workforce to reduce processing times or postpone due dates). No formal information can describe all the aspects of these choices. For this reason, the operator should be able to change manually the collection of blocks to be produced. Starting from our "fairly good" feasible solution  $X_n$ , iterative  $X_{n+1} = \text{GlobalOpt}(X_n)$  calls are ordered manually after deliberate changes ( $X_n \rightarrow X'_n$ ) in the assignment. In addition, a final procedure provides a list with each block that is not assigned even though a feasible solution that includes the block can be found.

By not regenerating solutions on a random basis, some of the information from previous solutions is preserved. Of course a drawback to this approach is that the structure of a previous solution can confine GLS to an area of the solution space that can be difficult to escape<sup>35</sup>. We may therefore not reach the very best solution. However, the modus operandi described in this section is developed for daily industrial use. In that setting, the above drawback may actually be viewed as an advantage. Indeed, it may be very costly for the company to mix up the schedules over and over again. Traditionally, methods for problems of similar classes utilize a construction algorithm during the search<sup>36</sup> and a slight improvement may disturb the whole solution; with GLS, non-problematic regions are not perturbed.

### ***Additional real-life issues***

Additional constraints may occur in any firm-specific situation. The tool proposed is easily customizable to most of them. For example, we may need to restrict or force the position of a block (e.g. a tool is only available in one area or the block is already in process).

---

<sup>35</sup> Faroe & al. (2003) suggest a similar problem in their approach for 3D-BPP.

<sup>36</sup> Murata & al. (1996) developed a tricky construction technique based on partial-orders coding scheme for the 2D-BPP. Imahori & al. adapted this approach for a problem very close to the one considered here.

Those constraints were called individual in the section *Problem formalization* and integrating them is trivial: restricted positions are not generated and unfeasible neighbours simply do not exist. As a result, the end-user may fix the value of any variable (including  $p_j$ ) or reduce its domain.

Specific collective constraints may also appear in the wording of a problem. In our case, the areas of the factory have one single door, and the crane bridge can only carry blocks up to a certain height. As a result, a large block may, for example, obstruct a door, and some blocks might not be deliverable in time because there is no route to transport them out. We dealt with this issue in the same way as for overlaps. For each generated solution  $X$ , we add to the objective function  $h(X)$  a new term accounting for exit difficulties:

$$g(X) = h(X) + ExitProblems(X) = \sum_{i < j} overlap_{ij}(X) + \lambda \sum_{i,j} p_{ij} \cdot I_{ij}(X) + ExitProblems(X)$$

The *ExitProblems* values should somehow be scaled to the  $h(X)$  values, but there is no need to find a precise weight for this term, because the procedure is designed to find a zero objective function and weights are less relevant in this case.

We believe that other collective constraints may be included in the algorithm using this approach.

### 5.2.3.2 Implementation of the algorithm

The methodology of the algorithm has been explained. The implementation has been done in Java as for the user interface. In practical terms the scheme is the following one. Firstly we analyse the workshop and allocated blocks. If the initial situation contains overlaps we first try to find an acceptable solution. In the worst case we have to remove some blocks. Once a solution is found we will add a new block into the workshop and try to find an authorized position. If no solution is found we have to remove a block and try with another one. Which block can we move? Of course fixed blocks cannot be removed. But for others many possibilities remain:

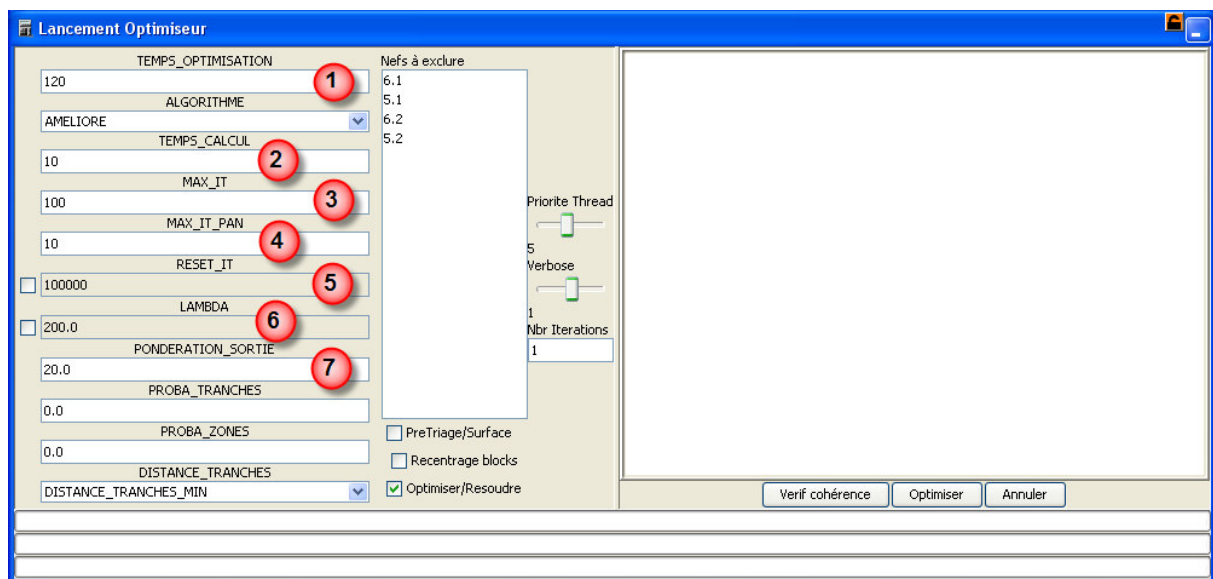
- Moving the last block placed;
- Detect the overlap of each block and remove the one with the biggest overlap;
- Detect overlapping blocks and remove one randomly;
- Etc.

