

Czech Technical University in Prague
Faculty of Electrical Engineering

DIPLOMA THESIS



Anne-Laure Coiffier

Analysis and design of manufacturing operations

Department of Cybernetics

Supervisor: Ing. Pavel Burget, Ph.D.

Prague, May 2017

Declaration of Authenticity

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, dated:

.....
Signature

**Czech Technical University in Prague
Faculty of Electrical Engineering**

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Anne-Laure Coiffier

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Analysis and design of manufacturing operations

Guidelines:

1. Perform research in the literature how the production operations are defined and planned.
2. In the layout of the manufacturing line identify the production resources and their related operations. Define shared zones where several resources can operate simultaneously and plan the atomic operations with respect to the zones.
3. Design a mechanism of transforming the production operations for individual products into the framework of the production line to create and possibly generate automatically the sequences of operations. Take into account the shared zones and respective schedules when individual product need to be produced. Take into account the possibility to have different products in the production at the same time.
4. Implement the production operations in the available robots and other production resources.
5. Implement interfaces into the existing control system in cooperation with another student who does the implementation.

Bibliography/Sources:

- [1] Kanthabhabhajeya Sathyamyla, Berglund Joakim, Falkman Petter, Lennartson Bengt - Interface between SysML and Sequence Planner Language for Formal Verification – INCOSE 2013.
- [2] Augustsson Svante, Gustavsson Christiernin Linn, Bolmsjö Gunnar– Human and robot interaction based on safety zones in a shared work environment - 2014 ACM/IEEE international conference on Human-robot interaction.
- [3] Siemens Tecnomatix Process Simulate, User documentation, 2017

Diploma Thesis Supervisor: Ing. Pavel Burget, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, February 17, 2017

Abstract

Adaptability is a key factor for manufacturing companies for remaining competitive. Indeed, the fast changing of product demand but also the new trend of mass customization urge factories to improve their way to design production lines. Until now, production lines are installed in such a way that it has to be completely changed when a new product has to be produced.

In this thesis, models of both the factory line and the production plan are first developed based on a capability description. Mapping of production resources to production plan is then performed by a simple matching algorithm. Taking into account the material flow, production schedule is then automatically generated with a Depth-First Search algorithm with backtracking applied on the tree resulting of the mapping. This implementation is done using the Python language.

The schedule is then evaluated in Process Simulate software by modelling the industrial production line that will be later installed in CTU buildings.

Keywords: flexible production system, scheduling algorithm, virtual commissioning

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Pavel Burget, PhD. for offering me this very interesting topic for my thesis. Indeed, this topic is closely linked to my future work, so working on this thesis has been very instructive and has enabled me to gain essential skills. I also thank him for his continuous guidance during all the time of research and writing my thesis.

Beside my supervisor, I would also like to thank my teammates, Petr and Tomas, for their cooperation on the project. They have been very helpful, reactive and patient during our discussions.

Finally, I shall express my very profound gratitude to my parents and to my boyfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Table of contents

CHAPTER1 INTRODUCTION.....	1
1.1 PROBLEM DESCRIPTION	1
1.2 RELATED WORK.....	1
1.3 STRUCTURE OF THE THESIS	3
CHAPTER 2 PRODUCTION SYSTEMS MODEL	4
2.1 PRODUCTION PLAN MODEL	4
2.2 FACTORY SETUP MODEL.....	5
CHAPTER 3 PRODUCTION SCHEDULE ALGORITHM.....	8
3.1 MAPPING OF FACTORY RESOURCES TO PRODUCTION PLAN.....	8
3.2 CREATION OF THE SCHEDULING TREE	9
3.3 SCHEDULING ALGORITHM	11
3.3.1 <i>Depth-First Search algorithm with backtracking</i>	11
3.3.2 <i>Checking of material flow</i>	14
CHAPTER 4 VIRTUAL COMMISSIONING.....	15
4.1 HARDWARE ARCHITECTURE	15
4.2 PROCESS SIMULATE IMPLEMENTATION	18
4.2.1 <i>Standard mode: time-based simulation</i>	18
4.2.2 <i>Line simulation mode: event-based simulation</i>	20
4.2.2.1 <i>Event-based simulation</i>	20
4.2.2.2 <i>Off-line Programming (OLP)</i>	20
4.2.2.3 <i>Signals</i>	24
4.2.2.4 <i>Generation of appearances: material flow</i>	27
4.3 IMPLEMENTATION IN TIA PORTAL.....	29
CHAPTER 5 EXPERIMENTAL RESULTS	33
5.1 EVALUATION OF THE SCHEDULING ALGORITHM ON A SIMPLE EXAMPLE	33
5.2 EVALUATION OF THE SCHEDULING ALGORITHM ON THE TESTBED	36
5.2.1 <i>Presentation of the testbed</i>	36
5.2.2 <i>Results of the scheduling algorithm</i>	39
5.3 EVALUATION OF VIRTUAL COMMISSIONING.....	41

5.3.1 Evaluation of Process simulate implementation	42
5.3.1.1 CEE simulation	43
5.3.1.2 PLCSIM simulation	44
5.3.2 Evaluation of PLC program	45
CONCLUSION.....	48
REFERENCES	50
APPENDICES	51
APPENDIX A: FUNCTION BLOCK OF THE BIN PICKING OPERATION	51
APPENDIX B: CD CONTENT	53

Lists of figures

- 2.1 Metamodel of the production plan 5
- 2.2 Metamodel of the factory setup 6
- 3.1 Production schedule algorithm principle 8
- 3.2 Schema of a scheduling tree 10
- 4.1 Real hardware architecture and virtual commissioning architecture 15
- 4.2 Configuration of OPC UA. 16
- 4.3 Configuration of PLCSIM Advanced for emulating PLC program in TIA portal 17
- 4.4 Compound operation in Process Simulate. 19
- 4.5 Gantt chart of time-based simulation in standard mode 19
- 4.6 General basic organisation of a robotic program 21
- 4.7 Program Inventory 21
- 4.8 Paths in a robot program 22
- 4.9 Robot status signals 22
- 4.10 Basic relationship between robot (OLP) signals and PLC signals 23
- 4.11 Principle of robotic program in Process Simulate 24
- 4.12 Definition of signals for triggering compound operations 25
- 4.13 Declaration of transition condition of compound operations 25
- 4.14 Declaration of ending signal of compound operations 26
- 4.15 Gantt diagram of event-based simulation in line simulation mode 27
- 4.16 Definition of product instances in operation properties panel 28
- 4.17 Alternative material flow links in material flow viewer 29
- 4.18 Program blocks in the PLC program 30
- 4.19 Main program (OB1) 32

- 5.1 Example testbed 33
- 5.2 Result of the mapping for example testbed. 34
- 5.3 Scheduling tree for example testbed. 35
- 5.4 Resulting valid schedule for example testbed 35
- 5.1 Testbed implemented in Process Simulate 36
- 5.2 PERT diagram of the car process 38
- 5.7 Console view for the demonstration scenario 40
- 5.8 Result of the scheduling for the testbed scenario 41
- 5.9 Setting of CEE / PLCSIM simulation 42
- 5.10 Simulation panel for CEE 43
- 5.11 Addressing of signals in Signal Viewer 44
- 5.12 Monitoring signals from PLCSIM 45
- 5.13 Watch table in online mode 46
- 5.14 OB 1 in online mode 47

Chapter1 Introduction

1.1 Problem description

Flexible factory is nowadays critical for competitiveness of companies, as producing a high quality product for a small price does not guarantee success anymore. All the more as mass customisation is becoming a new reference for manufacturing industries and lifecycle of products are decreasing gradually. Therefore, production lines must be adaptable for keeping pace with this fast product turnover. As a result, lots of researches have been done in this field in the past years to meet the increasing demand of new adaptable production systems.

In order to determine the scope of the above, flexibility must be defined as it can have different meaning: it can be the capability to increase the range of available product, it can also refer to the ability to change from one product to the next one with very little effort and financial means for adapting the production line, or it can simply mean the capability to adapt volumes of production to customer demand. In the following, only the second definition of flexibility will be considered.

Today, the way how production is planned does not allow flexibility. Production line and production plans are often designed for producing only one product. Then, a change of production plan involves a lot of work for adapting or modifying the mechanical system and the information technology systems. It also requires a lot of time as usually the production has to stop for being able to reconfigure or replace some of the production line components. In order to address the quick changeovers goal, current approaches are focussed on adjustable equipment with mechatronic compatibility. Existing industrial solution rely as well on the integration of smart robots and on computer-integrated manufacturing which use computers for controlling the entire production line.

1.2 Related work

Capability-based approaches are the new trend for production planning and scheduling. Zah et al. propose in [1] to model the production plan and the different resources of the factory line from the perspective of capabilities. The schedule is optimized locally on a machine level and the production plan is stored using the Radio Frequency Identification technology which is

not suitable for real application in factory because of the slowness of the reading and writing process.

N. Keddis et al. propose in [2] a very similar approach but consider the material flow for generating the schedule automatically: operations of the production plan are assigned to the available machines and needed transportation operations are added. The optimization is rather done globally at the production process level. In [3], they focus on describing the workflow in an explicit and very accurate way by listing all the required data needed for a process. Therefore, the model of the workflow can be reused in different factories. The model is generated with the Eclipse Modelling Framework software and saved in XML-like file (.xmi) in order to use it directly in their scheduling application written in C++.

In a later work, Keddis and her team show how to transform the generated schedule obtained from the capability description in action sequences that are readable by the machines and can be automatically executed on them [4]. However, the work is limited to a simplified industrial setup and so need to be extended to more complex situations. In [5], they also propose a model-based plug and play approach relying on middleware communication for detecting new stations in the production line. Thus, the factory setup model is directly established during run time and really reflects the current setup. This allows to increase the adaptability of the Information Technology (IT) system about the factory setup.

The approach of the thesis is mainly based on the work of N. Keddis and her team. Both production plan and factory setup models are established off-line based on a capability description. Required operations are mapped as well to the available machine resulting in the creation of a tree diagram. Depth-First Search algorithm with backtracking if material flow is not possible between two consecutive operations is then run on this tree for generating a valid schedule for the given workflow and factory setup. The schedule is then tested by simulating the production line with Process Simulate; and the PLC program in TIA portal by emulating the PLC with PLCSIM Advanced.

The work of Keddis and her team is enhanced as the experimental setup is much more complex than the one used to evaluate their approach. Moreover, industrial components and industrial programming environment are used. Indeed, machines such as manufacturing robots which are able to do cooperative tasks are going to be used. Right now, the cooperation task only mean sharing their workspaces but the robots should eventually work on the same part. Production plans are also going to have many more production steps and the required operations are more elaborated.

1.3 Structure of the thesis

The thesis has the following organisation. Chapter 2 describes the two models used for representing a production system which are the production plan model and the factory setup model. Chapter 3 provides in a depth description of the algorithm that generates valid schedule for a given production plan with the current factory setup; it includes the creation of a searching tree and the use of a Depth-First Search algorithm with backtracking on it. Chapter 4 outlines the hardware architecture. In this chapter is also described the implementation of the production line simulation in Process Simulate and the PLC program in TIA portal. Finally, Chapter 5 presents the results of the scheduling algorithm and the results of the simulations performed on the virtual testbed and the PLC code.

Chapter 2 Production systems model

2.1 Production plan model

For increasing adaptability of manufacturing systems, the production plan must be described independently from specific technical information or the current factory setup and should model explicitly the workflow as proposed in [3].

Production plan have been modelled using object-oriented programming language Python. The metamodel in Figure 2.1 illustrates the different classes that have been implemented for modelling the production plan. The user specifies the demand of the customer, that is to say which product has to be produced and in which quantity. Each product is represented by a workflow which contains all the necessary production steps with their dependencies. Indeed, for executing one step, some other steps must be done before, thus each step has a list of the preceding steps. A production step is composed of several atomic operations that may require some material and particular tool.

Another information data can be required to describe each operation more accurately such as the geometry of the material or product, process relevant data as for example the exact position where the material must be glued, screwed, etc., error tolerance and quality requirements [3]. It is crucial to define these additional parameters because the same operation can apply to different type or size of material, so different tools may be needed (i.e. driving a screw of size 8mm or 10mm).

Each workflow has been designed using the Teamcenter Manufacturing software. It generates an Excel file with the production steps and their related operations as well as a Program Evaluation and Review Technique (PERT) diagram which describes dependencies of each step (c.f. Appendix 2: CD contents).

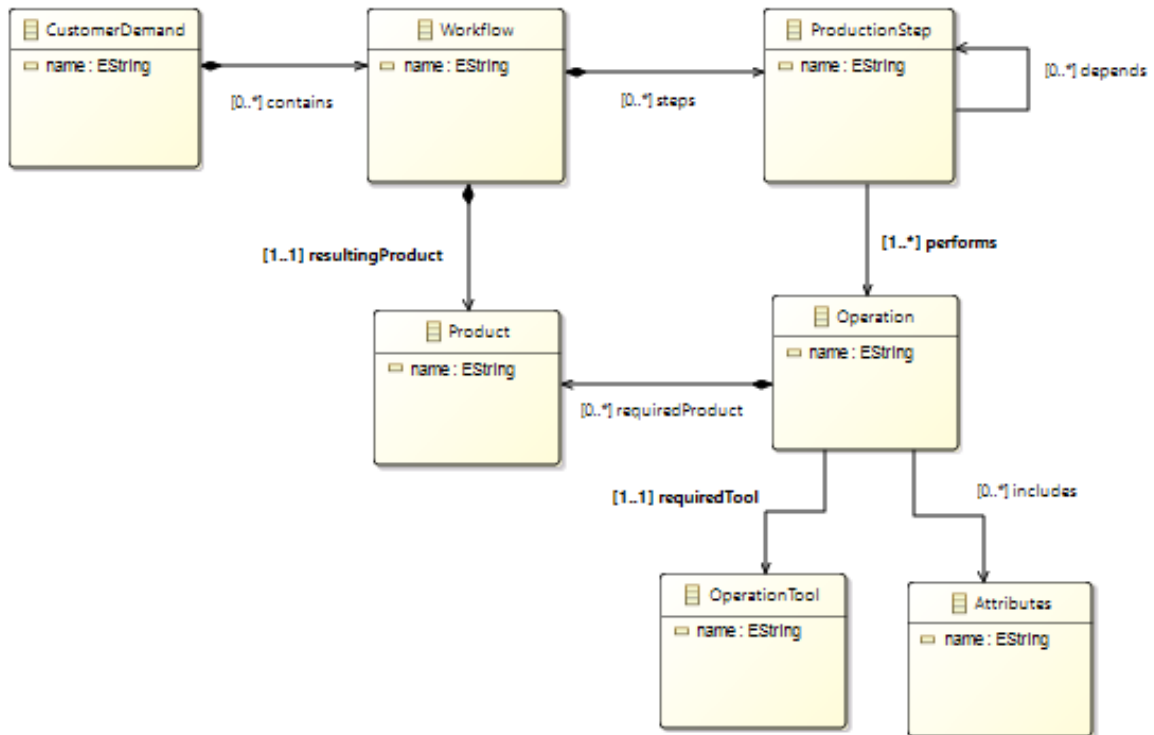


Figure 2.1: Metamodel of the production plan

2.2 Factory setup model

In [2], the factory setup is modelled with a capability-based approach: for each machine of the factory a list of all capabilities is established. In figure 2.2, which illustrates the implemented classes for the factory setup model, capabilities are labelled as process.

For describing more accurately the specificity of every type of machine such as robot, conveyor, or human, subclasses of the machine class are defined. Special parameters of each class are not yet defined, but it will be later refined using some mechanical properties as for instance the payload for a robot.

Each machine has a list of workspaces which it can work in. The workspaces for a machine in the subclass conveyor is obviously any workspace that is mechanically attached to it and that has a particular location. For a robot, it can be any position that can be reached by its end-effector. Finally, we define the workspace of the CollaborativeRobot class as the zone shared by both robots.

Each machine is able to perform several processes. Some examples of process for a robot might be screw, pick and place, bin picking, ... In order to more precisely describe every process

performed by the different machines, subclasses of the process class have to be created. Indeed, a transportation process only need few parameters such as the initial and final location, and the duration; whereas a manufacturing process (e.g. gluing, snapping, screwing...) requires much more information.

A manufacturing process has further parameters such as the location where to do the action, some attributes of the process (e.g. size), and the material it should handle. It is also needed to specify in which workspace the process can be executed. Indeed, the same process can be done in most of the workspaces reached by the machine. Each process is associated to one tool.

Regarding collaborative process, robots that are collaborating on the same process have to be defined with their specific role.

Data about the current factory setup are stored in an XML file visible on the attached CD (Appendix 2). In this file, described the resources of the production line (machine, workspace, tool) but also the capabilities of each machine are especially described.

