CHALMERS

Visualize an event-based simulation model made in Process Simulate

Master of Science Thesis performed for Volvo Cars Corporation, Sweden

ANDREAS LARSSON
HENRIK NILSSON

Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2011
Report No. EX044/2011

REPORT NO. EX044/2011

Visualize an event-based simulation model made in Process Simulate

ANDREAS LARSSON
HENRIK NILSSON

Examiner and supervisor at Chalmers University of Technology:
Petter Falkman pf@chalmers.se
Department of Signals & Systems
CHALMERS TEKNISKA HÖGSKOLA
412 96 Göteborg

Supervisor at Volvo Car Corporation:
Mathias Sundbäck msundbac@volvocars.com
Manufacturing Simulation
Volvo Car Corporation
Dept. 81320/PV2A2
SE-405 31 Göteborg
Sweden

Department of Signals and Systems
Division of Automatic Control, Automation and Mechatronics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2011

Visualize an event-based simulation model made in Process Simulate
ANDREAS LARSSON
HENRIK NILSSON

Technical report no EX044/2011
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Chalmers Reproservice
Göteborg, Sweden 2011

Abstract

This master's thesis has been conducted at Volvo Car Corporation at the department for Manufacturing Simulation. The goal was to extract information regarding operations and their associated logic from Process Simulate. The information should then be visualized as a logical flow. The purpose was to facilitate for staff at Volvo Cars Corporation, working with the simulation model, to easier understand the logic in the model and to ease the communication between simulation engineers and other staff involved. This decrease error sources and development time. Since preparations in simulation often is carried out by different engineers with different types of software, visualization can increase and improve the communication between different departments.

Several approaches to different problems have been evaluated to find the best solution. The problems included issues regarding how to build the model in Process Simulate, which programming technique to use, how to extract information from Process Simulate and how to visualize the logical flow. Pros and cons have been found for every approach to support the chosen solution better.

An event-based simulation model was built in Process Simulate with operations and all necessary logic to make it functional. Information has then been extracted through the built-in extraction function in Process Simulate and was carefully chosen to avoid too large amount of data exported. Since the focus was on operations and logic, XML files including this information was extracted. An application has been constructed using C# as programming language. The purpose with the application is to create files containing visualized logical flows of the operations in the simulation model. The files created are executable in Microsoft Visio or in Sequence Planner, a tool developed at Chalmers University of Technology.

The master's thesis project were also supposed to contain an evaluation regarding the possibility of importing data to Automation Designer from Process Simulate, a software tool developed by Siemens. However, communicating with the developers of Automation Designer, gave the authors knowledge that the functionality in Automation Designer of importing information regarding operations and logic is absent. Therefore this is not raised in the report as much as other initially stated aims.

A few future recommendations are also included for VCC and Chalmers to take part of. The recommendations is made through reflections the authors have encountered during the thesis work as issues regarding the event-based approach of building a simulation model and how a connection between Process Simulate and a visualization tool can be established.

This report is written in English.

Keywords: *sequence of operations, simulation model, extraction, event-based, visualization, flow chart, XML, Process Simulate, Process Designer, Sequence Planner, C#, VCC*

Acknowledgments

First we would like to thank our two supervisors, Mathias Sundbäck at VCC and Petter Falkman at Chalmers. Mathias has been a great support through the whole project. From the beginning when we struggled with the functionalities in Process Simulate to the end when we needed input regarding the visualization. It has always been easy to ask questions about everything from VCC standards to barefoot running and chia seeds. Petter Falkman introduced us to this project and has been a great support throughout the work for discussing different aspects about how the project should be structured.

We would also want to thank Peter Bodin at Siemens. Peter was a great support in the beginning of the project when the simulation model was built. Regarding the understanding of the data extracted from Process Simulate we would like to thank Kurt Van Damme at VCCs office in Gent. He has really put effort to explain his knowledge about the XML structure within the exported files.

Nina Sundström and Fredrik Westman have previous done projects involving Process Simulate and have been a good support in the project. Nina helped us a lot in the beginning with all from the installation to the basic understanding. Fredrik who in 2008 developed an plug-in that extracted information from Process Simulate have helped us with the functionalities in that plug-in and which part of it that we could use.