After several tests, we have chosen another methodology. We detect all overlapping blocks. Then we remove the worst block but we straightaway do another overlap evaluation. Again we detect all overlapping blocks and we remove the worst block. We continue this process until we have a solution without an overlap. It means that we can remove several blocks at the same time. This solution is the one that gives the best results in the shortest time.

Little by little we increase the number of blocks in the workshop.

Which block to add when we have found a solution? Here again we have several possibilities. In fact from the interface the user has the possibility to sort blocks by multiple factors. For example we can sort them by surface area from the biggest one to the lowest. In that case the optimizer will first try the biggest block. Then the second one, and so on. When we remove a block we will not try to add it again until all the blocks have been tested at least once. Once this is done, blocks are chosen randomly. To summarize we test each block in the order specified by the user and after we select blocks randomly.

Some parameters – see Fig. 163 – have been added during the development:



**Fig. 163: Parameters of the Optimization**

- 1 : *Total Time (TEMPS\_OPTIMISATION)*: once this time has elapsed the optimization is stopped even if the optimum is not yet reached;
- 2 : *Time allowed for the local search (TEMPS\_CALCUL)*: after this time if we have not solved the local problem we will try with another block;
- 3 : *Maximum number of iterations for the local search (MAX\_IT)*: same as the previous parameter but after a number of iterations – not a time. If this number is high, the local search will always be ended by the specified time. If this number is low, this parameter will be the most influential;
- 4 : *Maximum number of block trials (MAX\_IT\_PAN)*: in other words we count the number of times we tried to allocate a block and did not achieve the objective. Once

this number is equal to the parameter we definitively give up this block and fix it into the bin;

- 5 : *Parameter to initialize penalty function (RESET)*: when the number of iterations reaches this parameter we reset the penalty for each block. This is not a fundamental parameter but after a huge number of iterations, penalties applied during the first one are not yet very significant. So it could be judicious to reset all penalties after a time;
- 6 : *Lambda*: this is the only real parameter influencing the algorithm's behaviour;
- 7 : *Exit factor (PONDERATION\_SORTIE)*: this the weighting factor to manage exit problems. As explained earlier, this factor is not really important for the optimizer.

There are still some parameters (*PROBA\_TRANCHES*, *PROBA\_ZONES*, and *DISTANCE\_TRANCHES*) but they are not used for all workshops. Some blocks could have preferential areas or needs to build next blocks of the same family. We thus have special position restrictions. In this case a priority problem arises: is it better to put a maximum number of blocks even if they do not respect these restrictions, or is it better to satisfy them even in reducing the number of blocks allocated? These parameters are weight parameters to decide to take into account (or not) the restrictions.

Note that we also have the possibility of selecting the working areas to be used for the optimization. We can also find a solution for a given situation without adding new blocks. Other smaller options are available but they will not be explained in detail.

In Fig. 163 we also have a window to get information of the optimisation process.

## 5.2.4 Applications

### 5.2.4.1 Introduction

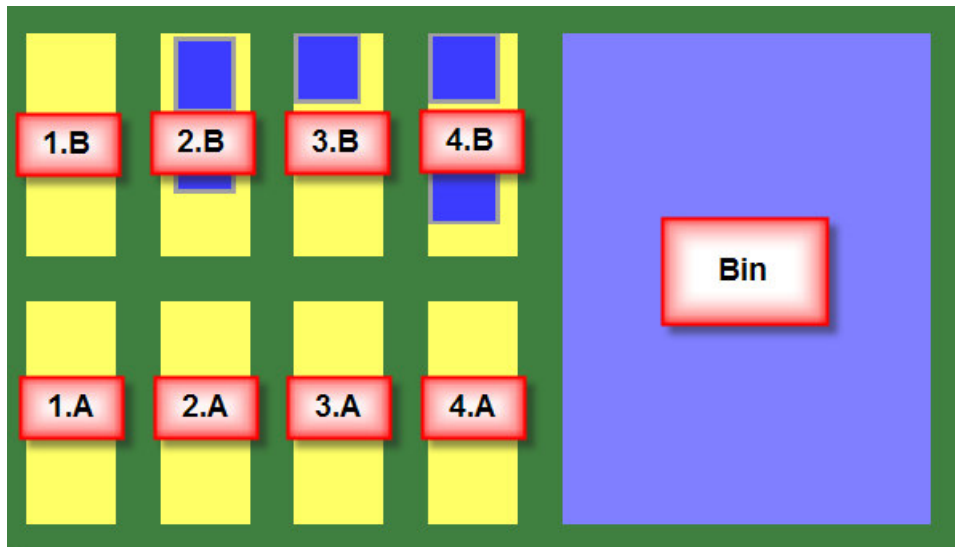
In the Aker Yards France shipyard of Saint-Nazaire the tool is already used in three different workshops. This is a proof of its efficiency! We will briefly describe each of them and their specificities to show the potential applications of the tool. Now we have to keep in mind that a totally new workshop can be modelled without difficulty.

The performances of the tool are not easily presentable on paper, the real interest can clearly be felt when we use it!

### 5.2.4.2 120 tons workshop

The workshop is used to build blocks before being mounted next to the dry dock in the mounting area. Most of these blocks should be built in the 180 Tons workshop but the smallest ones are built in this workshop.

This workshop has eight different working areas – see Fig. 164.

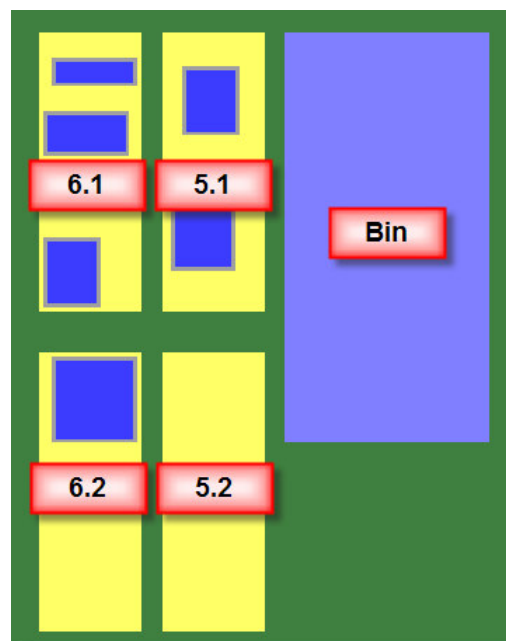


**Fig. 164: The 120 Tons workshop**

This workshop is the most “classical” one with no specific scheduling rules. Exit doors are all located in the middle horizontal lane that marks the separation between working areas A and B.