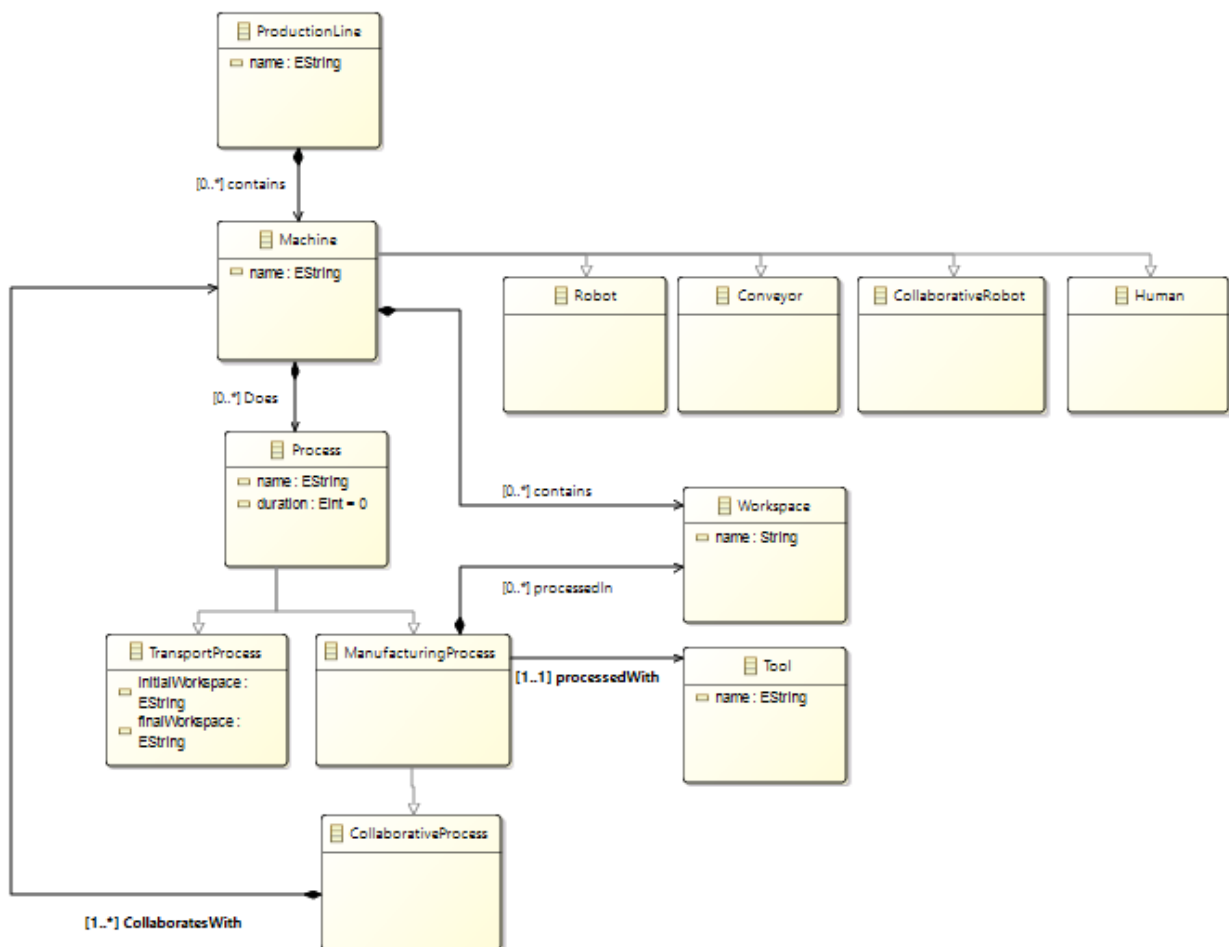


Figure 2.2: Metamodel of the factory setup

Both previous models are established off-line and are used during run time to generate the schedule of the desired workflow: defined classes and stored data, for example the XML factory setup file and Excel file generated by Teamcenter Manufacturing, allows to instantiate different objects such as machine, workspace and operation when running the production schedule algorithm.

Chapter 3 Production schedule algorithm

In this chapter, I present the algorithm that generates a valid production schedule for a given workflow with the available factory setup. The general principle of the algorithm is illustrated in figure 3.1. Each part will be fully explained in the following sections.

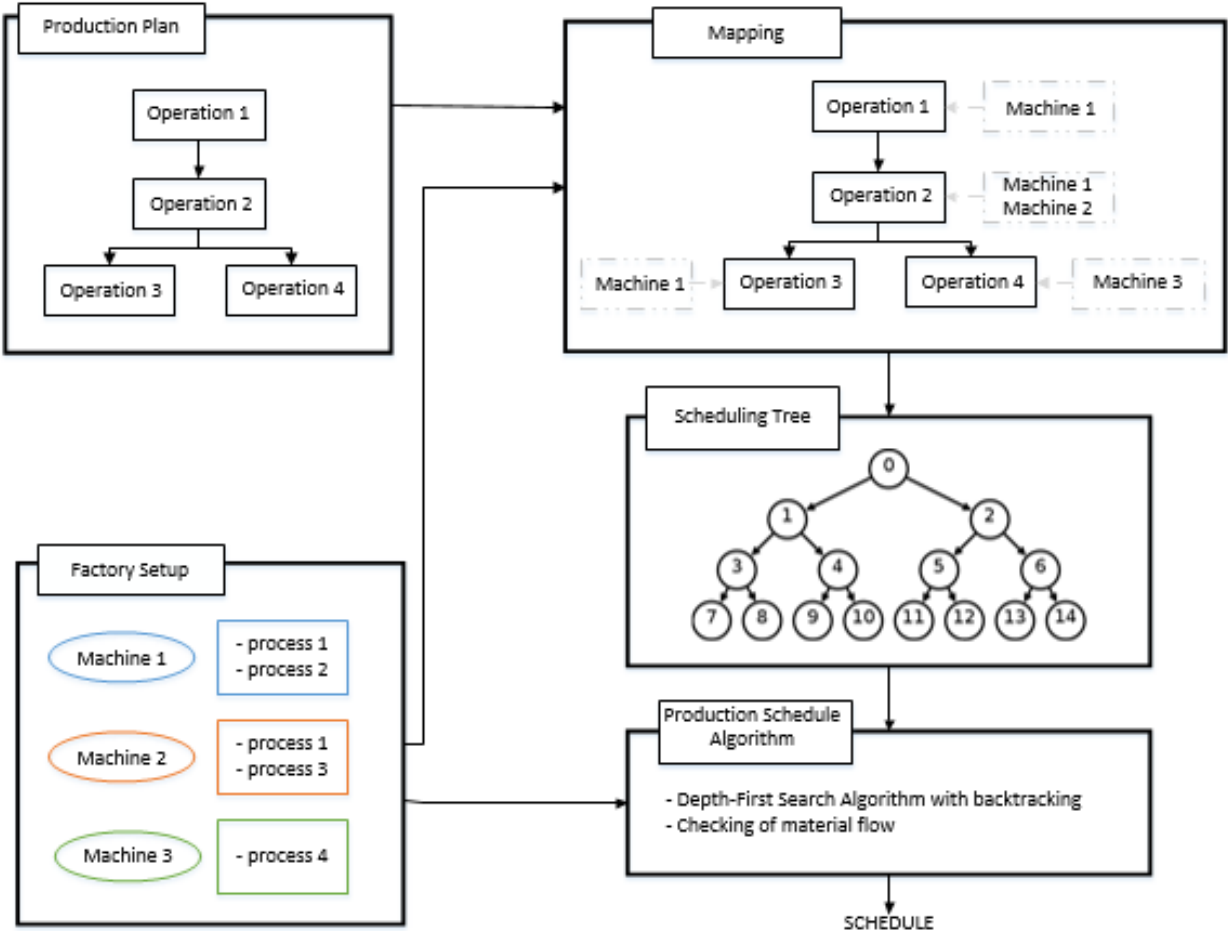


Figure 3.1: Production schedule algorithm principle

3.1 Mapping of factory resources to production plan

In the previous chapter, both the factory line and production plan have been modelled. These models are used now for mapping current factory resources to production plan: the purpose

of the mapping is to establish for each operation in the production plan a list of machines able to perform it, including the workspaces in which it can be performed. It is at this stage that importance of describing each model with the same vocabulary and in a generic way comes to light. Indeed, for each operation the mapping function iterates over all the processes of every machines; if the name of operation such as screw, snap, or transport is not identical in both descriptions, no matching is possible.

Moreover, it is necessary to check the material used for each operation. Actually, as no tool is defined right now, it has to be determined if a machine can do the same operation such as “Bin Pick the part” or “Assemble the part” for different materials. This is simply done by verifying that the required material of an operation of the production plan is the same as the material that can manipulate a machine in the factory setup. This checking will have to be redefined latter by checking the associated tool of an operation and its attributes.

During the mapping, if no matching is found for one operation, it means that the production plan cannot be scheduled on the currently available machines of the factory line. Either the production plan or the production line has to be modified, for example by integrating a new machine with this capability, to solve this issue.

3.2 Creation of the scheduling tree

The result of the mapping is then represented in an ordered directed tree. For this, the operations are first sorted according to the dependencies of their respective production steps. Besides taking into account the dependencies, checking of cooperative or concurrent tasks is done. Indeed, if two operations have been mapped on different machines and workspace they can be performed at the same time. Moreover, cooperative tasks must be scheduled at the same time; information about cooperation is contained in the Excel file generated from Teamcenter Manufacturing.

Ordered list with parallel tasks is not used at this step, because if a given node contains information of several operations, the checking of material flow between two nodes of the tree is not intuitive (see following section). The schedule will be refined at the very end, after the scheduling algorithm, in order to take into account parallel tasks.

The tree is created simply by iterating over all sequential operations. The result of the mapping is used: the number of machines able to perform the given operation and in which workspace determines the number of nodes to create. For example, if a given operation can be done on machine M1 in workspace W1 and on machine M2 in workspaces W2 and W3, then there will be 3 nodes in the scheduling tree for this operation. Each new created node is linked to every node created with the mapping of the preceding operation.

The main data stored in each node are:

- Name of the production plan operation
- Name of the machine
- In which workspace (or in which workspaces if the operation is a transport operation)
- Duration

A given node also contains the list of its children and a reference to its parent. The root node, that is to say node 0, is special as it does not contain any data. It is only defined because it is the node at which the scheduling algorithm on the tree begins.

Figure 3.2 shows a basic example that illustrates the creation of the scheduling tree. Two operations in the production plan are considered: the first operation has been mapped on two machines (node 1 and node 2). The second operation has been mapped on one machine (node 3 and node 4), as shown in figure 3.2(a) or on two machines (node 3 to node 6), as shown in figure 3.2(b).

Considering the fact that the tree is created in an exponential way and that the production plan can contain lots of operations, subtrees must be created. Each subtree represents the mapping of operations of several production steps of the whole production plan.

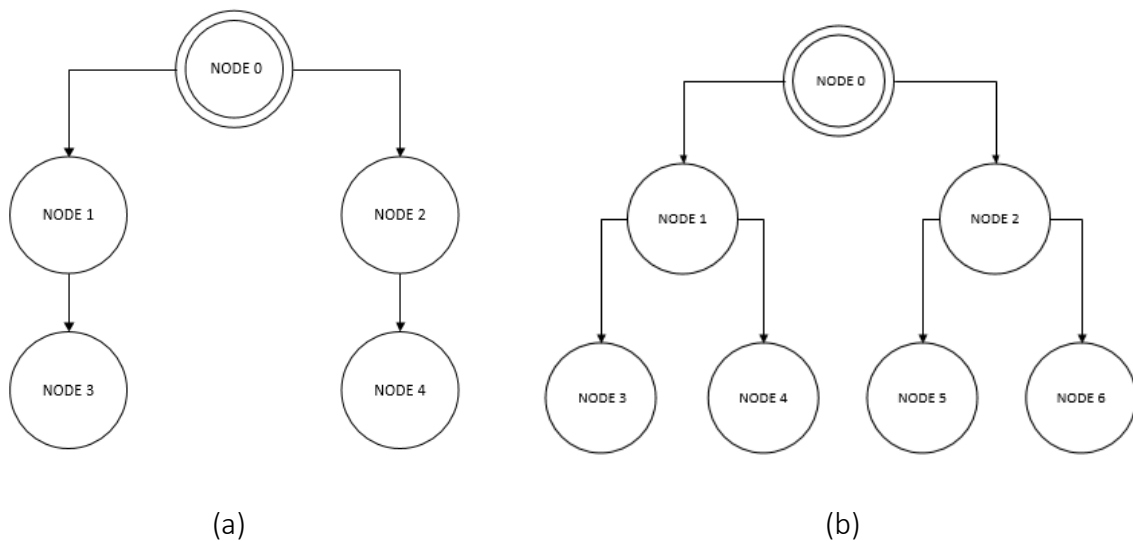


Figure 3.2: Schema of a scheduling tree. Root node is node 0

3.3 Scheduling algorithm

3.3.1 Depth-First Search algorithm with backtracking

Instead of using a Breadth-First Search algorithm (BFS) with backtracking as in [2], Depth-First search algorithm (DFS) with backtracking has been used for generating valid schedule. Indeed, as using BFS algorithm, using the DFS algorithm for searching the tree allows as well to quickly pruned the branch in which material flow is impossible, but above all to build the schedule in an effective way: each time the flow is possible for a given node, this node with all the information it contains such as the machine and the workspace in which the given operation is scheduled is added to the scheduling path. On the contrary, if backtracking is necessary due to the fact that no material flow is possible, then nodes are removed from the path according to how far the backtracking is necessary for finding a new branch that is still unvisited. For instance, if no flow is possible for the current node and its parent node has all of its children already visited, then the backtracking is needed two times which means that the current node and its parent node are removed from the path. Therefore, the main advantage of the DFS algorithm is that it allows to keep track of the path when going through the tree.

As the search is done on the whole tree that is to say every possible schedule has been considered, the obtained schedule is therefore the optimal one according to the desired criteria. Shortest schedule duration has been chosen, however other criteria such as makespan (completion time of the last task), energy consumption, or delivery time can be used.

The solution space that is explored are the subtrees created in the previous section. The DFS algorithm is iteratively executed on each subtree. Each obtained schedule for the given subtree is saved and later used as an input for the next subtree. At the end, a list of all possible schedules for the complete workflow is generated and the one with the shortest duration is chosen as the optimal schedule.

The algorithm 1 describes the implementation of the DFS algorithm which will be run for each subtree. It starts with the checking of whether the material required for the operation of the current node has been used before (line 2). Indeed, by looking at the whole path and more particularly at *productLocalisation* which stores the required materials of the current node and their location, it is possible to determine if a material has been already used and its current location. If the material has never been used before, then the checking of the flow is not required so the current node can be directly appended to the path (line 4). Otherwise, the flow of material must be checked (line 6) in order to determine if the material is directly at the current workspace (line 8), or if some intermediate transport steps are needed for bringing the material to the current desired workspace (line 10), or finally if the required material cannot be

transported from its position to the workspace of the current node. In the latter case backtracking is necessary (line 14).

The checking of flow will be further explained in the following section.

If the node has been added to the path, then the algorithm can continue recursively by selecting an unvisited child of the current node (line 19). If the current node is a leaf node (line 22), then the algorithm managed to find a valid schedule for the workflow with the current factory settings. This schedule is added to the subtree path list (line 24) and then we backtrack until some node has not been visited yet in order to find another valid schedule.

The resulting schedule with the shortest duration time is selected and saved in an XML file. This file will be sent to the Programmable Logic Controller (PLC); it will be further explained in the following chapters.

Algorithm 1: PseudoCode of the DFS algorithm with backtracking

Input: subtree, root node, path, production line

```
1 if current node is not root node then
2     v = checkProduct (current node, path)
3     if v == 1 then
4         path.append ( [currentNode, productLocalisation] )
5     else
6         flow, transportStep = checkFlow ( current node, production line, path)
7         if flow == 1 then
8             if transportStep == [] then
9                 path.append ( [currentNode, productLocalisation] )
10            else
11                path.append (transportStep)
12                path.append ( [currentNode, productLocalisation] )
13        else
14            backtrackingNeeded = 1
15 if backtrackingNeeded == 1 then
16     newNode = backtracking(current node, path)
17     DFS(subtree, newNode, path, production line)
18 else
19     if current node is not a leaf node then
20         foreach child of the current node not yet visited do
21             DFS(subtree, child, path, production line)
22     else
23         compute total time of the path
24         subtree.listPath.append ([path, total time] )
25         newNode = backtracking(current node, path)
26         DFS(subtree, newNode, path, production line)
```

3.3.2 Checking of material flow

In this section, I explain how the material flow checking is performed. Indeed, it is necessary to do it because a given operation can be scheduled on a different machine or in a different workspace than the ones of the previous operation, therefore the material must be transported between each workspace or machine.

The checking is done with the function *checkFlow* (line 6 in algorithm 1) in 3 consecutive steps. First, we verify that the localisation of the required material is in the workspace of the current node. If this is the case, the checking is finished otherwise we do the second step.

For each material which is not in the current workspace, we check if a direct flow is possible, in other words if a single machine can transport this material from its location to the workspace. This is done by verifying if one of the machines of the production line has a transport process with these workspaces as attributes. If more than one machine can do the transport, then the one with the shortest time is chosen. The intermediate transport step and the current node is added to the path in that order.

If the previous step is still unsuccessful we do the last step, which is the checking of indirect flow. Two lists are created: the first one contains the machines that have the initial workspace (where the required material is) in their workspace list; the second one contains the machines that have the current workspace in their workspace list. By iterating over the two lists, it is possible to find an intermediate workspace which belongs to both machines in the two lists. If there is such a workspace, we check that the transport process can be done between the initial and intermediate workspace but also between the intermediate workspace and final workspace.

If the flow is possible, the function is exited and all intermediate transport steps with the current node is added to the path. On the contrary, we run recursively for all workspaces of machines in the first list, the *checkIndirectFlow* algorithm until all machines have been visited.

As some operation involved machines such as robot, virtual workspaces must be defined in order to model the system in a global way. Indeed, it can be possible that after an operation, the material is not in a specific workspace but still in the gripper of some robot. Introducing virtual workspace is also crucial for checking the flow, as it is needed to know after each operation where are the different materials.