During the development of the application a lot of questions regarding how to structure the program came up. When solving these questions Linus Helgesson at Semcon has been a great support. We would also want to thank Patrik Magnusson at the department of Signals and Systems at Chalmers for the continuous improvements of Sequence Planner according to many of our wishes.

We want to thank Per Nyqvist at the department of Product and Production Development at Chalmers for initial help with the software needed to start this thesis. Per showed a great load of enthusiasm for us as students.

Finally we would like to thank all great colleagues at the Manufacturing Simulation department at VCC. They have been a great support at the floorball games every Friday. Even if they usually won when we did not participate.

An engineer's most important function is often described as the natural bridge between different areas of expertise. - *Unknown*

Contents

List of Figures

List of Tables

Nomenclature

API	Application Programming Interface
CAD	Computer-aided Design
CSV	Comma-separated values
DOM	Document Object Model
IPA	In-Process Assembly
OLP	Off-Line Programming
PD	Process Designer
PLC	Programmable Logic Controller
PS	Process Simulate
SAX	Simple API for XML
SE	Simultaneously Engineering
VCC	Volvo Car Corporation
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformations

Chapter 1

Introduction

This master's thesis has been conducted at VCC, in cooperation with the department of Signals and Systems at Chalmers University of Technology, henceforth called Chalmers. The thesis presents a way to visualize logic from a simulation model made in Process Simulate. This will be achieved by interaction with two softwares, Sequence Planner and Automation Designer. The scope of the thesis also includes building a station with robot operations, kinematics, OLP commands and logic.

1.1 Overview

Process Simulate is a simulation tool developed by Siemens that is used in manufacturing engineering. It handles issues like reachability, zone allocations and avoidance of interlocks. Each operation in Process Simulate can be given logic and conditions to trigger different events. Process Designer is just as Process Simulate a tool developed by Siemens used in the simulation area, but can be used in another fields within simulation. In this thesis, it is used to handle the material flow. Sequence Planner is a software tool developed by Chalmers. Its purpose is to visualize sequences of operations in a more user-friendly way than simulation software does and also be able to check for deadlocks. Since the software still is in an early development phase, all functions are not fully formed and therefore not in use. Automation Designer is a software tool from Siemens which can be useful for implementing a more virtual commissioning thinking in a manufacturing environment. It can be seen as the last step before implementation of the preparation work in the physical station.

1.2 Background

Preparing an automatic production system based on a product and the required activities for fulfilling the product specification involves many different tasks. These activities range all the way from early design of the system (SE-phase) to final preparations of control functions for the equipment such as robots, turntables, fixtures etc. (OLP level). There are also additional requirements on the prepared production system. The most important ones are cost, cycle time, collision prevention, reachability and operation sequence. Cycle time shall not only be short but it should be the same as the rest of the systems so that a balanced flow through the entire manufacturing plant is achieved. Many factors influence the final cycle time and these factors have to be dealt with. When a satisfying operation sequence is implemented, the cost for production preparations is reduced. To design an optimal operation sequence many aspects have to be considered, such as product requirements cycle time, potential collisions, reachability etc.

Today, there is no easy way for the simulation engineers to visualize different sequences of operations before delivering the simulation model to the next phase. This increases the risk for misunderstanding between them when data exchange is performed. A problem is also the communication with the PLC engineers. By just handing over a simulation model gives them little information about the actual system.

1.3 Purpose

Visualization of sequences of operations is done in order to decrease error sources and development time. Preparations in simulation are often carried out using different types of software and by different people. This increases the risk of information getting lost between different users. Thus, the purpose of this thesis is to make a visualization standard to better show sequences of operations with including logic of a station. It is also done in order to evaluate functions in an event-based simulation so that VCC are able to develop their working methods. Since a few line builders have started to use an event-based approach, it is natural for VCC to start using it to a larger extent. In the end, this will save time and money by facilitating the exchange of information.