#### 5.2.4.3 180 tons Workshop

This workshop is very similar to the previous one. It contains only 4 working areas named 5.1, 5.2, 6.1 and 6.2 – see Fig. 165.



**Fig. 165: The 180 Tons workshop**

This workshop requires special scheduling. Although blocks also have an earliest starting date and a latest starting date we generally want to start at a specific date. In fact each

block has a preferential starting date. In the information block – Fig. 155 – we can specify this target date (*Cible* in French). If the optimiser has a choice it will try to start the block as close as possible to this target date. Adaptation has thus been done in the optimisation algorithm.

#### 5.2.4.4 Pre-Mounting area

Many particularities can be found in this workshop. First of all the scheduling is simpler for two reasons: allocation is done in almost one direction due to the shape of the working areas – see Fig. 166 – and the earliest starting date is always equal to the latest starting date. In other words the problem is reduced to a two-dimensional problem!

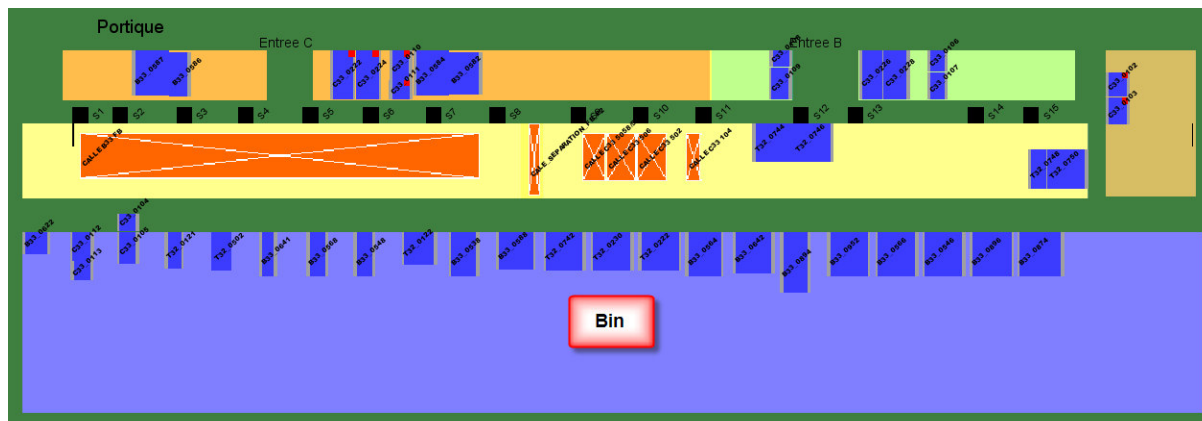


Fig. 166: The dry dock and the Pre-Mounting area

However other functionalities had to be added. Firstly the scheduling could concern blocks for different ships. Planners wanted to have a specific industrial calendar for each ship. This point was really more difficult to solve than it might seem. Secondly, blocks have preferential areas. This constraint involves adapting the optimisation algorithm. Thirdly, blocks can be divided up in “families”. The new constraint is that blocks of the same family should preferentially be allocated next to each other.

Furthermore the number of blocks to be allocated is higher than for previous workshops. This single point makes manual scheduling very complicated without the developed software.

Notice that for each workshop several graphical results are available. For instance we can have the relative occupation in each working area – see Fig. 167.