Chapter 4 Virtual commissioning

In this chapter, I present the future hardware architecture of the system and the work carried out regarding the virtual commissioning. Particularly, I will describe the implementation of the simulation in Process Simulate and the creation of the function blocks in TIA portal.

4.1 Hardware architecture

As the testbed is not installed yet in the university buildings, the hardware architecture will only be generally described in this section. I will lay the emphasis on the architecture used for the virtual commissioning. Figure 4.1 illustrates concepts that will be discussed here and in sections that follow.

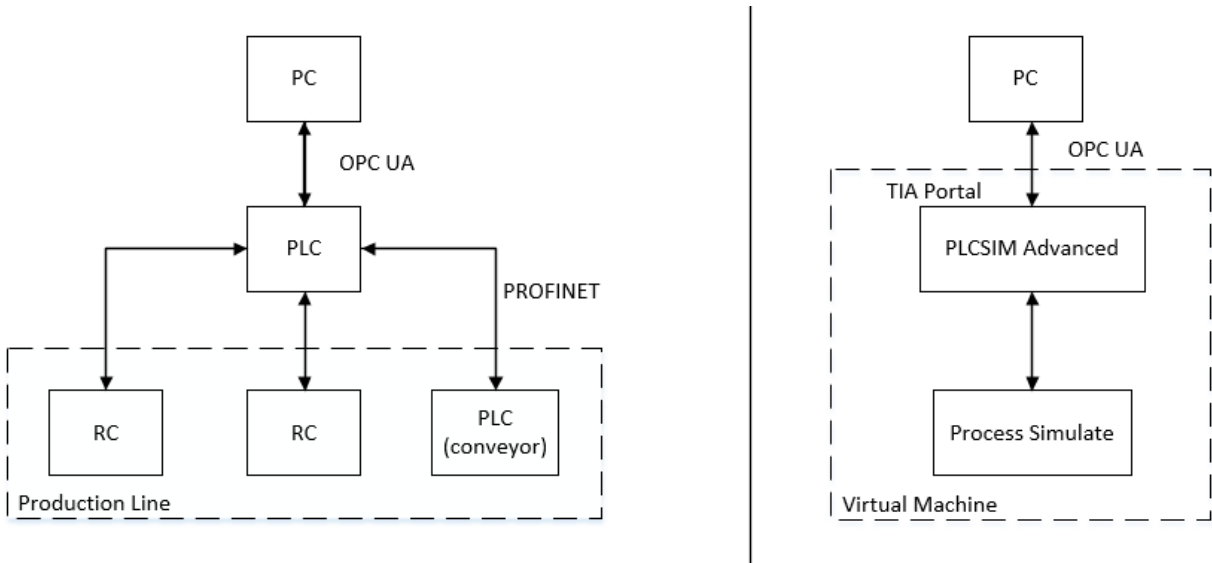


Figure 4.1: Real hardware architecture (left) and virtual commissioning architecture (right)

At this stage, the schedule is computed on the PC level by the Python program. In the previous chapter, I described the output file, containing the name of the machine and the program number for each step, that should be send to the PLC.

In order to do that and for abstracting from different platforms, Open Platform Communications Unified Architecture (OPC UA), which is a machine to machine communication protocol, is used for handling the data exchange between the PC and the PLC. This is an

interoperable, secure and reliable communication protocol, therefore it is considered as a standard in industrial automation [6]. As such, it makes the approach more general. Besides, using a protocol-independent interface is effective as only one interface needs to be installed for numerous applications.

As shown in figure 4.2, the PC will be used as a client for the application (sends production schedule) and the PLC will be used as a server (waits for the incoming data which is the schedule).

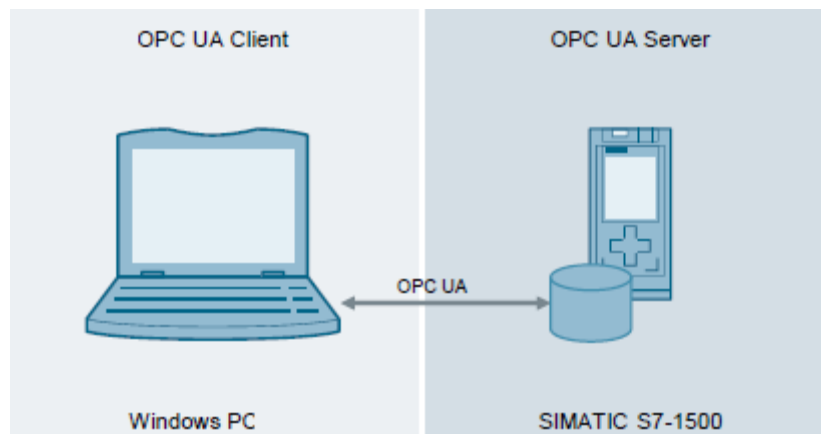


Figure 4.2: Configuration of OPC UA (source: [6])

The production line contains several robots and a conveyor which are respectively controlled by robot controller (RC) and another PLC. Exchange of inputs and outputs between the PLC and these different components of the production line will be done with PROFINET which is a standard for data communication in industrial systems. The PLC sends to corresponding machines the program number and the start signal for realising the operation (output signal) and gets back input signals about the status of the task that have been done by the different machines: done, error, etc.

Regarding the virtual commissioning architecture (cf. figure 4.1), a virtual PC is used for simulating the production line. It contains the following software:

- TIA portal in which the configuration of the PLC device (CPU 1516-3 PN/DP V2.0) is done and the program blocks are implemented.
- PLCSIM Advanced which is used for emulating the S7-1500 station. It allows comprehensive simulations without the need of physical connection to a real PLC. The configuration of this software is illustrated in figure 4.3 by creating a virtual PLC instance named "test" with some specific address.
- Process Simulate for simulating the complete behaviour of the production line. Resources of the production line (machines, tools, material) and operations (e.g. material flow and robotic operation) are modelled.

The PLC program (TIA portal) will be connected to Process Simulate via PLCSIM or OPC.

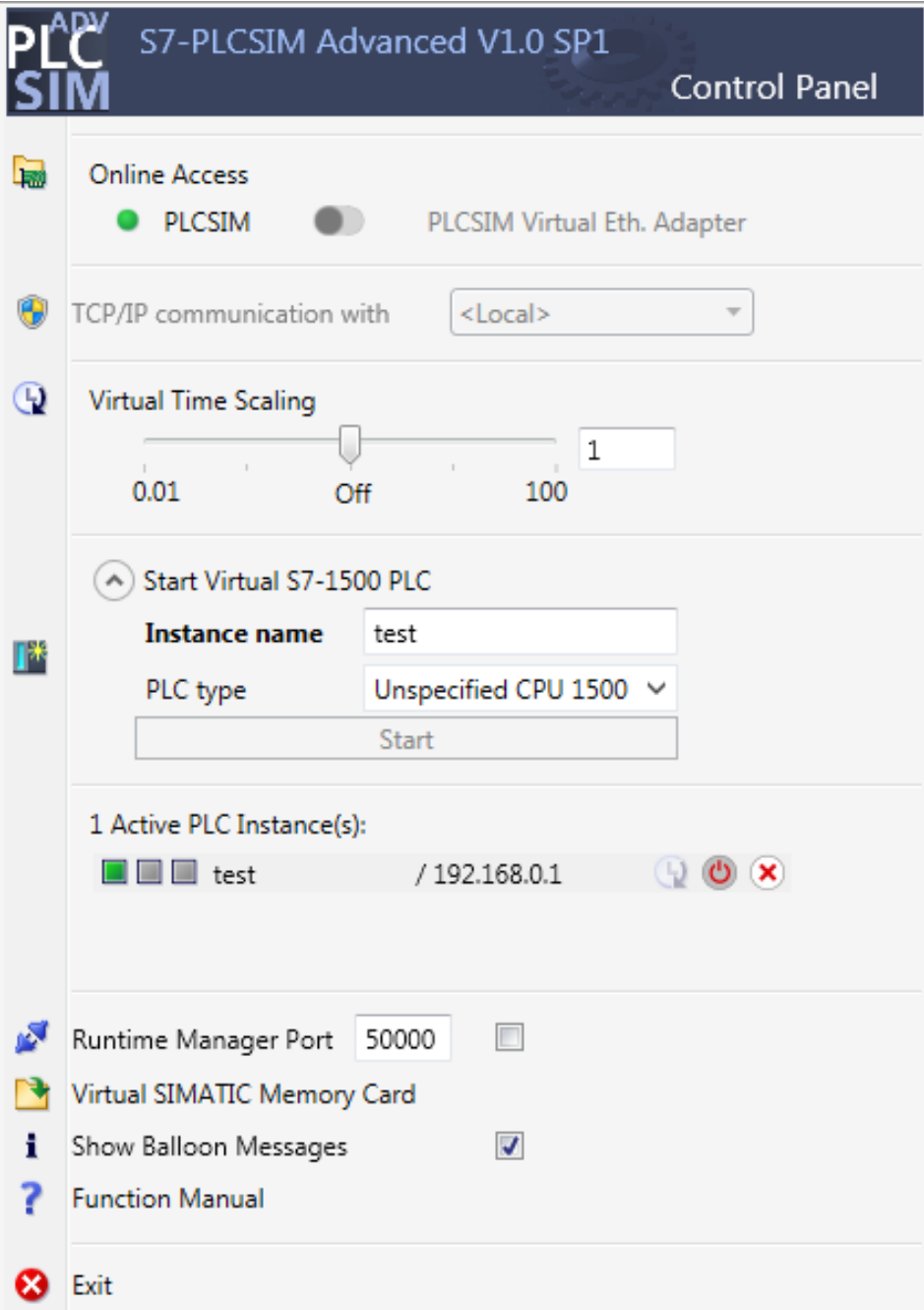


Figure 4.3: Configuration of PLCSIM Advanced for emulating PLC program in TIA portal

4.2 Process simulate implementation

4.2.1 Standard mode: time-based simulation

The time-based simulation is determined by the definition of the resources (machines, tools, ...), products and operations. The simulation is limited by the duration of operation and defines only one scenario since the logic is based on a Gantt chart diagram which is unique and describes a particular sequence of operation. As event-based simulation is quite difficult, time-based simulation is usually the first step to do for modelling the production line for checking its behaviour.

In time-based simulation, the execution of operation is determined by the sequence of operation: the evaluation of the transition criteria, which is the end criteria for the previous operation, is used for controlling the start of an operation.

From my colleague, I got the study with all the resources, products and operations already defined. Two types of operations were used:

- Object Flow operation for handling the behaviour of the conveyor. Indeed, this type of operations allows to move an object from one location to another. Thus, it is possible to model the transportation of the different parts involved in the workflow from one machine to another.
- General Robotic Operation for describing each path of robots. This operation is defined such as it contains all the points of the path that the robot should go.

As a given operation in the production plan can involve several operations defined in Process Simulate, it is needed to describe several operations as only one. As an example, the bin picking operation requires to transport a shuttle in the workspace where the robot put down the part on the conveyor. To do that, the compound type of operation is used. It is a node which contains operations, either object flow or robotic operation, or other compound operation. Figure 4.4 shows the compound operation “Bin Pick chassis” which involves the robotic operation bin picking executed by the liwa robot but also transportation of one shuttle (vozik_2_op_1) and opening and closing of the 2 clamps situated on the shuttle.

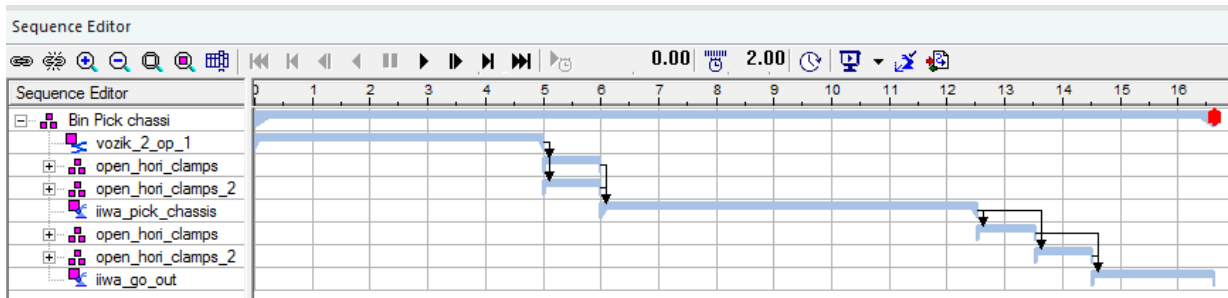


Figure 4.4: Compound operation in Process Simulate

The resulting Gantt diagram used for the time-based simulation is illustrated in figure 4.5. In this diagram, operations are linked such that it is the sequence of operations that determines the order of executed operation. For example, the compound operation “Bin Pick small battery” requires the operation “Assemble upper desk” and “Big battery assembly” to be finished for being able to start. The end of “Assemble upper desk” operation is needed because the bin picking and assemble upper desk operation use the same resources which is one of the shuttle of the conveyor. We can notice that some operations can be executed at the same time (e.g. “Small Battery assembly” and “Bin Pick ball holder”) as it does not involve the same machine and the same workspace.

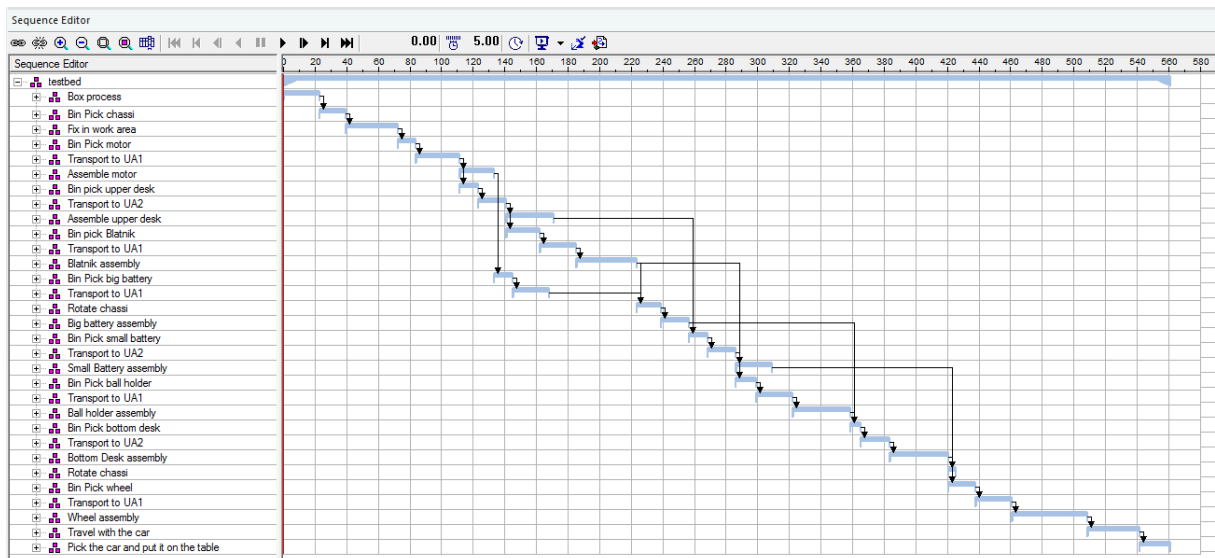


Figure 4.5: Gantt chart of time-based simulation in standard mode

The video of the time-based simulation can be seen in the CD attached to this thesis (see appendix B).

4.2.2 Line simulation mode: event-based simulation

4.2.2.1 Event-based simulation

The time-based simulation of the standard mode does not allow to simulate the production line properly with all the resources (robots, conveyor, control devices) in full synchronisation. As opposed to the time-based simulation which uses the predefined sequence of operations for simulating the line, the connections in Gantt chart do not determine anymore the executing orders of the operation in event-based simulation of the line simulation (LS) mode. This is actually the logic of the process and the events that occur which drives the simulation. Therefore, each simulation in event-based simulation are unique as it depends on events that can vary.

Switching from standard to LS mode demands some effort as it implies to use transition conditions and signals for handling the process sequence. It also implies creating robot programs instead of operation, and material flow for generating appearances. These 3 parts will be detailed in the rest of the section.

4.2.2.2 Off-line Programming (OLP)

Firstly, I will describe in this section how to create robot programs by using operations defined in the standard mode. As illustrated in figure 4.6, a robot has to execute two types of task which are organised in a robot program: motion task and logic instructions (non kinematics program modules). Robot program are listed in the program Inventory as shown in figure 4.7. In this panel, it is possible to edit a program as well as to create or delete one. Programs can also be downloaded to a shop-floor robot or uploaded from a shop-floor robot to Process Simulate. I only used the first three possibilities.

For using the robot program with its robot controller, programs have to be set as default (bold labelling). In this way, a given program can be executed directly by their path number during simulation.