1.4 Aims

The main goal of this thesis is to extract relevant information from Process Simulate and structure it in a way which makes it possible to import it into Sequence Planner and Automation Designer. To reach this goal, a few sub-goals can be stated:

- Define enough kinematics, logic and robot commands in Process Simulate so it can be extracted.
- Find a way to extract the wanted information from Process Simulate.
- Structure the extracted information so it can be imported into Sequence Planner.
- Visualize sequence operations and station logic in Sequence Planner.
- Review the possibility to enable transfer from Process Simulate to Automation Designer.

1.5 Rules of procedure

The thesis work has been divided in three main parts:

- Build an event-based simulation model in Process Simulate
- Examine possibilities for data extraction from Process Simulate
- Visualize the extracted sequence of operations

To be able to evaluate how to visualize the logic in a current cell a simulation model of a station has to be built according to the VCC standards. Creating the model according to VCC standards will make the solution for extracting the logic general for VCC models. Before visualizing the cell methods for visualizing sequences has to be evaluated. The different methods will be compared and benchmarked with restrict to what VCC want to achieve and what should be visualized. Different methods will also be used to extract the data from the simulation model.

To set up the material flow, logic and kinematics in Process Simulate, VCC standards will be used to make it easier to follow. The current existing logic blocks will be evaluated to see if any standard blocks interfere with each other in any way. In that case, new blocks might need to be created. To then export the wanted information from Process Simulate, at least two different methods will be tested. The first method is simply to use the default export tool in Process Simulate and edit the information outside Process Simulate before importing it into Sequence Planner. The second method is to use a plug-in to Process Simulate, created by previous master's thesis students at VCC, to extract exactly the information that is requested. This is to hopefully get a smoother import into Sequence Planner.

The visualization in Sequence Planner will be done according to how VCC wants it to be visualized. If it is not possible due to too slow development process of Sequence Planner, the logic will instead be structured in a way so it can be imported into Sequence Planner when it is further developed.

It will be investigated whether it is possible to import the logic from Process Simulate to Automation Designer. It will then be evaluated which logic that Automation Designer can import

and which information is not included in the import. It can then be stated how Automation Designer needs to be improved to better work with Process Simulate.

1.6 Limitations

The time scope is of course the biggest limitation of the thesis. In 20 weeks, both a concrete solution with a physical result consisting of a visualization tool that can be used by Process Simulate and a report, which describes the work done, should be produced.

No major workload in this thesis project has been on optimizing information in Sequence Planner for importing it back to Process Simulate. Developing Sequence Planner and Automation Designer is outside the scope of this thesis.

For the visualization method found in this project to work properly, the simulation model has to be built according to VCC standards in 2.1.1.

The simulation model was achieved when it was not totally clear which parts that should be assembled in this station. Therefore it was not able to reflect the reality fully.

1.7 Assumptions

It is assumed that the reader of this report has experience of working in Process Simulate.

The manufacturing cell used consists of some equipment that is not supposed to be in the physical station. For instance there are a couple of welding guns which do not fit correct on the robots. This is because all welding guns are not available as CAD-files yet. It is therefore assumed that these non-correct guns fit on the robots anyway.

It is also assumed that Sequence Planner works as it supposed to do, i.e. all necessary Boolean logic is available.

Chapter 2

Theoretical framework

This chapter will describe the most crucial theory the reader needs to know to be able to understand the report, except basic understanding in Process Simulate.

2.1 Process Simulate

When building a model in PS there are a number of ways to do it. There are two main concepts regarding a simulation model; event-based and time-based. Event-based simulation is when the model only considers which signals that are true and false before performing an operation. Time-based simulation on the other hand does not consider signals; instead it considers which operations that has happened. As can be imagined these two methods can be combined in many ways. To create a event-based simulation in a more time-based way you can have the the operations setting signals when performed which then can trigger the next operation to happen. PS can handle both concepts and the fact that they can be combined like this makes the possibility for different solutions almost infinite. This implies that there has to be a well structured working order to be able to work with the model.

The following subsections contain information that are crucial when building a event-based simulation model at VCC. [11]

2.1.1 VCC standards

To make sure that all simulation models are built in a structured way, VCC have come up with a few standards. The standards cover everything from how to build the logic for the robots to how to build the material flow.