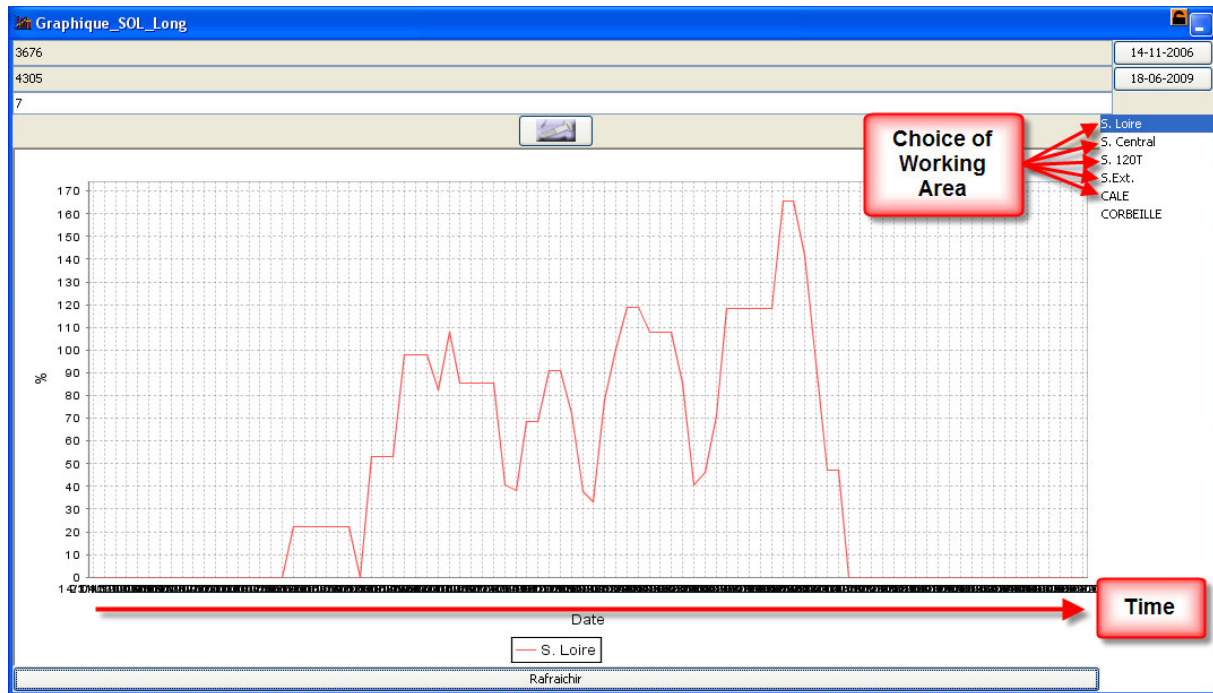


Fig. 167: Relative occupation of Working Areas

### 5.3 Conclusion

The power of the tool can be summarized by just one comment made by the scheduler of the 120 tons workshop: after three days of using the tool, he did the work he usually does in two weeks! And these three days included being introduced to the software for the first time and learning how to understand it! The use of this software could also have unexpected results. For example in the Aker Yards France shipyard where the tool is used, the division of a ship has been reviewed because the planner of the 180 Tons workshop predicted scheduling problems in the workshop with the division chosen – and that, thanks to our tool, has been carried out!

One great quality of the tool is its flexibility. With few changes it is used for three different workshops. The XML interface to define the workshop is very useful in creating new workshops.

The optimizer is also very powerful. Solutions can be found in a very short time. An important point is that we always stay in the acceptable space of solutions. We can stop the software after five minutes and the solution obtained will be feasible. If the number of blocks allocated is not high enough, we can resume the optimization! At the end, manual changes are of course always possible thanks to the diversity of tools contained in the software! Indeed there are often some special restrictions that only the user knows and that an optimizer could not guess. Consequently a manual retouch is often needed.

Nevertheless there are numerous ways of improving the software. Probably the most important possibility is to extend the software to non parallelepiped products! However this is a major change that requires a complete revision of the optimizer methodology.

# Chapter 6

## Conclusion

### 6.1 Direct results

The results obtained in the framework of this thesis can be classified into two categories.

Firstly there are the “direct” results. In other words those that can be used straightaway by someone else. The simulation model being completely finished and developed, it could be used straightaway to help scheduling and to optimize production. However the use is obviously limited to this workshop. This is not really true because we have to take into account the “object oriented” nature of all the developments done. The consequence is that most work done can be used for other simulation models. The simplest example is the crane bridge: most workshops use at least one and the crane bridge object of our simulation could be used in other simulation models. In this case, there are only slight changes to be made for a new model: only simple parameters – and also the icon – have to be changed such as horizontal and vertical speed, accelerations, handling equipment, etc.

In Chapter 4 the PERT tool developed is the most flexible tool of the thesis. Each project – small or large, at a local or global level, in shipbuilding or not etc. – can draw benefits from the tool by a clear highlighting of critical activities. Although it has been linked to the simulation model, its independence is total.

Chapter 5 also led to a very robust tool that can be used without major changes in many other plants. This space allocation solver is above all a scheduling tool and not only a nesting tool. We have shown its versatility in applying it to the three different workshops of a French shipyard. It is clear that its outlets are not limited to shipbuilding industries. Other possible applications are storing areas! They meet exactly the same characteristics as the workshops studied: space is limited and scheduling must be optimized. Potentially the tool thus has diversified and numerous markets.

### 6.2 Indirect Results

Other “indirect” results can be drawn from our work. Here we mean that results cannot be exploited directly but the methodology used can be copied and adapted for other problems. Chapter 3 on the development of the simulation specifically belongs to this category. Indeed

models can be used only for the specific workshop – the PrePreFabrication workshop – even if, as mentioned above, some objects can directly be used on other projects. In this chapter the attention must be focused more on the methodology used and mainly on result possibilities. For example we have shown the importance of performing statistical analyses when developing a simulation model.

Chapter 4 and Chapter 5 have led to exportable products.

### 6.3 Conclusion

The challenge of building a model of a shipbuilding workshop from scratch was huge. The specificities of the workshop have been widely highlighted in Chapter 3. They showed the complexity of the task. Working on a complex workshop – with success – has guaranteed that the methodology used throughout the chapter can be exploited and used again for simpler systems. It seems obvious that simulation of other workshops of the shipyard – and generally of other shipyards too – will not encounter difficulties more insurmountable than those met in this thesis. In particular this is still true for the optimization process! As explained in the thesis, simulation can offer three levels of information. The first analysis gives us important information about bottlenecks within the system, its weaknesses and strengths. Problems are clearly highlighted and corresponding solutions are given. The second analysis is the statistical approach to the workshop. By modifying values of the parameters we can observe the impact of each individual operation on the total production time. Here we glean important information about how to modify bottlenecks within the system! Thirdly we can carry out an optimization of the workshop. The production sequence is optimized to improve workshop productivity! If we have shown that the optimization does not lead to a global optimum we gain about 10% of the total production time. Weaknesses of the optimization are due to the particular functioning of the workshop and also to the algorithm itself, and this can probably be improved.

In addition a complete and independent tool has been developed to manage PERT diagrams. The software is really convenient to use and systems can easily be modelled due to the user-friendly graphical interface. Information on the model can be obtained straightaway: critical activity, bottleneck, earliest and latest starting/ending dates at each step, etc. An integrated optimizer can automatically optimize the network to minimize the total cost. Furthermore we can make a parametric study to fix the total time of the system in taking into account the cost of each activity! Important application has been done in linking the software

to the simulation model developed. Consequently new important information is available for the PrePreFabrication workshop. The main advantage is seeing immediately which kits and which operations cannot suffer any delay without having an influence on the total production time. Chapter 4 describes in depth the development of this PERT tool.

Space allocation problems are successfully solved in Chapter 5. The software developed can reduce a lot the time needed to schedule workshops that face these problems. Furthermore this scheduling is also better now. The use of the software in the three workshops is the proof of its utility!