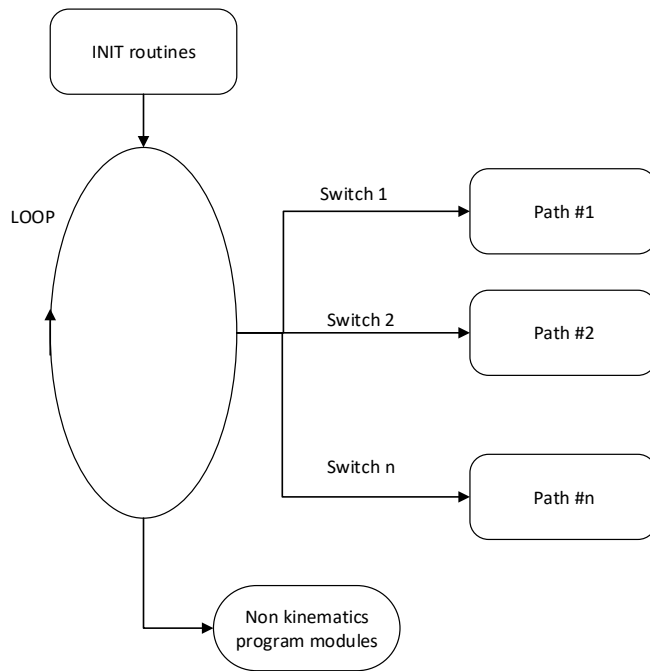


Figure 4.6.: General basic organisation of a robotic program

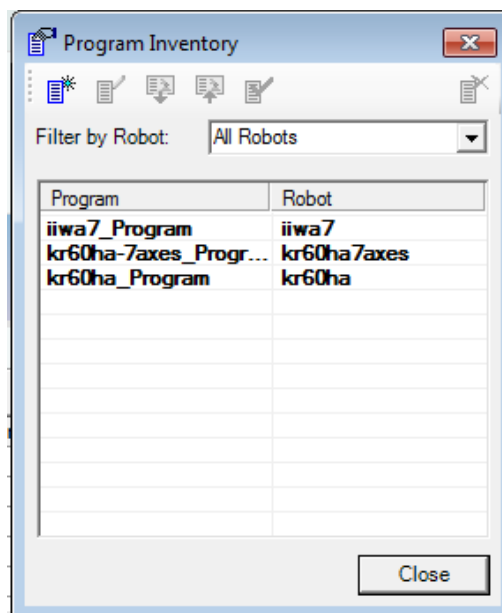


Figure 4.7: Program inventory

After creating an empty robot program, each path that the robot should execute is added to this program with a path number. This number is also called ProgramNumber in the status signal. As a robot can have several motion task, each path has a number in order to distinguish them. In order to execute a given motion task, the PLC should therefore send the corresponding ProgramNumber.

The figure 4.8 shows all the path saved in the program of the liwa robot which consist mainly in pick-and-place motion. A robot program has been created for each robot of the production line.

Paths & Locations	Path #	Attachment	X	Y
iiwa7_Program				
iiwa_pick_chassis	1			
iiwa_motor_pick	2			
iiwa_upper_desk_pick	3			
iiwa_fender_pick	4			
iiwa_big_battery_pick	5			
iiwa_small_battery_pick	6			
iiwa_ball_pick	7			
iiwa_lower_desk_pick	8			
iiwa_wheel_pick	9			
iiwa_pick_car	12			

Figure 4.8: Paths in a robot program

To assure that the path number correspond to the number send by the PLC, some mechanisms relying on signal exchange are used. Some of these status signals also prevent the robot from starting at the wrong time.

The figure 4.9 shows the default signal for a robot called here Kr60ha_7axes.

Signal Name	Memory	Type	PLC Connection	Resource
kr60ha7axes_programEnded	<input type="checkbox"/>	BOOL	<input checked="" type="checkbox"/>	kr60ha7axes
kr60ha7axes_mirrorProgramNumber	<input type="checkbox"/>	BYTE	<input checked="" type="checkbox"/>	kr60ha7axes
kr60ha7axes_errorProgramNumber	<input type="checkbox"/>	BOOL	<input checked="" type="checkbox"/>	kr60ha7axes
kr60ha7axes_robotReady	<input type="checkbox"/>	BOOL	<input checked="" type="checkbox"/>	kr60ha7axes
kr60ha7axes_at_HOME	<input type="checkbox"/>	BOOL	<input checked="" type="checkbox"/>	kr60ha7axes
kr60ha7axes_startProgram	<input type="checkbox"/>	BOOL	<input checked="" type="checkbox"/>	kr60ha7axes
kr60ha7axes_programNumber	<input type="checkbox"/>	BYTE	<input checked="" type="checkbox"/>	kr60ha7axes
kr60ha7axes_emergencyStop	<input type="checkbox"/>	BOOL	<input checked="" type="checkbox"/>	kr60ha7axes
kr60ha7axes_programPause	<input type="checkbox"/>	BOOL	<input checked="" type="checkbox"/>	kr60ha7axes

Figure 4.9: Robot status signals

Signals are controlling event-based simulations. Based on them, it is possible to trigger operations or events. Robot signals might be of the different following type:

- Default Input Signal; it is an input signal from the point of view of a Programmable Logic Controller (PLC). For instance, if the robot finished a path or if some errors occurred during this task, input signals indicating these type of events will be send to the PLC.
- Default Output Signal; it is an output signal from a viewpoint of a PLC. For example, if a task has to be executed by the robot during the simulation an output signal will be send by the PLC indicating the start of the operation.
- Memory Signal. It has been not used in the thesis.

The relationship between the robotic status signals and PLC signals are illustrated in figure 4.10. These status signals are continuously evaluated by the robot controller, as for example some input emergency stop signal or the input home position signal indicating that the robot is at its home position.

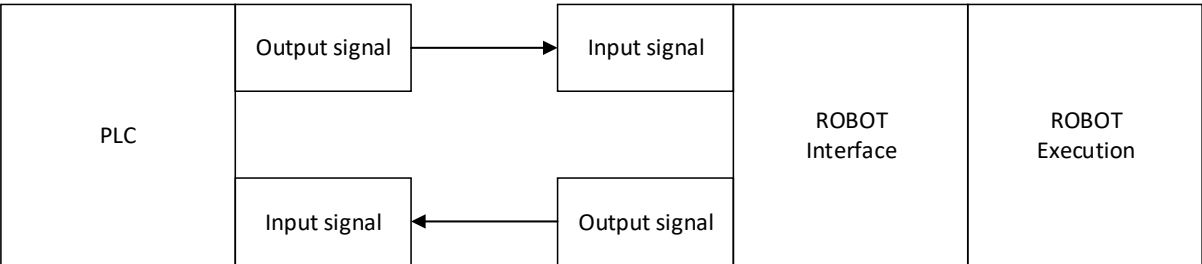


Figure 4.10: Basic relationship between robot (OLP) signals and PLC signals

These signals are illustrated in figure 4.11, which shows the principle of a robotic program. The signals are described from the PLC view. When the robot is mechanically and electrically ready, it sets the “robotReady” signal to TRUE. Then, the PLC sends a path number to the robot, this number is checked to assure that this number exists and is correct. This is done by mirroring the number received by the robot. If this number coincides with one in the robot program, the procedure can continue otherwise the signal “errorProgramNumber” is send to the PLC. Finally, the PLC sends the “startProgram” signal. On the rising edge of this signal, the robot starts its action. When the path is finished, the robot sets to TRUE the “programEnded” signal.

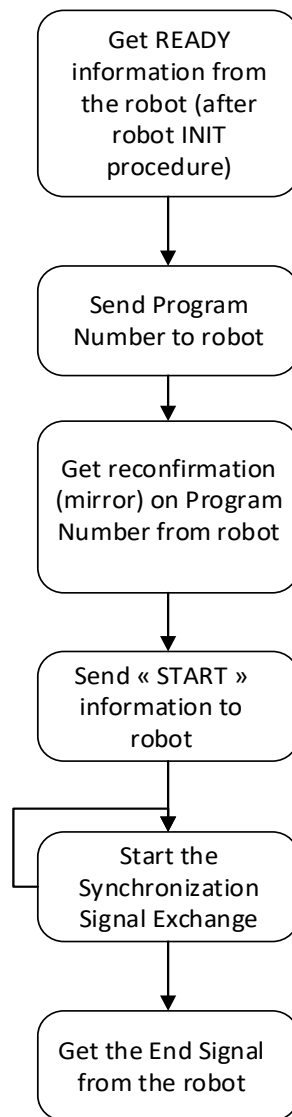
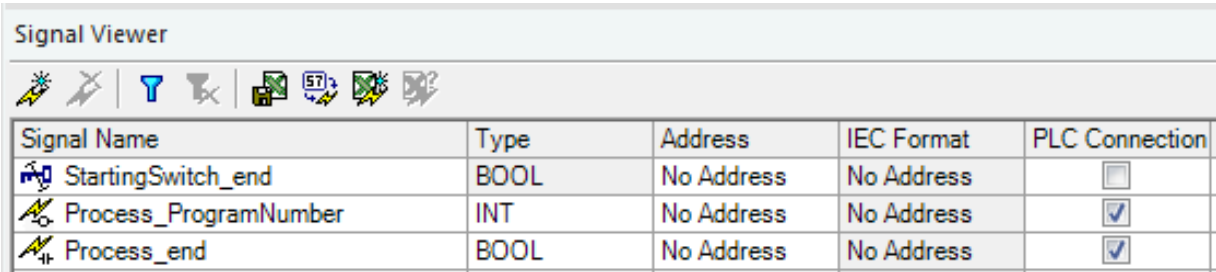


Figure 4.11: Principle of robotic program in Process Simulate (source [7])

4.2.2.3 Signal

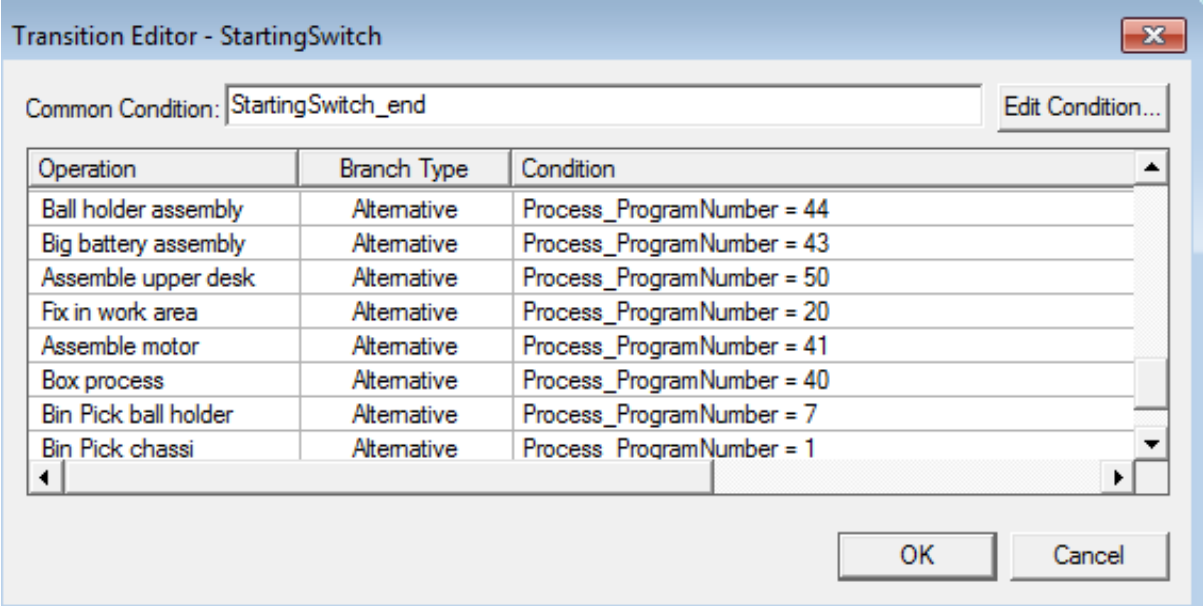
In event-based simulation, the operations are not starting anymore by using the end condition of the previous operation and links. Instead operation starts when their start-operation signal is triggered. These signals are independent of robotic operation which run using the robot programs and robot status signal. Therefore, it is required to create a start signal for every compound operation that we want to run and trigger it from the PLC. The principle of the signals is exactly the same as robotic signals: input signals are received by the PLC and output signals are send by the PLC.

2 new signals have been defined in the signal viewer (see figure 4.12). The first one is Process_ProgramNumber; this integer is an output signal from the viewpoint of the PLC and indicates which compound operation to execute. As illustrated in figure 4.13, every compound operation is triggered when this integer is equal to a particular value which is send by the PLC. The second signal is the Boolean Process_end which becomes TRUE when a compound operation ends. Its definition is illustrated in figure 4.14 for the box process. This is obviously an input signal as it is send to the PLC once the operation finishes.



Signal Name	Type	Address	IEC Format	PLC Connection
StartingSwitch_end	BOOL	No Address	No Address	<input type="checkbox"/>
Process_ProgramNumber	INT	No Address	No Address	<input checked="" type="checkbox"/>
Process_end	BOOL	No Address	No Address	<input checked="" type="checkbox"/>

Figure 4.12: Definition of signals for triggering compound operations



Common Condition: StartingSwitch_end Edit Condition...

Operation	Branch Type	Condition
Ball holder assembly	Alternative	Process_ProgramNumber = 44
Big battery assembly	Alternative	Process_ProgramNumber = 43
Assemble upper desk	Alternative	Process_ProgramNumber = 50
Fix in work area	Alternative	Process_ProgramNumber = 20
Assemble motor	Alternative	Process_ProgramNumber = 41
Box process	Alternative	Process_ProgramNumber = 40
Bin Pick ball holder	Alternative	Process_ProgramNumber = 7
Bin Pick chassi	Alternative	Process_ProgramNumber = 1

OK Cancel

Figure 4.13: Declaration of transition condition of compound operations

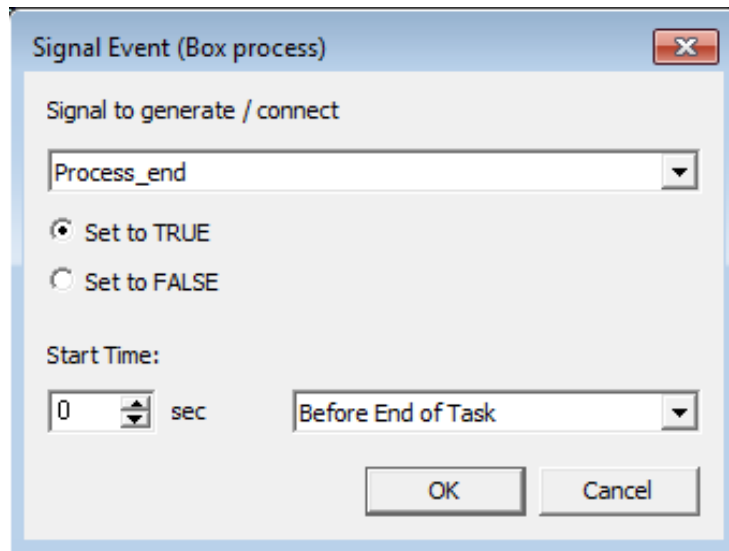


Figure 4.14: Declaration of ending signal of compound operations

A Non-Sim Operation called StartingSwitch is also defined. This is an empty operation that is added as the first operation under *testbed*. It is only used for logic purposes. Every compound operation is linked to this non-sim operation and it is set as common condition as precondition of transition for all operations as already shown in figure 4.13.

The finally obtained Gantt diagram for event-based simulation is illustrated in figure 4.15.

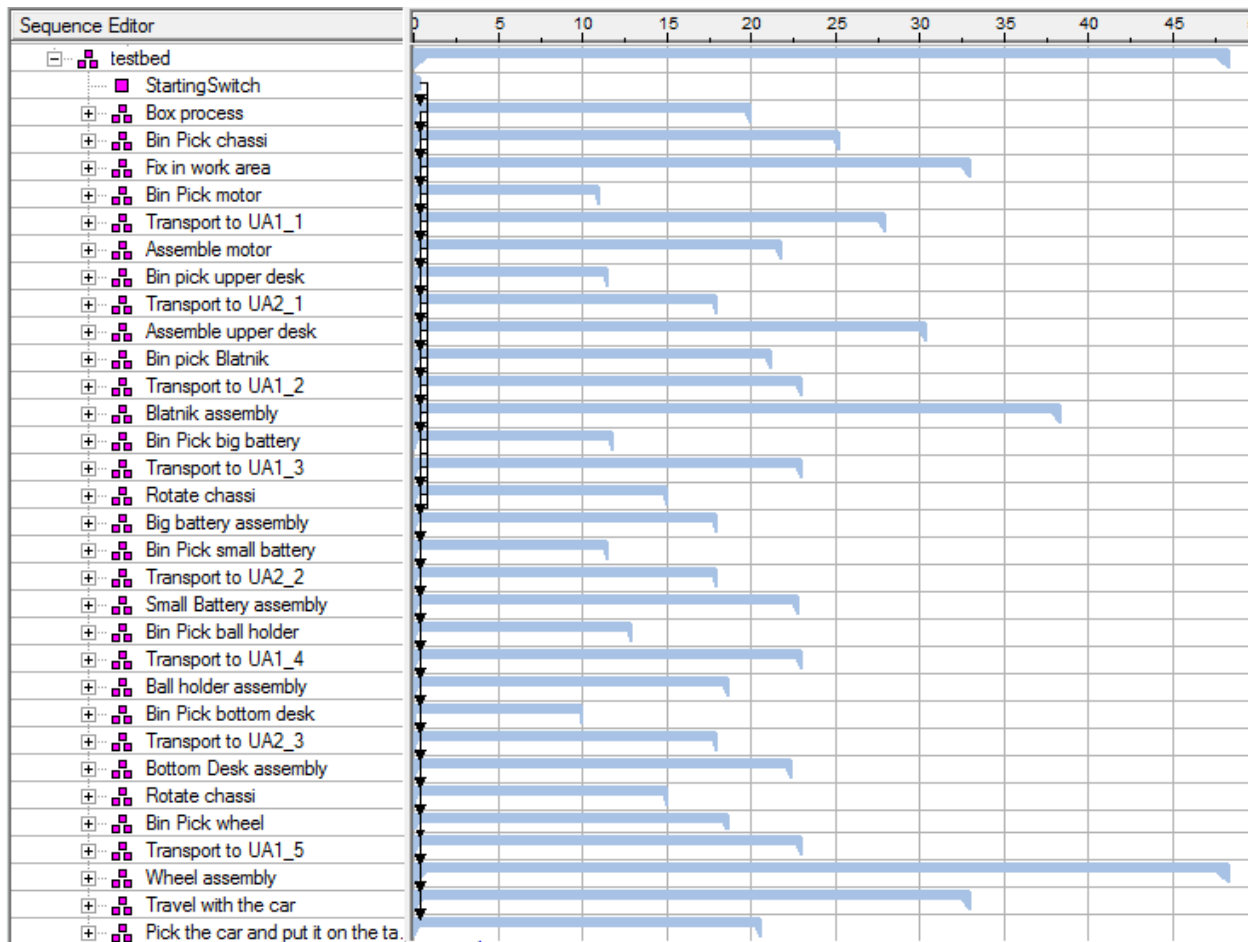


Figure 4.15: Gantt diagram of event-based simulation in line simulation mode

4.2.2.4 Generation of appearances: material flow

The Appearance option in Process Simulate is the only way in line simulation mode for visualising product data. Indeed, when a study is opened in line simulation mode for the first time, the products associated to the operations are not shown. Appearances allows as well to view a product at different locations at the same time, which is needed if different products have to be produced simultaneously on the same production line or if the simulation is repeated several times (several products are consecutively produced by the production lines).

When a simulation is running, part appearance is automatically generated when an operation uses a part. This part will remain “alive” until the part is no longer needed. When the simulation is reset, all appearances of the parts are completely removed.

For generating a part appearance automatically, the classic method is to define the product instances in the operation properties panel. Figure 4.16 illustrates the definition of the product

newChassis in the product instance of the robotic operation “Bin Picking chassis” executed by the liwa robot.

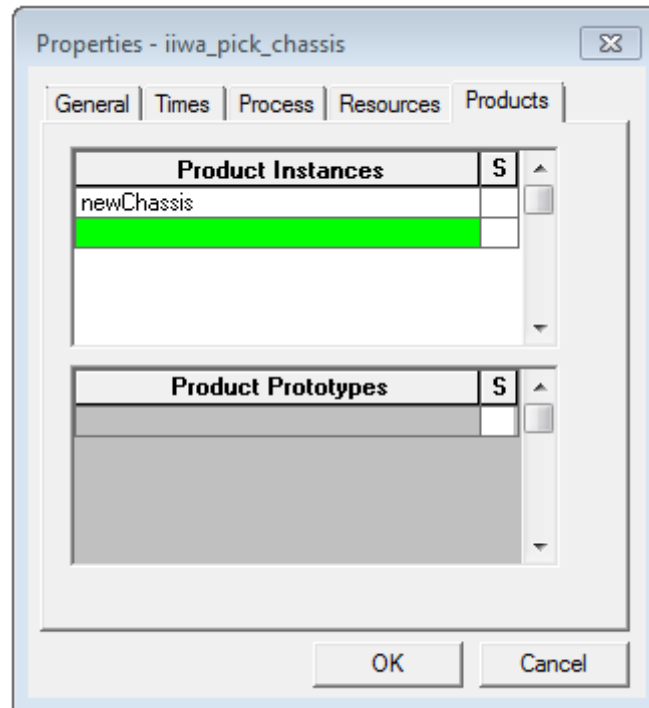


Figure 4.16: Definition of product instances in operation properties panel

Once the product instances of each operation has been defined, the material flow diagram is automatically generated thanks to the links in Gantt chart. Indeed, as explained earlier, the links in the Gantt chart does not determine anymore the order of executing of the operations. However, these links are used for controlling how an appearance is passed from one operation to another.

The whole material flow viewer can be seen in the CD attached to the thesis (see appendix B). As the material flow viewer can not contain compound operation, it contains all the operations and all material flow links. Dashed line between two operations in the material flow viewer represents an alternative material flow as illustrated in more detail in the figure 4.17. It allows parts to be passed in an exclusive way to different successors. Here parts that are situated in the box after the R1_pick_box process, can be passed to one of the bin picking operation and then to the transport process, called here *vozik*, according to which transition condition is currently true.

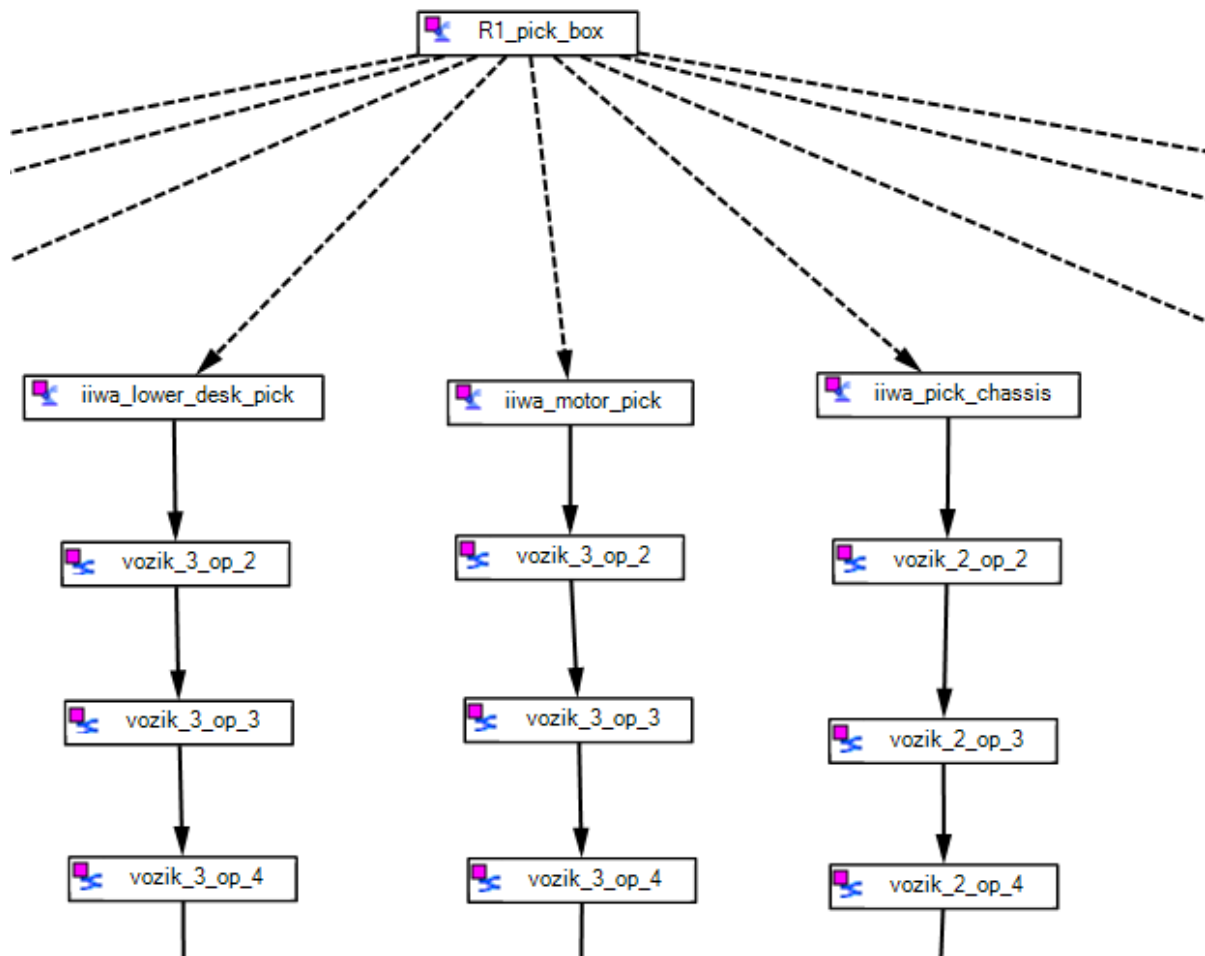


Figure 4.17: Alternative material flow links in material flow viewer

4.3 Implementation in TIA portal

Function blocks corresponding to each operation of the production plan are implemented in the PLC program blocks as shown in figure 4.18. Since some of these operations have the same logic, as for example “Bin Pick the part” in Chassis assembly and Motor assembly production step, I decided to use the same function block for controlling their execution. Therefore, the PLC controls and monitors the “Bin Pick the part” operation with the function block “Operation BP liwa” (FB1).

All the following blocks have been implemented using the Ladder logic, which is a programming language for PLCs.

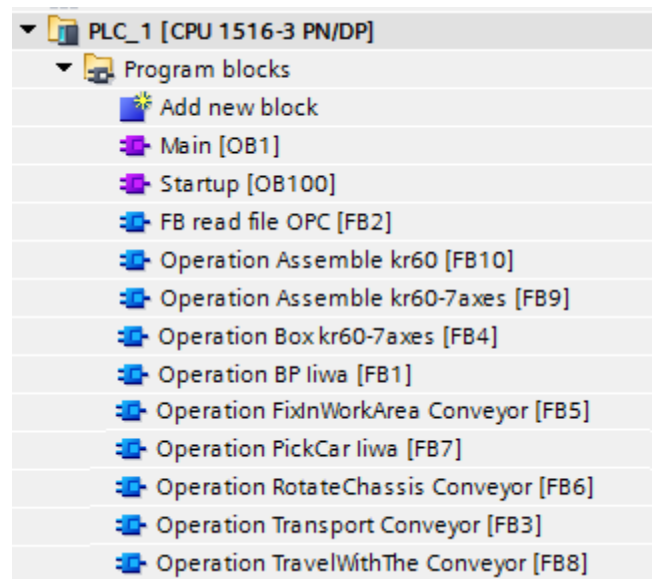


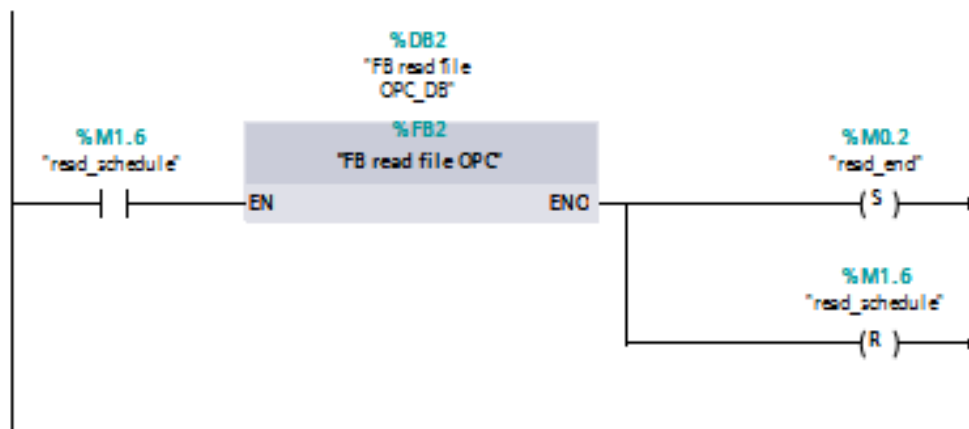
Figure 4.18: Program blocks in the PLC program

The main program is implemented in the organisation block OB 1 as illustrated in figure 4.19. In STEP 7, OB1 is processed cyclically by the CPU. The CPU reads line by line and executes the program commands. The general principle of the implementation is as follow:

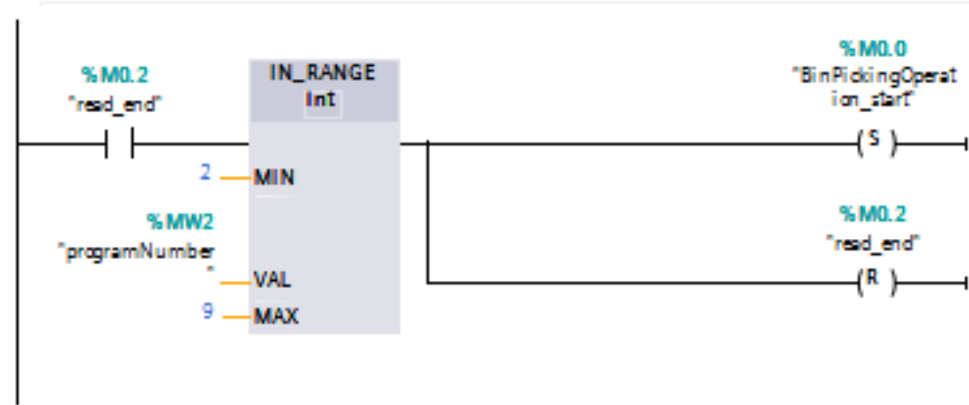
- Network 1: When the *read_schedule* Boolean is TRUE; the CPU reads the value of the current *programNumber* which is in the schedule file sent via OPC to the PLC (see section 5.2.2). This number indicates to the PLC which function block corresponding to operations in the production plan to run. The Integer *programNumber* is passed on as block parameters from the organization block OB 1 to the function block. As the OPC connection between the PC and PLC is not established yet, the function block FB 2 executing this task is actually empty. The *read_schedule* Boolean is firstly initialised to TRUE in the Startup organisation block (OB100).
- Network 2 and 3: Depending on the integer *programNumber*, the function block is executed. The checking of this integer is done with a IN_RANGE block which sets to true the Boolean *operation_start* if *programNumber* has the specified value. When the previous Boolean is TRUE, the function block can be run. Network 2 and 3 illustrate the principle of the program for the bin picking operation.
- network 24 and 25: When the function block execution is finished, the verification of the status of the execution is done. If the function block has the output *done*, then the Boolean *read_schedule* is set to TRUE and a new cycle can be run with a new *programNumber*. Otherwise, if the output is *error*, then a human operator has to be called to check where the error comes from.

► **Block title:** "Main Program Sweep (Cycle)"

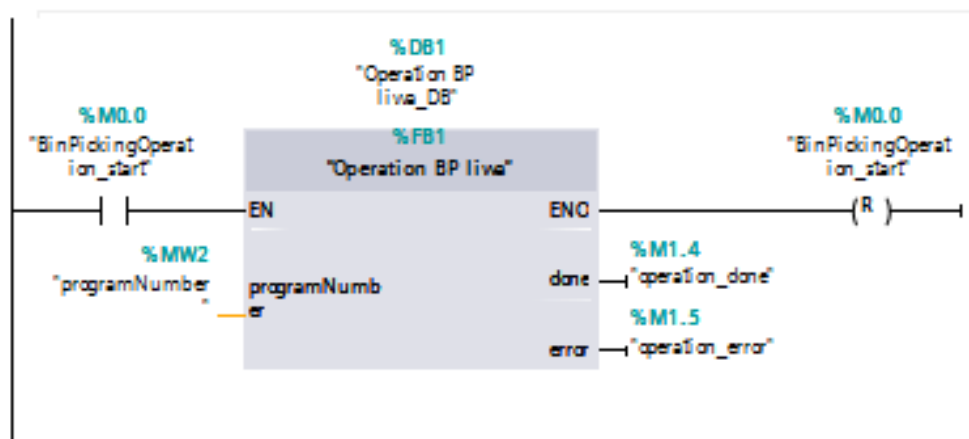
▼ **Network 1:** Read programNumber in schedule file OPC



▼ **Network 2:** check bin picking operation program number



▼ **Network 3:** Bin Picking Operation liwa



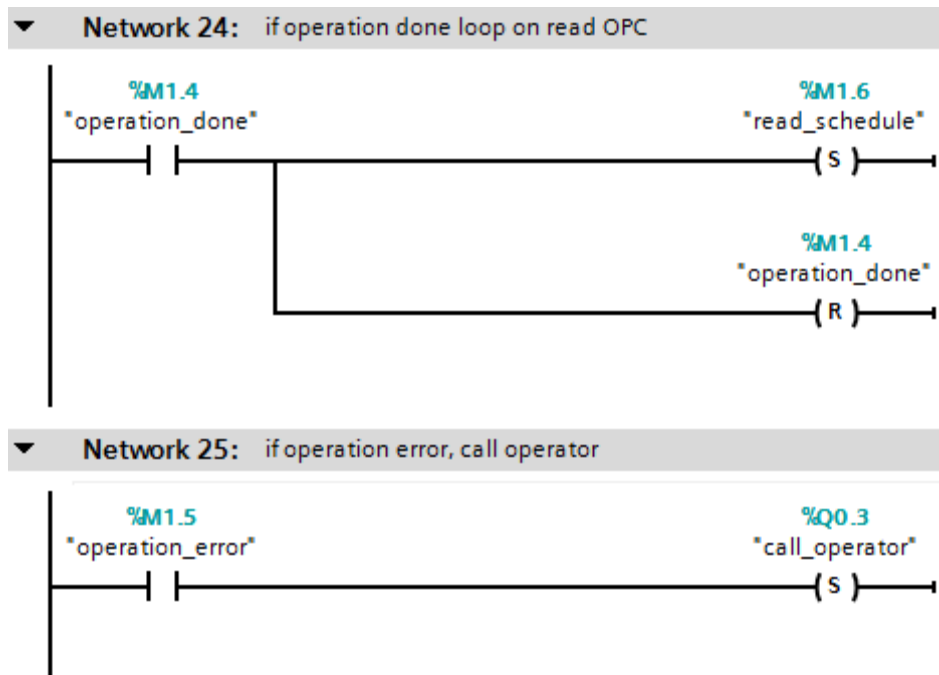


Figure 4.19: Main program (OB1)

All function blocks have almost the same implementation. The integer *programNumber* is defined in the variable declaration table as input and the Booleans *error* and *done* as output (declaration “in” and “out”).