Various useful tools have been developed using different kinds of optimization: from genetic algorithms to simpler methods such as the Revised Simplex Method and also to heuristic optimization. Each problem has its own characteristics; it implies that each solution is unique and adapted.

## 6.4 Perspectives

In production, there are numerous simulation perspectives. Simulation can be used at different levels: it could be used several months before the real production or just day by day. These kinds of simulation are different because they need different levels of detail. In a long term simulation we are more interested by the general production flow in the overall shipyard. One condition to getting such a simulation is thus to model the entire shipyard i.e. each workshop. Because we are working long term, probably not all the necessary data will be available. Consequently the workshop's modelling will be less detailed. One interesting perspective could be to simulate the shipyard at this time window – several months before production – to rapidly have information about future workloads. The sooner we can predict problems the more solutions we will find to solve them – at a lower price. Modelling each workshop at a low level of detail could be very interesting to optimize the general production of the workshop. Ideally the best is to have two simulation models for each workshop: one very detailed – as in this thesis – to optimize scheduling over a short term and one less detailed for the simulation of the set. The modelling of all workshops is a real contribution to improve the interest of the simulation. For instance one lack in the developed model is that we do not take into account the delays of supplier workshops! If all workshops are modelled we can study a specific workshop – such as the PrePreFabrication workshop – and link it to its supplier workshops which are modelled with the simple model. This modelling increases

notably the credibility of our studied model! As we can see, multiple perspectives are open for simulation in the shipyard!

The PERT tool is an independent tool that has a lot of possible future applications. Even if minor changes can improve the tool there is no need to improve it to increase its possible fields of application.

For the space allocation solver tool many perspectives are open. Here again the tool could be used already in many fields but improvements are possible. One important constraint of the software is the rectangular shape of the products. An extension could be to generalize the problem to all kinds of products of any shape. Such an extension is not insignificant because it implies changing completely the heuristic used for the optimization. A new methodology has to be found! Another addition is to take into account better the three geometrical dimensions of blocks. Currently we take it into account for “exit” problems: the optimizer looks if the block can be moved away with a crane without being hampered by other blocks. But the check is done in considering only direct movements from the position of the block to the exit door without trying to find a better way. Getting round this problem could improve the quality of solutions. Another change is to really take the third dimension at the same level as the other two. In many storing areas products are stored in taking into account the three dimensions. In that case we would have a real four dimensional problem! This last perspective is not necessary for shipyards that do not face this kind of storing but is still very interesting in terms of applying the tool to other industries. Changes must be made to the software but they seem to be not insurmountable.

As we can see there are various perspectives and this thesis has provided a good basis for further interesting developments.

# Glossary

Here is a list of special words encountered in this thesis. A more detailed explanation is given for each word where they are introduced in the thesis.

- *AS3*: Specific part of an assembly. Hierarchically the AS3 is just above the AS4 and below the assembly;
- *AS4*: Specific part of an assembly. Hierarchically the AS4 is just above the AS5 and below the AS3;
- *AS5*: Specific part of an assembly. Hierarchically the AS5 is the lowest sub-assemblies;
- *BDRO*: This is an ERP software user by the Aker Yards shipyard;
- *Cvs*: Comma Separated Value. Specific file where data is divided in fields – without any format – divided in fields separated by comma. This is the simplest way to store data;
- *DBREF*: Cvs file containing blocks information for the space allocation tool developed;
- *DCH*: Area in the simulation of the PrePreFabrication workshop where PMs containers are located;
- *ERP*: Enterprise Resource Planning. Kind of software used for planning and scheduling;
- *MAT*: Area in the simulation of the PrePreFabrication workshop used to stock temporarily plates, long girders or finished assemblies;
- *Nesting*: Operation that consists in minimizing the amount of scrap raw material produced by cutting operation to get manufacturing parts;
- *Mechanized gripper*: Handling equipment set up on tracks and that can be used by workers to handle heavy pieces and to tack them;
- *PERT*: Program Evaluation and Review Technique. This is a model for project management designed to analyze and represent the tasks involved in a given project;
- *Palanqué*: Small plates used in the PrePreFabrication workshop. It can be a basic plate and it could be joined on the joining area.

- *PHL*: For “*Programmation Hors Ligne*”. Process that consists to program *welding robot* for a given assembly. After the process detailed moves of robot’s arms are known and in consequence the estimation of the welding time is known;
- *PM*: Container used (by pair) in the PrePreFabrication workshop to bring long plates and to evacuate finished assemblies;
- *PRS*: Special long girders used in the PrePreFabrication workshop;
- *RAB*: Area in the simulation of the PrePreFabrication workshop where joining operation is done;
- *STK*: Area in the simulation of the PrePreFabrication workshop where small containers are produced;
- *Sub-panel*: Term used in this thesis to design fragment of a panel. Assemblies – coming from the PrePreFabrication – are grouped in function of the sub-panel to which they belong;
- *TCP*: containers used in the PrePreFabrication workshop to bring long girder;
- *UPC* (UPCA or UPCB): small girders used in the PrePreFabrication workshop;
- *UPS*: girders used in the PrePreFabrication workshop;
- *Variate*: a variable quantity that is random;
- *VBA*: Visual Basic for Application;
- *XML*: Extensible Markup Language. This is a generic framework for storing any amount of text or any data whose structure can be represented as a tree;



## Table of Figures

<i>Fig. 1: Magnetic Field Simulation</i> .....	14
<i>Fig. 2: Simulation of Automatic Robots</i> .....	14
<i>Fig. 3: Simulation of the descent of Opportunity Rover into the Victoria crater (on Mars)</i> .....	15
<i>Fig. 4: Simulation of the flow around a submarine by using fluid mechanical theory</i> .....	15
<i>Fig. 5: Microscopic Traffic Simulation</i> .....	15
<i>Fig. 6: Hierarchical view of Object Oriented Concept Example</i> .....	24
<i>Fig. 7: Discrete Event Simulation Concept</i> .....	27
<i>Fig. 8: List of events – simple example</i> .....	28
<i>Fig. 9: List of events</i> .....	28
<i>Fig. 10: Assembly sequence of pieces</i> .....	36
<i>Fig. 11: Machining of plates</i> .....	37
<i>Fig. 12: Machining of short profiles</i> .....	38
<i>Fig. 13: Machining of special profiles</i> .....	38
<i>Fig. 14: Fabrication of reconstructed welded profiles</i> .....	39
<i>Fig. 15: Workshop of forming and coiling</i> .....	39
<i>Fig. 16: Workshop of PrePreFabrication</i> .....	40
<i>Fig. 17: Panel Lines</i> .....	40
<i>Fig. 18: 120 Tons Workshop</i> .....	41
<i>Fig. 19: 180 Tons Workshop</i> .....	41
<i>Fig. 20: Area of Pre Montage</i> .....	42
<i>Fig. 21: Typical assemblies</i> .....	46
<i>Fig. 22: Non automatable assemblies</i> .....	46
<i>Fig. 23: Views of the PrePreFabrication Workshop</i> .....	48
<i>Fig. 24: Schematic view of the workshop</i> .....	49
<i>Fig. 25: Mechanized gripper</i> .....	49
<i>Fig. 26: Condition to saturate the welding robot</i> .....	53
<i>Fig. 27: Example of a KIT table</i> .....	58
<i>Fig. 28: Scheme of containers determination</i> .....	60
<i>Fig. 29: Scheme of the YL_KIT_CHILD request</i> .....	61
<i>Fig. 30: Scheme of the YL_STEELM request</i> .....	62
<i>Fig. 31: Scheme of the YL_STEPRT request</i> .....	63
<i>Fig. 32: Fields of the YL_KIT_TIME</i> .....	63
<i>Fig. 33: Requests to execute for a new treatment</i> .....	66
<i>Fig. 34: Simple interface to manage the database treatment</i> .....	67
<i>Fig. 35: The Frame Object</i> .....	70
<i>Fig. 36: The EventController Object</i> .....	71
<i>Fig. 37: The SingleProc Object</i> .....	72
<i>Fig. 38: The Source Object</i> .....	73

<i>Fig. 39: The Drain Object</i> .....	74
<i>Fig. 40: The Store Object</i> .....	74
<i>Fig. 41: The PlaceBuffer Object</i> .....	75
<i>Fig. 42: Icon of the Connector Object</i> .....	76
<i>Fig. 43: Example of a simple system modelled</i> .....	76
<i>Fig. 44: First step of the process</i> .....	77
<i>Fig. 45: Second step of the process – creation of a new MU</i> .....	77
<i>Fig. 46: Third step of the process</i> .....	78
<i>Fig. 47: State of the system after ten minutes</i> .....	78
<i>Fig. 48: Path Curved created by tracks objects</i> .....	79
<i>Fig. 49: Structure of the Frames of the PrePreFabrication</i> .....	86
<i>Fig. 50: Structure of frames</i> .....	87
<i>Fig. 51: The "Atelier" Frame</i> .....	89
<i>Fig. 52: Original icon of a frame</i> .....	90
<i>Fig. 53: The STK Frame</i> .....	95
<i>Fig. 54: MUs of the model</i> .....	96
<i>Fig. 55: TCP containers</i> .....	98
<i>Fig. 56: TCP containers in the simulation model</i> .....	98
<i>Fig. 57: The MAT Frame</i> .....	99
<i>Fig. 58: PM containers</i> .....	102
<i>Fig. 59: Structure of the DCH frame</i> .....	103
<i>Fig. 60: The DCH Frame</i> .....	106
<i>Fig. 61: The RAB Frame</i> .....	107
<i>Fig. 62: The "PostRAB" Frame</i> .....	109
<i>Fig. 63: The "Ilot" Frame</i> .....	111
<i>Fig. 64: Schematic representation of a cell</i> .....	112
<i>Fig. 65: The "Ilot_Demi" Frame</i> .....	114
<i>Fig. 66: The "Etat_Kit" Frame</i> .....	115
<i>Fig. 67: The "PosteAss" Frame</i> .....	117
<i>Fig. 68: Representation of the crane bridge</i> .....	119
<i>Fig. 69: Representation of a plate</i> .....	120
<i>Fig. 70: Representation of an assembly</i> .....	121
<i>Fig. 71: The "Pont" Frame</i> .....	122
<i>Fig. 72: Crane Bridge Handling Systems</i> .....	123
<i>Fig. 73: The "Prehenseur" Frame</i> .....	130
<i>Fig. 74: Representation of the Mechanized Gripper</i> .....	130
<i>Fig. 75: The "Robot" Frame</i> .....	132
<i>Fig. 76: Representation of the welding robot</i> .....	133
<i>Fig. 77: The "Interface" Frame</i> .....	134
<i>Fig. 78: The Main dialog box</i> .....	135