The general implementation of a function block controlling a robotic program is illustrated in appendix A with the bin picking operation executed by the liwa robot. The logic is exactly the same as the logic used in Process Simulate (figure 4.11). *programNumber* is firstly sent to the robot when it is mechanically ready and in its home position. Then, the number mirrored by the robot is checked by the PLC in order to ensure that the path can be executed safely. If this is correct, the robot starts the program. At the end of the execution, the status of the operation is determined (either *done* or *error*).

Function blocks controlling an operation defined as compound operation of Object Flow in Process Simulate are slightly different. This mainly concerns the conveyor operations. The function block only sends the *Process_ProgramNumber* which triggers the first operation of the compound operation. It also checks the end of the operation.

Each tag (Integer, Input, Output, Memory...) has some address. For example, an output of data type Boolean will have the address Q0.0 and an input of data type Boolean will have I0.0. The PLC tags can be seen in the Excel file *PLCTags* in the attached CD. This Excel file has been generated directly by TIA portal.

Chapter 5 Experimental results

In this chapter, I present and analyse the results of the experiments that were carried out for testing the scheduling algorithm. First, the scheduling algorithm has been tested on a simple example for evaluating the Depth-First Search algorithm, then the experimental setup has been used for validating the flexibility of the approach. Finally, results of the simulation performed on the production line model in Process Simulate and the PLC program in TIA portal are presented.

5.1 Evaluation of the scheduling algorithm on a simple example

I first evaluated the scheduling algorithm on a very simple example for testing if the flow checking of material was correct. Therefore, I considered the following scenario which involves the addition of transportation steps between two consecutive steps.

The example testbed is illustrated in figure 5.3 and is composed of two robots R1 and R2 and a conveyor. There are 5 workspaces:

- Robot R1 can work in workspaces 1, 2, 3, and 4
- Robot R2 can work in workspaces 4 and 5
- The conveyor can transport material between the workspaces 2, 3, and 4

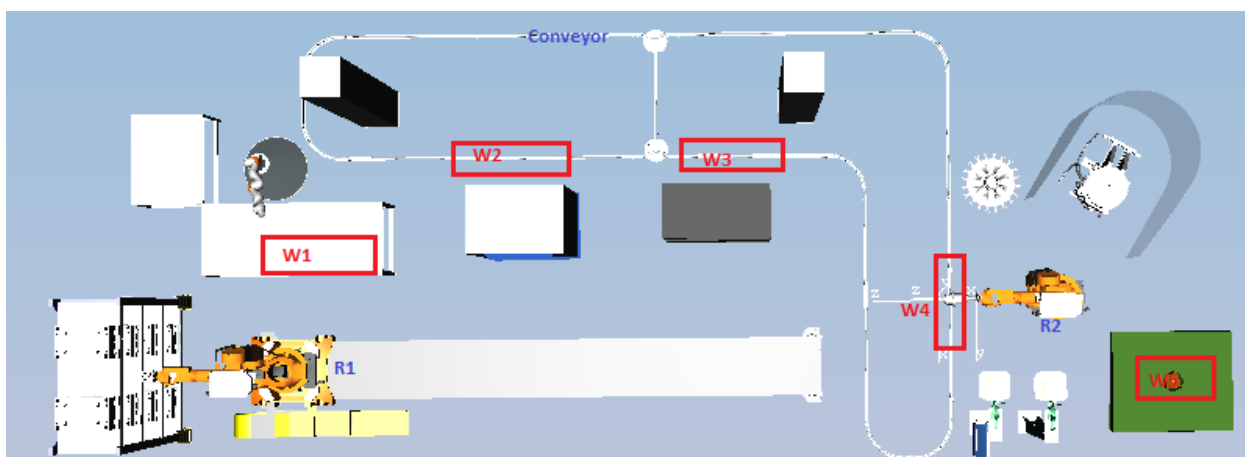


Figure 5.1: Example testbed

The machines have the following capabilities: robot R1 can screw in workspace 1 and get product from store in workspaces 2 and 3. Robot R2 can also screw in workspace 4 and executes mounting process in workspace 5. Finally, robot R2 can also transport an object from workspace 4 to workspace 5.

The considered production plan is really simple:

- 1- Get product 1
- 2- Screw product 2 in product 1
- 3- Get product 3
- 4- Mount the product 3 on the product 1-2

Results for the example testbed are illustrated in figures 5.4, 5.5, and 5.6 which represent respectively the result of the mapping between production plan operation to machines, the created scheduling tree and finally the obtained valid schedule.

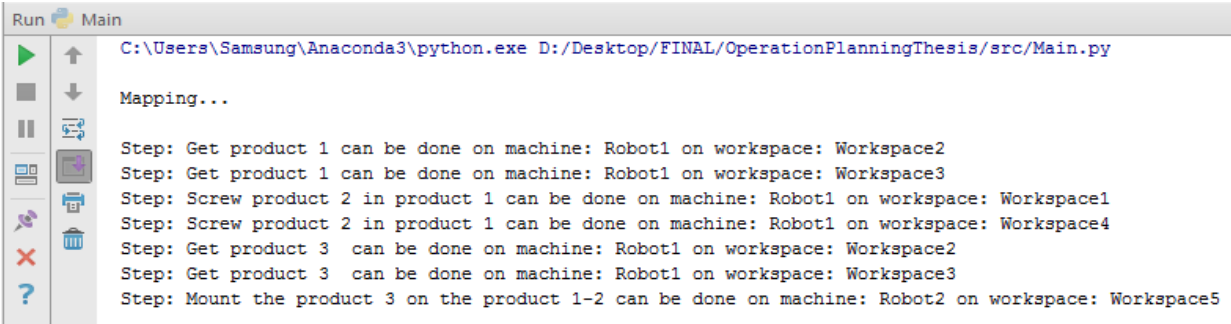


Figure 5.2: Result of the mapping for example testbed (screenshot)

```

Run Main
Creation of the scheduling tree...

node0 : <Tree.Node object at 0x000001D43D4A3668> has children [<Tree.Node object at 0x000001D43D4A3630>, <Tree.Node object at 0x000001D43D4A35F8>]
node1 : <Tree.Node object at 0x000001D43D4A3630> has children [<Tree.Node object at 0x000001D43D4A36A0>, <Tree.Node object at 0x000001D43D4A36D8>]
node2 : <Tree.Node object at 0x000001D43D4A35F8> has children [<Tree.Node object at 0x000001D43D4A3710>, <Tree.Node object at 0x000001D43D4A3748>]
node3 : <Tree.Node object at 0x000001D43D4A36A0> has children [<Tree.Node object at 0x000001D43D4A3780>, <Tree.Node object at 0x000001D43D4A37B8>]
node4 : <Tree.Node object at 0x000001D43D4A36D8> has children [<Tree.Node object at 0x000001D43D4A37F0>, <Tree.Node object at 0x000001D43D4A3828>]
node5 : <Tree.Node object at 0x000001D43D4A3710> has children [<Tree.Node object at 0x000001D43D4A3860>, <Tree.Node object at 0x000001D43D4A3898>]
node6 : <Tree.Node object at 0x000001D43D4A3748> has children [<Tree.Node object at 0x000001D43D4A38D0>, <Tree.Node object at 0x000001D43D4A3908>]
node7 : <Tree.Node object at 0x000001D43D4A3780> has children [<Tree.Node object at 0x000001D43D4A3940>]
node8 : <Tree.Node object at 0x000001D43D4A37B8> has children [<Tree.Node object at 0x000001D43D4A3978>]
node9 : <Tree.Node object at 0x000001D43D4A37F0> has children [<Tree.Node object at 0x000001D43D4A39B0>]
node10 : <Tree.Node object at 0x000001D43D4A3828> has children [<Tree.Node object at 0x000001D43D4A39E8>]
node11 : <Tree.Node object at 0x000001D43D4A3860> has children [<Tree.Node object at 0x000001D43D4A3A20>]
node12 : <Tree.Node object at 0x000001D43D4A3898> has children [<Tree.Node object at 0x000001D43D4A3A58>]
node13 : <Tree.Node object at 0x000001D43D4A38D0> has children [<Tree.Node object at 0x000001D43D4A3A90>]
node14 : <Tree.Node object at 0x000001D43D4A3908> has children [<Tree.Node object at 0x000001D43D4A3AC8>]
node15 : <Tree.Node object at 0x000001D43D4A3940> has children []
node16 : <Tree.Node object at 0x000001D43D4A3978> has children []
node17 : <Tree.Node object at 0x000001D43D4A39B0> has children []
node18 : <Tree.Node object at 0x000001D43D4A39E8> has children []
node19 : <Tree.Node object at 0x000001D43D4A3A20> has children []
node20 : <Tree.Node object at 0x000001D43D4A3A58> has children []
node21 : <Tree.Node object at 0x000001D43D4A3A90> has children []
node22 : <Tree.Node object at 0x000001D43D4A3AC8> has children []

```

Figure 5.3: Scheduling tree for example testbed (screenshot)

```

Run Main
Depth First Search algorithm running...

Step: 1 is done on machine: Robot1 in Workspace3
duration of this step: 100

Transport Step from Workspace3 to workspace Workspace4 is done by machine Conveyor
duration of the transport: 10

Step: 2 is done on machine: Robot1 in Workspace4
duration of this step: 5

Transport Step from Workspace4 to workspace Workspace3 is done by machine Conveyor
duration of the transport: 10

Step: 3 is done on machine: Robot1 in Workspace3
duration of this step: 100

Transport Step from Workspace3 to workspace Workspace4 is done by machine Conveyor
duration of the transport: 10

Transport Step from Workspace4 to workspace Workspace5 is done by machine Robot2
duration of the transport: 200

Step: 4 is done on machine: Robot2 in Workspace5
duration of this step: 50

Process finished with exit code 0

```

Figure 5.4: Resulting valid schedule for example testbed (screenshot)

This scenario validates the part of the algorithm related to the checking of flow: if no possible direct flow between two workspaces is possible, transportation steps are scheduled on the intermediate machines. These transportation steps in the case of indirect material flow checking are not always optimal considering the process duration. Indeed, some machines can have been chosen for providing the transportation capability whereas others also have this capability but with a smaller process duration.

5.2 Evaluation of the scheduling algorithm on the testbed

5.2.1 Presentation of the testbed

In order to evaluate the approach, I performed simulations on the testbed that will be installed in CTU buildings soon. The testbed is shown in Figure 5.1. Up to now, this setup is used for the production of a small car, but eventually it should produce a wide range of different products.



Figure 5.1: Testbed implemented in Process Simulate

The factory setup is composed of 3 robots. There are two KUKA KR60 robots, of which one is mounted on a 7th axis for allowing an additional translation movement. These two robots can cooperate together in the assembly area. Furthermore, there is a KUKA IIWA robot, which is a lightweight robot. It can cooperate with a human at the table. The production line has also a conveyor of the brand MONTRAC used for transportation of the different materials from one workspace to another. Transportation is done with self-propelled shuttles moving on monorails. The main capabilities of each machine are summarized in Table 1.

Machine	Capability 1	Capability 2
KUKA KR60 + 7 th axis	Pick box in the store	Assemble parts
KUKA KR60	Screw	Assemble parts
KUKA IIWA	Bin Picking	Pick car from MONTRAC
MONTRAC	Parts delivery	

Table 1: Capabilities of each machine used for the demonstration scenario

The workflow of the car process is really straightforward: the KUKA KR60 on the 7th axis picks a box in the store and puts it on the table. Then, the IIWA robot will successively bin pick the material required for the production step and put it on a shuttle. There are 2 shuttles: one of them contains a couple of clamps that maintains the chassis during the assembly operation of other parts and that will be fixed in the assembly area. The material will be transported by the other shuttle to the unload area where one of the two KUKA robots will successively proceed to take the part and proceed to the assembly operation (e.g. screw, snap ...). When all parts have been assembled, the car is transported back to the table.

The PERT diagram generated by Teamcenter showing the production steps of the car process and their dependencies for the car process is illustrated in Figure 5.2.

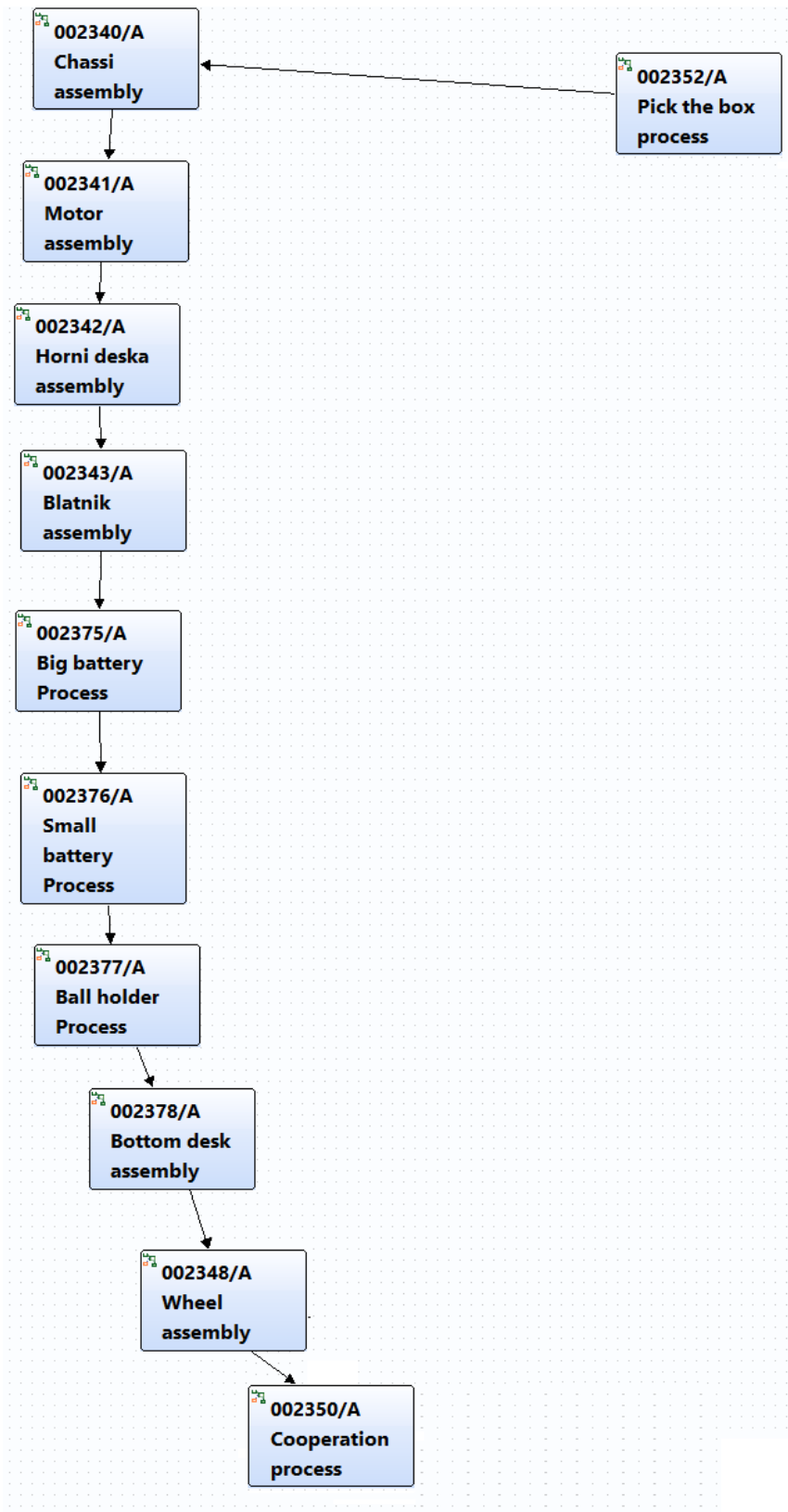


Figure 5.2: PERT diagram of the car process

5.2.2 Results of the scheduling algorithm

The capabilities of the machines saved in the factory setup XML file are exactly the ones implemented in Process Simulate.

As there are many more operations and steps in this scenario, instead of printing the result in the console of PyCharm; results of the different steps of the scheduling algorithm have been printed in external text files for a better reading as shown in figure 5.7. These files can be seen in the CD attached to this thesis (c.f. Appendix 2). Besides the mapping file, the tree file and the resulting schedule text file, one more output file is generated: OPCschedule. This file contains for each step of the production schedule the name of the machine on which the operation is scheduled and the number of the program that the PLC should run. The general structure of this XML file is shown below. A step can contain several operations to be executed at the same time (cf. 2 operations for task 2). This file containing the whole production schedule will be sent via OPC to the PLC.