Fig. 79: The sub-panel selection button.....	136
Fig. 80: List of sub-panels available.....	137
Fig. 81: Selection of kits to be simulated for each sub-panel .....	138
Fig. 82: Parameters dialog box .....	138
Fig. 83: Dialog box related to joining areas .....	139
Fig. 84: Parameters relative to the Tacking operation.....	140
Fig. 85: Parameters related to the finishing operations .....	140
Fig. 86: Handling parameters.....	141
Fig. 87: WorkerPools parameters.....	142
Fig. 88: Parameters of assemblies' layout on working areas.....	142
Fig. 89: Times and their distribution used in the simulation .....	143
Fig. 90: Interface of initialization of the Workshop.....	144
Fig. 91: Constraint for kits on a half-cell .....	145
Fig. 92: Access to results .....	146
Fig. 93: The "Resource" Frame .....	147
Fig. 94: The ShiftCalendar Object.....	148
Fig. 95: Different possible states of an object.....	149
Fig. 96: The "Tables" Frame .....	150
Fig. 97: Total production time.....	152
Fig. 98: Dialog box of the Gantt.xls file .....	154
Fig. 99: Gantt diagram of the PrePreFabrication Workshop.....	155
Fig. 100: Graphic representation of time operation for a list of kits.....	157
Fig. 101: Occupation of workers for each cell .....	158
Fig. 102: Occupation of cranes bridges.....	158
Fig. 103: Occupation of the mechanized gripper for each half-cell .....	159
Fig. 104: Welding duration division .....	160
Fig. 105: Detailed operations done on a selected half-cell .....	161
Fig. 106: Occupation of the welding robot for each cell .....	162
Fig. 107: The "Result_Log" Frame.....	164
Fig. 108: The "Resultats" Frame .....	167
Fig. 109: Simple interface to manage the Access database treatment.....	169
Fig. 110: Parameters of the Initialization.....	170
Fig. 111: Initial situation of the workshop.....	170
Fig. 112: View of a part of the workshop during the simulation.....	172
Fig. 113: List of sub-panels to be simulated .....	173
Fig. 114: Impact of the Finishing time on the Total Production time.....	175
Fig. 115: Impact of the Finishing time on the Total Production time with several runs .....	176
Fig. 116: Impact of the Manual Tacking time on the Total Production time.....	177
Fig. 117: The "Optimisation" Frame.....	181
Fig. 118: Optimisation objects in the root frame.....	181

Fig. 119: Homogenization of the population .....	183
Fig. 120: Non homogenisation of the population.....	184
Fig. 121: Results for a simplified optimization: only one sequence out of two is optimized.....	188
Fig. 122: Random creation of the first generation.....	189
Fig. 123: List of chromosomes for an individual .....	189
Fig. 124: Examples of chromosomes .....	189
Fig. 125: Family table with parents and children individuals.....	190
Fig. 126: Fitness values of all children.....	190
Fig. 127: Individuals selected to create the third generation .....	191
Fig. 128: Radical homogenisation of a population.....	192
Fig. 129 : Example of a simple PERT network.....	197
Fig. 130: Characteristics of an activity .....	201
Fig. 131: Interface of the PERT Network Software .....	211
Fig. 132: Small network created with the software.....	212
Fig. 133: Activity attributes .....	213
Fig. 134: Node attributes .....	213
Fig. 135: Calendar to choose the start date.....	214
Fig. 136: Optimized network.....	215
Fig. 137: Results of a Parametric Study .....	215
Fig. 138: Evolution of Time/Cost for a selected event .....	216
Fig. 139: Influence of an activity on the total time .....	216
Fig. 140: Fabrication of a kit in a PERT network .....	218
Fig. 141: Representation of 2 successive kits .....	218
Fig. 142: Representation of a complete cell .....	219
Fig. 143: Fictive activities of a complete cell .....	219
Fig. 144: A complete representation of a cell (4 kits).....	220
Fig. 145: Shift of a half-cell due to high difference of warm-up activities.....	220
Fig. 146: XML structure to automatically generate PERT network .....	221
Fig. 147: Example of a PERT of a sequence of kits for the PrePreWorkshop.....	223
Fig. 148: Critical path for the Cell 3 .....	224
Fig. 149: Saturated Welding robot for a cell.....	224
Fig. 150: Not saturated Welding robot for a cell.....	224
Fig. 151: Examples of space limitation for some shipyards .....	229
Fig. 152: Height filter of the spatial overhead view .....	233
Fig. 153: Timeline view of the workshop .....	233
Fig. 154: Fake overlapping blocks .....	234
Fig. 155: Block's information.....	237
Fig. 156: Manual allocation of a block.....	237
Fig. 157: Overlaps detection tool .....	237
Fig. 158: Legend of blocks.....	238

<i>Fig. 159: Assigned blocks and virtual blocks .....</i>	<i>239</i>
<i>Fig. 160: Hierarchical tree structure to manage activities.....</i>	<i>242</i>
<i>Fig. 161: 3D view of the workshop .....</i>	<i>242</i>
<i>Fig. 162: The three-dimensional bin packing problem.....</i>	<i>245</i>
<i>Fig. 163: Parameters of the Optimization .....</i>	<i>253</i>
<i>Fig. 164: The 120 Tons workshop.....</i>	<i>255</i>
<i>Fig. 165: The 180 Tons workshop.....</i>	<i>255</i>
<i>Fig. 166: The dry dock and the Pre-Mounting area .....</i>	<i>256</i>
<i>Fig. 167: Relative occupation of Working Areas .....</i>	<i>257</i>

## References

### **Papers/Journal/Books:**

- AARTS E. AND KORST J.; (1989), *Simulated Annealing and Boltzmann Machines – a stochastic approach to combinatorial optimisation and neural computing*, Wiley.
- ALFELD, L. E.; PILLIOD, C. S.; WILKINS, J. R.; (1998), *The virtual shipyard: A simulation of the shipbuilding process*, Journal of ship production, pp.33-40
- BAIR F., LANGER Y., CAPRACE J., RIGO P.; (2005), *Modeling, Simulation and Optimization of a Shipbuilding Workshop*, COMPIT 2005, Hamburg.
- BANKS, J., CARSON, J.S., NELSON, B.L.; (1999), *Discrete-Event System Simulation*, Second Edition, Prentice Hall, Upper Saddle River, New Jersey.
- BIRK, L.; HARRIES, S.; (2003), *Optimistic: Optimization in Marine Design*, Mensch & Buch, Berlin.
- BOOCH; (1991), *Object-Oriented Design with Applications*.
- CHUNG, H.; LAMB, T.; SHIN, J. G.; (2000), *Generic shipyard computer model - a tool for design for production*, IMDC 2000.
- COFFMAN E.G., Garey M.R. and Johnson D.S.; (1997), *Approximation algorithms for bin packing: A survey*, D.S. Hochbaum editor, *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company Boston.
- COLEY, D.A.; (1999), *An Introduction to Genetic Algorithms for Scientists and Engineers*, World Scientific, Singapore
- FAROE O., PISINGER D. AND ZACHARIASEN M.; (2003), *Guided Local Search for the Three-Dimensional Bin-Packing Problem*, INFORMS Journal on Computing, Vol.15, No.3, pp. 267-283.
- GAREY M.R. AND JOHNSON D.S; (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York.
- GLOVER F.; (1990) *Tabu Search: A Tutorial*, Interfaces, Vol.20, No 4, pp.74-94.
- GOLDBERG; (1989) *Genetic algorithms in search, optimization, and machine learning*.
- GORDON, P; LEDENT J, (1980), *Modeling the dynamics of a system of metropolitan areas: a demoeconomic approach*, Environment and Planning A 12(2) pp.125 – 133.
- HAUPT, R.L.; HAUPT, S.E.; (1998), *Practical Genetic Algorithms*, John Wiley & Sons, Inc., New York.
- HILLIER; LIEBERMAN; (2001) *Introduction to operational research*, Seventh Edition.
- IMAHORI S., PISINGER D. AND VIGO D.; (2000), *Improved local search algorithms for the rectangle packing problem with general spatial costs*, Department of Applied Mathematic and Physics, Graduate School of Informatics, Kyoto.
- KARR, D. G.; BEIER, K. P.; NA, S. S.; RIGO, P.; (2002), *A framework for simulation-based design of ship structures*, Journal of ship production, pp.33-46.

- KOTESWARA, K.R., MADHUSUDHANA T.V.R., VEERABHADRA P., SUMENDER M.R., SIRUVURU SIVA SANKAR SHARATH; (2008), *Optimal Selection of PERT for large complex and distributed projects*, IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.6.
- LANGER Y., BAIR F., BAY M.; CAPRACE J., CRAMA Y. AND RIGO P.; (2005), *Optimization of surface using heuristic approaches*, COMPIT 2005, Hamburg.
- LEE, T.E.; SONG, J. S.; (1996), *Search-based heuristic algorithms for basic planning in a large shipyard*, Journal of ship production, pp.211-219
- LILLEY, D.; DEGRAW, K.; WALLEN, R.; (2001), *Development and implementation of computer simulation models for the manufacturing of outfitting components*, Journal of ship production, pp.16-26
- MAN, K.F.; TANG, K.S.; KWONG S.; (1999), *Genetic Algorithms*, Springer-Verlag, Londo.
- MARTELLO S., PISINGER D. AND VIGO D.; (2000), *The three dimensional bin packing problem*, INFORMS Operations Research, Vol.48, No. 2, pp. 256-267.
- MURATA H., FUJIYOSHI K., NAKATAKE S. AND KAJITANI Y.; (1996), *VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair*, IEEE transactions on computer-aided design of integrated circuits and systems, Vol. 15, No. 12, pp. 1518-1524
- PEGDEN., R E SHANNON, AND R P SADOWSKI.; (1995) *Introduction to simulation using SIMAN*, McGraw-Hill, 1995.
- PRASAD, V. R.; GRAUL, M.; BENJAMIN, P.; CAHILL, P. D.; MAYER, R.; (2000), *Resource-Constrained shop-level scheduling in a shipyard*, Journal of ship production, pp.65-75
- SASAKI, Y.; (2003), *Application of factory simulation to the shipyard*, COMPIT'03, pp.362-376.
- SCHOLL A., KLEIN R. AND JRGENS C.; (1997), *BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem*, Computers & Operations Research 24, pp. 627-645.
- SHEWCHUK, J.P. AND CHANG, T.C.; (1991), *An Approach to Object-Oriented Discrete-Event Simulation of Manufacturing Systems*, (Phoenix, Arizona, US) IEEE: 302-311.
- SHIN, J. G.; SOHN, S. J.; (2000), *Simulation-Based evaluation of productivity for the design of an automated workshop in shipbuilding*, Journal of ship production, pp.46-59
- SHIN, J. G.; LEE, K. K.; WOO, J. H.; LEE, J. H.; KIM, S. H.; *A modeling and simulation of production process at a virtual shipyard*,
- SLADOLJEV, Z.; (1996), *Search for a model of effective ship production management*, Journal of ship production, pp.220-229.
- STEINHAEUER, D.; (2003), *The virtual shipyard - Simulation in production and logistics at Flensburger*, COMPIT 2003, Hamburg.
- SMITH, R.D. ; (1999), *SIMULATION: The Engine Behind The Virtual World*, Volume I in the *Simulation 2000* Series.
- VOUDOURIS C.; (1997) *Guided local search for combinatorial optimization problems*, Ph.D. Thesis, Department of Computer Science, University of Essex, Colchester, UK.

- VOUDOURIS C. AND TSANG E.; (1997), *Fast local search and guided local search and their application to British Telecom's workforce scheduling problem*, Operations Research Letters 20, pp. 119-127.
- VOUDOURIS C. AND TSANG E.; (1997), *Guided local search and its application to the traveling salesman problem*, European Journal of Operational Research, No. 113, pp. 469-499.
- WANG C.J. AND TSANG E. (1991), *Solving constraint satisfaction problems using neural-networks*, Proceedings of IEE Second International Conference on Artificial Neural Network, pp. 295-299.

**Websites:**

- <http://www.akeryards.com;>
- <http://www.anast.ulg.ac.be;>
- <http://www.netmba.com/operations/project/pert;>
- [http://search.com.com/reference/Program\\_Evaluation\\_and\\_Review\\_Technique;](http://search.com.com/reference/Program_Evaluation_and_Review_Technique;)
- <http://www.wintersim.org;>

**Other references:**

- *Plant-Simulation Help v.7.0.11.*