Structure of the XML file sent via OPC to PLC

```
- <Schedule>
  - <Task> task 1
    - <Operation>
      - <Machine> name of the machine <\Machine>
      - <programNumber> number <\programNumber>
    - <\Operation>
  - <\Task>
  - <Task> task 2
    - <Operation>
      - <Machine> name of the machine <\Machine>
      - <programNumber> number <\programNumber>
    - <\Operation>
    - <Operation>
      - <Machine> name of the machine <\Machine>
      - <programNumber> number <\programNumber>
    - <\Operation>
  - <\Task>
- <\Schedule>
```

With the demonstration scenario, 4 scheduling trees have been created and a valid schedule with a duration of 629 seconds has been found within less than 0.4 seconds.

```
Run Main
C:\Users\Samsung\Anaconda3\python.exe D:/Desktop/FINAL/OperationPlanningThesis/src/Main.py
List of available product:
0 carProcess

Which process do you want to perform? Type the corresponding number: 0
Do you want to perform any other process? If no, please press key Z: Z

Mapping...
The result of the mapping can be seen in mappingFile.txt

Ordering of steps...

Creation of the scheduling tree...
The trees can be seen in src folder

Depth First Search algorithm running...
Exploring New Tree
Exploring New Tree
Exploring New Tree
Exploring New Tree

FINAL SCHEDULE: [[<Tree.Node object at 0x0000021F9D18F208>, '23', [[<ProductionPlan.Product c
DURATION: 629
The final schedule can be seen in scheduleFile.txt

Export schedule to XML file
--- 0.32726025581359863 seconds ---

Process finished with exit code 0
```

Figure 5.7: Console view for the demonstration scenario (screenshot)

The results obtained for the testbed setup are illustrated in the Gantt diagram in figure 5.8. Only the first steps of the production plan are represented: Pick the box process, Chassis assembly, and Motor assembly. Operation along with the machine on which it is scheduled is indicated on the vertical axis. Its corresponding execution time is represented on the horizontal axis.

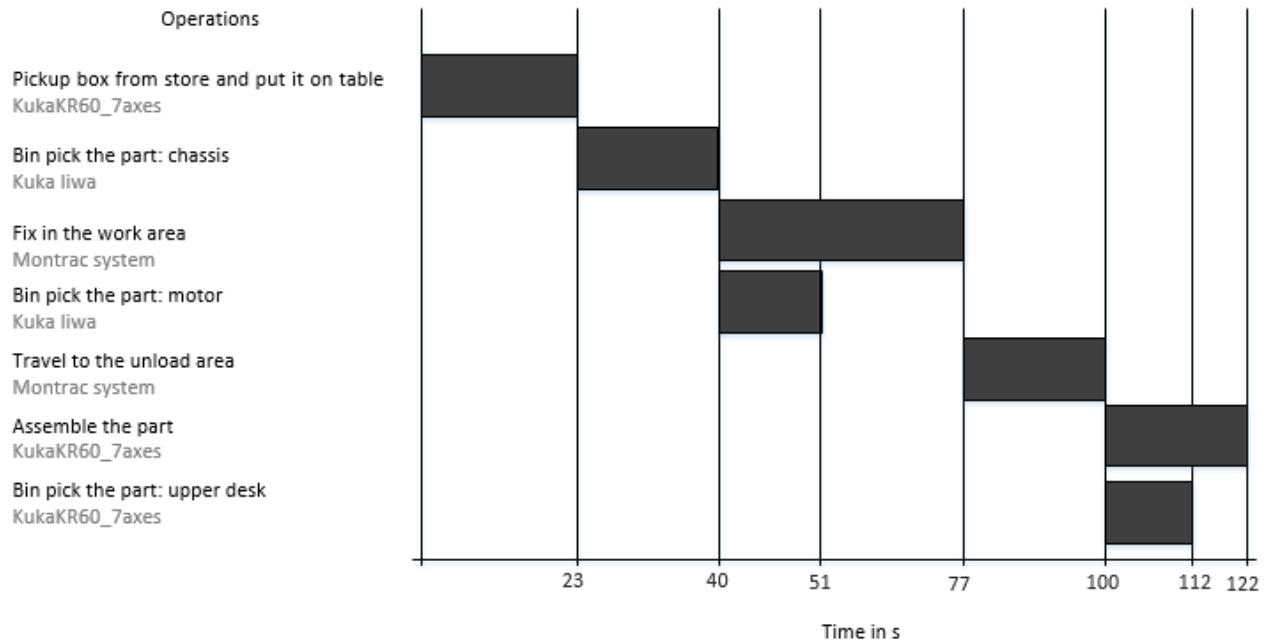


Figure 5.8: Result of the scheduling for the testbed scenario

As only one capability for each operation has been implemented in Process Simulate, the computation of the schedule with the Python program is really straightforward. Nevertheless, it also works fine for more capabilities, as for example two robots being able to do the same operation. The only difference is that the algorithm takes more time (i.e. several minutes) since the created tree is exponential. However, even if the computation time of the algorithm is in minutes, it is still interesting to use it as long as the production plan does not change too often.

With the testbed setup, I showed that a valid production schedule can be generated without configuring manually the factory setup. The same algorithm can be applied for different production plans, but also for different factory setups. If the production line is modified, only the XML file containing the factory setup must be updated by adding for example new machines or new capabilities. Therefore, flexibility of the production line has been increased.

5.3 Evaluation of virtual commissioning

As I did not manage to establish the connection between TIA portal and Process Simulate, I evaluated both parts separately. I simulated the TIA portal program by forcing manually variables coming from the production line (i.e. Process Simulate), and similarly I run the Process Simulate simulation by forcing variables as if they were controlled by the PLC.

5.3.1 Evaluation of Process simulate implementation

In Line Simulation mode of process simulate, two types of simulation can be performed:

- Cyclic Event Evaluation (CEE) which uses the internal PLC of Process Simulate for controlling the event-based simulation.
- PLCSIM emulation: Event-based simulation is driven from actual programmable logic controller (PLC) code.

The setting of both simulation types is done in PLC section of the Options panel (see figure 5.8).

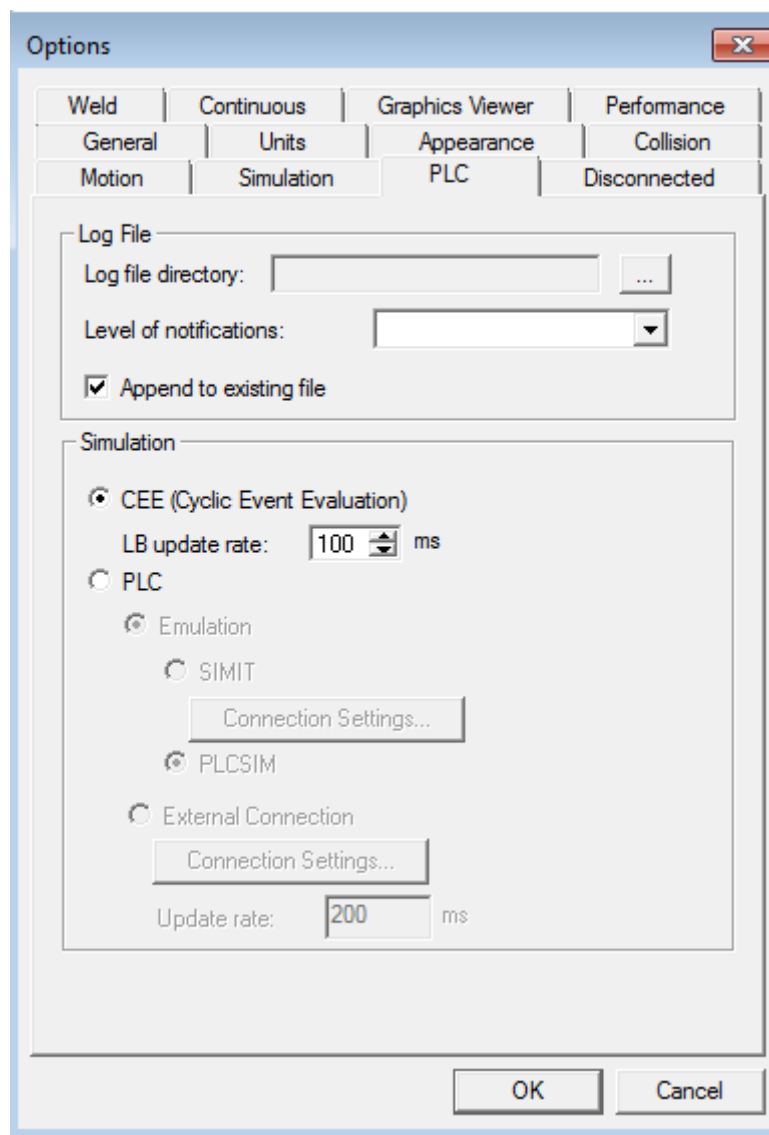


Figure 5.9: Setting of CEE / PLCSIM simulation

5.3.1.1 CEE simulation

The event-based simulation has been firstly tested with the CEE mode. I used the Simulation Panel for monitoring the signals chosen from Signal Viewer manually. The simulation panel with the signals used for supervising the simulation is illustrated in figure 5.9. By forcing the value of *Process_ProgramNumber* to some given number, it is possible to trigger the compound operation which has the same transition condition. For triggering a robot program, the program number has to be forced to a chosen number and the start signal has to be forced to TRUE. As the robot is starting its program on a rising edge of startProgram signal, the box must be deselected after that, otherwise the robot does not move.

When running the simulation, all the inputs signals from the viewpoint of the PLC were acting correctly. For example, after the end of a compound operation, the Boolean *Process_end* is set to TRUE. Similarly, robots were mirroring the right number and set *programEnded* to TRUE after the execution of their program.

Simulation	Inputs	Outputs	Forced	Forced Value
RobcadStudy				
StartingSwitch_end	■		<input type="checkbox"/>	■
Process_ProgramNumber		0	<input type="checkbox"/>	0
Process_end	●		<input type="checkbox"/>	■
liwa_startProgram		●	<input type="checkbox"/>	■
liwa_programNumber		0	<input type="checkbox"/>	0
liwa_programEnded	●		<input type="checkbox"/>	■
liwa_mirrorProgramNumber	0		<input type="checkbox"/>	0
liwa_errorProgramNumber	●		<input type="checkbox"/>	■
kr60ha7axes_startProgram		●	<input type="checkbox"/>	■
kr60ha7axes_programNumber		0	<input type="checkbox"/>	0
kr60ha7axes_programEnded	●		<input type="checkbox"/>	■
kr60ha7axes_mirrorProgramNumber	0		<input type="checkbox"/>	0
kr60ha7axes_errorProgramNumber	●		<input type="checkbox"/>	■
kr60ha_startProgram		●	<input type="checkbox"/>	■
kr60ha_programNumber		0	<input type="checkbox"/>	0
kr60ha_programEnded	●		<input type="checkbox"/>	■
kr60ha_mirrorProgramNumber	0		<input type="checkbox"/>	0
kr60ha_errorProgramNumber	●		<input type="checkbox"/>	■

Figure 5.10: Simulation panel for CEE

The video of event-based simulation in CEE is attached in the CD of the thesis (see appendix B). The video only records the first steps of the production plan: Pick the box process, Chassis assembly, and Motor assembly.

However, during the CEE simulation the Part Appearances feature was not working properly. Indeed, parts were not following the right material flow as defined earlier. Therefore, parts have not been generated during the video recording.

5.3.1.2 PLCSIM simulation

The second simulation has been performed by emulating the PLC behaviour with PLCSIM. Address of signals must be defined in order to monitor them from the PLCSIM software. Figure 5.10 shows the addressing for signals and robot status signals. The address of signals must be the same as the ones defined in TIA portal (see PLCTags Excel file) to be able to monitor them directly from the PLC.

Signal Name	Memory	Type	Address	IEC Format	PLC Connection	Resource
StartingSwitch_end	<input type="checkbox"/>	BOOL	No Address	No Address	<input type="checkbox"/>	
Process_ProgramNumber	<input type="checkbox"/>	INT	4	Q4	<input checked="" type="checkbox"/>	
Process_end	<input type="checkbox"/>	BOOL	0.1	I0.1	<input checked="" type="checkbox"/>	
liwa_startProgram	<input type="checkbox"/>	BOOL	0.0	Q0.0	<input checked="" type="checkbox"/>	● liwa
liwa_programNumber	<input type="checkbox"/>	BYTE	2	Q2	<input checked="" type="checkbox"/>	● liwa
liwa_programEnded	<input type="checkbox"/>	BOOL	0.0	I0.0	<input checked="" type="checkbox"/>	● liwa
liwa_mirrorProgramNumber	<input type="checkbox"/>	BYTE	2	I2	<input checked="" type="checkbox"/>	● liwa
liwa_errorProgramNumber	<input type="checkbox"/>	BOOL	0.2	I0.2	<input checked="" type="checkbox"/>	● liwa
liwa_robotReady	<input type="checkbox"/>	BOOL	0.3	I0.3	<input checked="" type="checkbox"/>	● liwa
liwa_at_HOME	<input type="checkbox"/>	BOOL	0.4	I0.4	<input checked="" type="checkbox"/>	● liwa

Figure 5.11: Addressing of signals in Signal Viewer

Figure 5.11 lays the emphasis on how to monitor and control the signals from PLCSIM and the effect in Process Simulate: the robot signal *liwa_startProgram* is triggered from PLCSIM by clicking the box Q0.0. PLCSIM also allows to watch input signals, as for example *liwa_at_HOME* and *kr60ha7axes_at_HOME* signals are TRUE (box corresponding to signals with address I0.4 and I0.6 are ticked) and *liwa_programEnded* is FALSE (box with address I0.0 is not marked).

Once the connection is established, it leads to the same previous simulation.

Simulation	Inputs	Outputs	LB	Forced	Forced Value	Comment
RobcadStudy						
liwa_startProgram		<input checked="" type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>	
liwa_at_HOME	<input checked="" type="checkbox"/>			<input type="checkbox"/>	<input checked="" type="checkbox"/>	
kr60ha7axes_at_HOME	<input checked="" type="checkbox"/>			<input type="checkbox"/>	<input checked="" type="checkbox"/>	
liwa_programEnded	<input checked="" type="checkbox"/>			<input type="checkbox"/>	<input checked="" type="checkbox"/>	

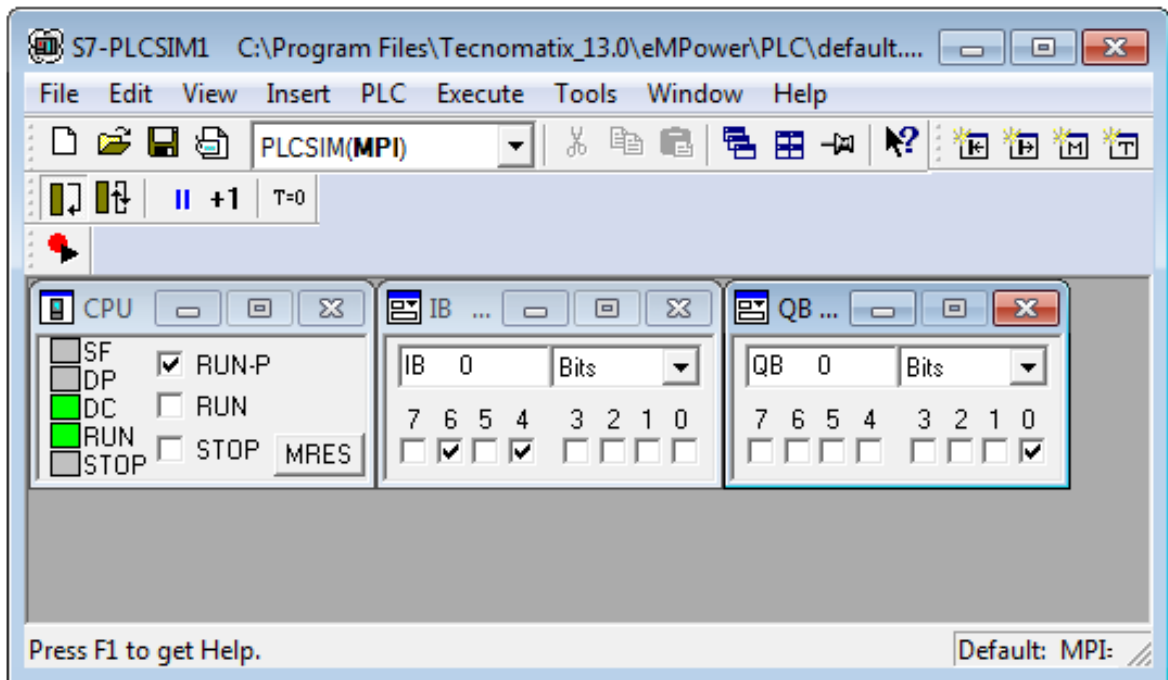


Figure 5.12: Monitoring signals from PLCSIM

5.3.2 Evaluation of PLC program

Finally, PLC program has been evaluated. As PLCSIM advanced emulates the PLC, CPU program can be download to the virtual device, and then it can be simulated with the online mode.

A watch table enables to monitor and control the desired signal. In Figure 5.12, the signals used for simulating the bin picking operation in online mode are represented. In order to simulate the real robot behaviour with real communication between the PLC and robot controller, all robotic input signals have been forced to TRUE (e.g. *iiwa_at_HOME* or *liwa_at_HOME*). Moreover, for simulating the fact that the schedule is sent from OPC UA, programNumber is forced to the number corresponding to one of the bin picking operations: in this case, the number 3 was chosen corresponding to the operation “bin pick the upper desk”.

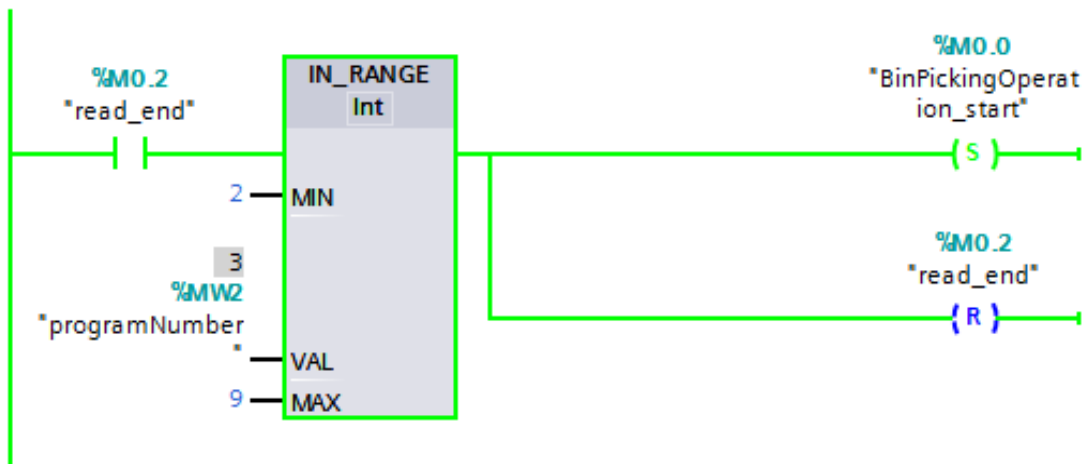
coiffann ▶ PLC_1 [CPU 1516-3 PN/DP] ▶ Watch and force tables ▶ Watch table_1

	i	Name	Address	Display format	Monitor value	Modify value	⚡
1		"liwa_programEnded"	%I0.0	Bool	<input checked="" type="checkbox"/> TRUE	TRUE	<input checked="" type="checkbox"/>
2		"liwa_at_HOME"	%I0.4	Bool	<input checked="" type="checkbox"/> TRUE	TRUE	<input checked="" type="checkbox"/>
3		"liwa_robotReady"	%I0.3	Bool	<input checked="" type="checkbox"/> TRUE	TRUE	<input checked="" type="checkbox"/>
4		"liwa_errorProgramNumber"	%I0.2	Bool	<input type="checkbox"/> FALSE		<input type="checkbox"/>
5		"liwa_mirrorProgramNumber"	%IW2	DEC+/-	3	3	<input checked="" type="checkbox"/>
6		"liwa_startProgram"	%Q0.0	Bool	<input checked="" type="checkbox"/> TRUE		<input type="checkbox"/>
7		"Process_end"	%I0.1	Bool	<input checked="" type="checkbox"/> TRUE	TRUE	<input checked="" type="checkbox"/>
8		"BinPickingOperation_start"	%M0.0	Bool	<input type="checkbox"/> FALSE		<input type="checkbox"/>
9		"programNumber"	%MW2	DEC+/-	3	3	<input checked="" type="checkbox"/>
10		"Process_ProgramNumber"	%QW4	DEC+/-	3		<input type="checkbox"/>
11		"liwa_programNumber"	%QW2	DEC+/-	3		<input type="checkbox"/>

Figure 5.13: Watch table in online mode

The simulation is running correctly as output values (e.g. *liwa_startProgram* or *Process_ProgramNumber*) are set to the correct value. In figure 5.13, OB 1 in online mode is highlighted: with the previous variable forced in the watch table, the output *operation_done* of the bin picking function block is set to TRUE.

Network 2: check bin picking operation program number



Network 3: Bin Picking Operation liwa

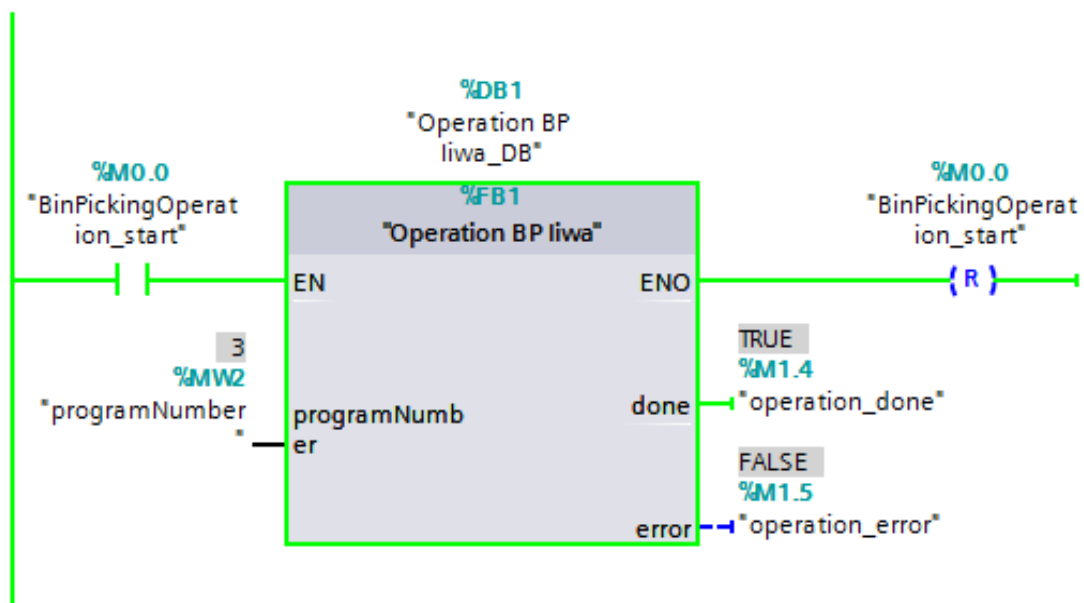


Figure 5.14: OB 1 in online mode

Conclusion

In order to increase the flexibility of production systems, I have implemented a Python algorithm that generates a valid schedule based on a distinct capability-based description of the production plan and the current factory setup. Available resources are firstly mapped to the different operations of the production plan and then a scheduling tree is created. A depth-first search algorithm with backtracking goes through this tree for finding a schedule optimized according to the shortest schedule duration criteria. The contribution of the thesis is that material flow is checked between each node of the tree: location of the required material for execution of operation of a given node is found, and the algorithm determines if some intermediate transport steps are needed for transporting the material from their respective locations to the current workspace.

Some enhancements regarding the mapping need to be done as capability of a machine to execute some operation is only described by the process name and the material it should handle for now. Tools with their attributes have to be defined in both the production plan and factory setup. It is also possible to improve the DFS algorithm since it sometime does not return the most optimal valid schedule.

Then, the generated schedule should be sent via OPC UA to the PLC which controls and monitors the whole production line according to the schedule. The OPC UA communication has not been implemented yet.

Virtual commissioning of the production line has also been performed. Model of the production line has been used to create an event-based simulation in the Process Simulate software. This simulation is based on logic and signal events, and can be driven from actual PLC code. Therefore, the complete behaviour of the production line can be emulated. Two simulations have been performed: the first one uses the internal PLC of Process Simulate for controlling the event-based simulation (CEE) and the second one uses the PLCSIM software for emulating the PLC behaviour. Both simulations validated the implementation of the production operations in the available production resources.

PLC program has also been implemented. Function block for each operation of the production plan has been designed according to the machine involved in the given operation. By using PLCSIM Advanced software, it was possible to emulate the CPU S7-1500 and thus to run the program in online mode. Once again, this simulation allowed the validation of the control system and more particularly the implemented function blocks by triggering with the watch table the input signals such as robot status or the operation status, but also by forcing their starting condition (i.e. the number of the program in the schedule file to run transmitted via OPC UA).

The OPC communication between Process Simulate and the PLC (TIA portal) has not been established yet. Therefore, the whole virtual commissioning has not been validated yet since this interface is missing.

References

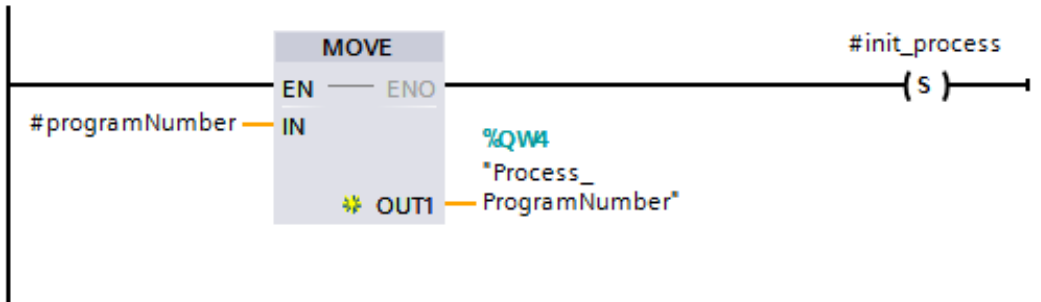
- [1] M.-F. Zah, M. Beetz, K. Shea, G. Reinhart, O. Stursberg, M. Ostgathe, C. Lau, C. Ertelt, D. Pangercic, T. Ruhr et al., "An Integrated Approach to Realize the Cognitive Machine Shop," in Proceedings of the 1st International Workshop on Cognition for Technical Systems, 2008, pp. 6–8.
- [2] Keddis, N., Kainz, G., and Zoitl, A. (2014). Capability based Planning and Scheduling for Adaptable Manufacturing Systems. In *IEEE International Conference on Emerging Technologies & Factory Automation (ETFA)*.
- [3] Keddis, N., Kainz, G., Zoitl, A., Knoll A. (2015). Modelling Production Workflows in a Mass Customization Era. In *IEEE International Conference on Industrial Technology (ICIT)*.
- [4] Zoitl, A., Kainz, G., and Keddis, N. (2013). Production Plan-Driven Flexible Assembly Automation Architecture. In *Industrial Applications of Holonic and Multi-Agent Systems*, 49-58. Springer.
- [5] Keddis, N., Kainz, G., Buckl, C., and Knoll, A. (2013). Towards Adaptable Manufacturing Systems. In *IEEE Int. Conf. on Industrial Technology (ICIT)*, 2013, 1410-1415. IEEE.
- [6] Documentation OPC UA .NET Client for the SIMATIC S7-1500 OPC UA Server
<https://support.industry.siemens.com/cs/ww/en/view/109737901>
- [7] Siemens Tecnomatix Process Simulate, User documentation, 2017
- [8] Simatic S7 documentation
https://cache.industry.siemens.com/dl/files/056/18652056/att_70829/v1/S7prv54_e.pdf
- [9] Programming Guideline for S7-1200/1500 STEP 7 (TIA Portal)
https://www.industry.siemens.nl/automation/nl/nl/industriële-automatisering/industrial-automation/simatic-controller/modulaire-controllers/simatic-s7-1500/Documents/81318674_Programming_guideline_DOKU_v12_en.pdf

Appendices

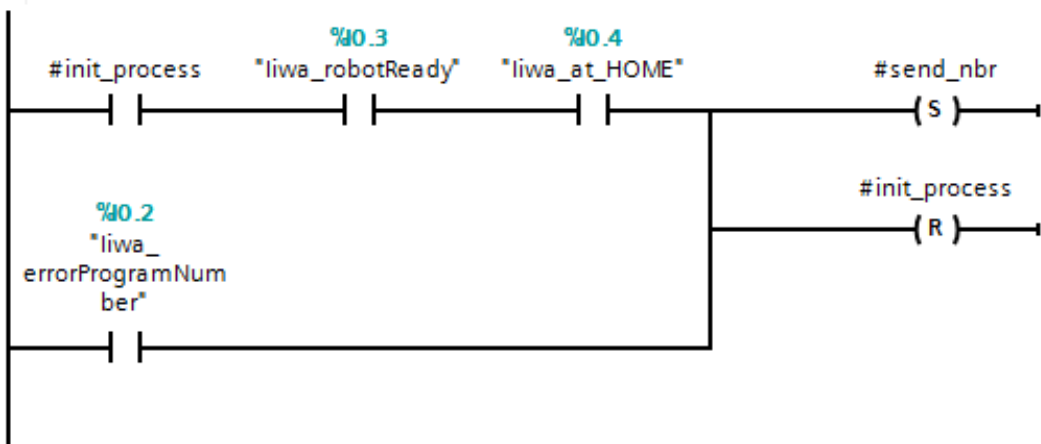
Appendix A: Function block of the bin picking operation

▶ **Block title:** Operation BinPick the part

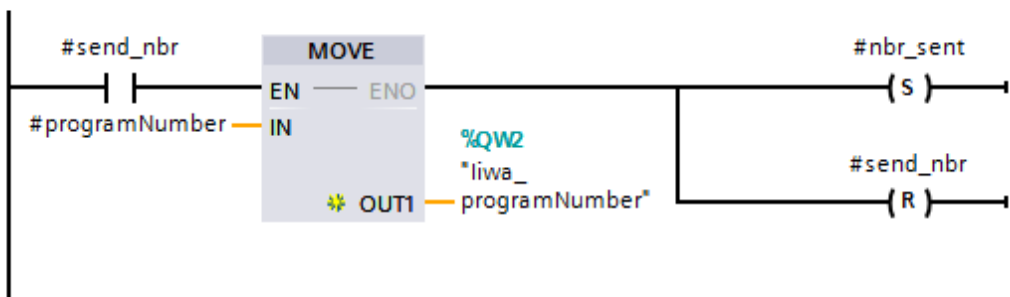
▼ **Network 1:** send program of the process



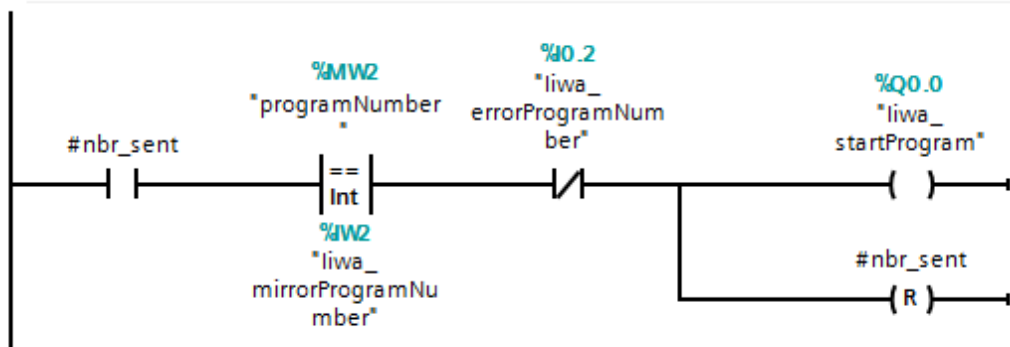
▼ **Network 2:** Send program number to robot



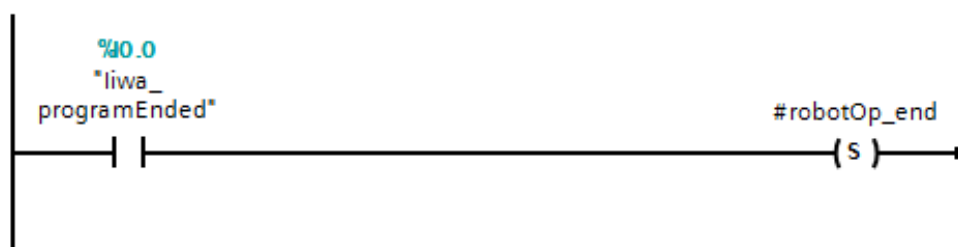
▼ **Network 3:** Send program number to robot



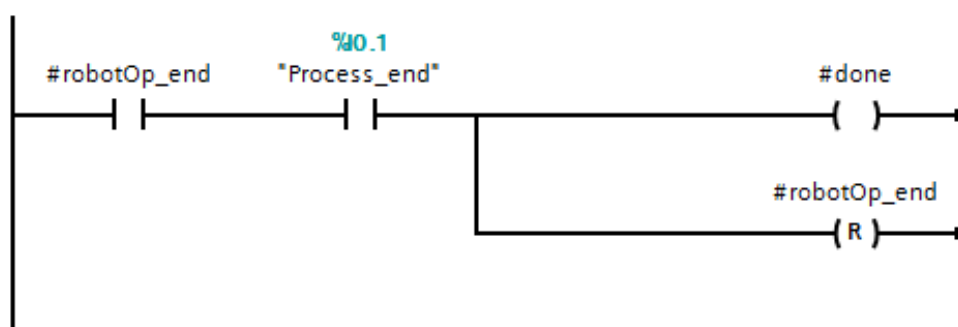
▼ **Network 4:** send start command to robot if program number ok



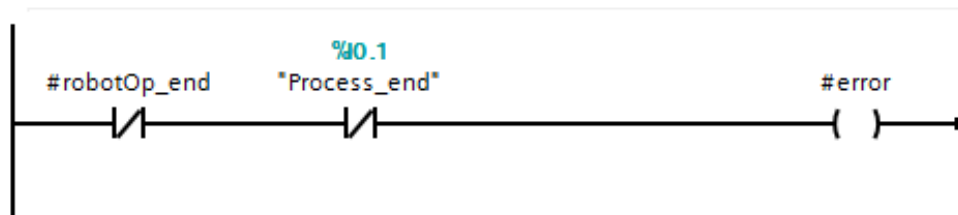
▼ **Network 5:** end robotic operation



▼ **Network 6:** compound operation finished without error



▼ **Network 7:** compound operation finished with error



Appendix B: CD content

Thesis.docx	Diploma thesis report in Word format	
Thesis.pdf	Diploma thesis report in PDF format	
Folder 1: Production Schedule Algorithm		
Folder 1.1: Src		
Python source files		
Folder 1.1.1: ProductionPlan		Excel and PERT file generated by TeamCenter
factorySetup.xml		XML file describing factory setup
Folder 1.2: Output_files		Files generated by Python program
Folder 2: TIA portal		
coiffann.ap14	TIA portal project	
PLCTags.xlsx	Excel file containing PLC tags defined in CPU	
Folder 3: Process Simulate		
Original_study.psz	Original PS study	
Final_study.psz	Final PS study	
Time-based_simulation.mp4	Video of time-based simulation	
CEE_simulation.mp4	Video of CEE simulation	
Material_Flow.jpg	Material Flow Viewer exported from Process Simulate	