Signals

To communicate with the robots there are a number of standardized signals. The signals are standardized to get a system with signals that are as like the reality as possible. There are a total of eight standardized signals: [4]

- LockNumIn: Interlock signal to inform the robot if it for example can enter a zone or not. Used in the OLP command WaitSignal.
- LockNumOut: Interlock signal used for the robot to request zones from the PLC. Used in the OLP command WaitSignal.
- SmallDress: Order the robot to perform a small dress.
- BigDress: Order the robot to perform a big dress.
- CycleDone: Informs the PLC when a cycle has been performed.
- DressDone: Informs the PLC when the dressing is completed.

- DirectionIn: Case signal to inform the robot which case to perform. Used in the OLP command WaitCase.
- DirectionOut: Case signal used for the robot to check with the PLC which case to perform. Used in the OLP command WaitCase.

Signal operations

Signal operations as OLP commands are operations that are used by the robots to communicate with the PLC. The operations always communicate on two signals, one input (LockNumIn or DirectionIn) and one output (LockNumOut or DirectionOut).

WaitSignal This operation is used when the robot wants to either enter a zone or a station. The operation sends a predefined integer value on the output signal LockNumOut and waits to get the same integer value as a response on LockNumIn before entering the zone or station. After leaving the zone/station the robot clears this by sending the integer for the request plus one. If the robot for example has requested zone1 (21) it will send 22 on LockNumOut after leaving the zone. For this to work there are standard integer values depending on what is requested:

- 1, 3, 5... : Station 1, Station 2, Station 3 is allocated
- 2, 4, 6... : Station 1, Station 2, Station 3 is released
- 21, 23, 25... : Zone 1, Zone 2, Zone 3 is allocated
- 22, 24, 26... : Zone 1, Zone 2, Zone 3 is released

The WaitSignal is realized by selecting WaitSignal as an OLP command of the robot.

WaitCase This operation is used when you want the robot to make a logical choice, for example if a robot both loads a part into a fixture and then welds it. It is then possible for the robot to perform either the welding operation or the loading operation depending on if there is a part in the fixture. The WaitCase sends a signal on the output DirectionOut and then waits for which case to perform. In the load/weld example the robot should get either 1 or 2 as response on the input DirectionIn. As the WaitSignal, the WaitCase is also implemented in the OLP command. It is realized by selecting WaitCase as an OLP command. [4]

Logic Blocks

They are used as a simulated external PLC. They contain entries, exits and logical functions. The entries are signals from the robots to the external PLC and exits are signals from the external PLC back to the robot. The logical expressions determine how the entry signal will be processed to get the exit signal. Logic blocks can be imported into a new study or a new project, which means that they can be used over again. VCC has set up a few standard logic blocks that deal with common used functions in PS as zone allocation and robot start signals. [4]

Allocate_Zone This is the logic block that keeps track of which zones that are free to enter and which ones that are taken. The Allocate_Zone block is the block that gets the signal from the WaitSignal operation. When retrieving a request for a zone the Allocate_Zone block checks if it is free and returns the same integer value as soon as the zone is free to enter. It uses the signals LockNumIn and LockNumOut for communication with the robots in the simulation model. [4]

Robot_StartBlock This block is used to give the robot a signal to start the robot program. The logic behind this block is quite simple. It simply gives the robot a starting signal the first time the program runs and then every time the robot has finished its previous program. This makes the robot loop its program. [4]

Standard operations

ToolChange ToolChange specifies a sequence for performing a tool change between two tools. It is realized as a compound in the simulation model. The sequence needs some standard locations in order to work. These locations also need to be named in a ABB robot standardized way for the module to know which one to take in which order. The names and the function of each location can be seen in Table 2.1.

Table 2.1: The different operations in the ToolChange compound.

OPERATION	DESCRIPTION
HomeToStand	Moves the robot from the home position for the tool to its tool stand.
StandToPutChk	Moves the robot from the tool stand to a position just above.
PutChkToBetw	Moves the robot from the previous position to the home position for the robot.
BetwToStand	Moves the robot from the home position for the robot to the tool stand.
StandToGetChk	Moves the robot from the tool stand to a position just above.
GetChkToHome	Moves the robot from the previous position to the home position for the tool.

Depending on whether the tool is to be returned or picked the ToolChange compound runs different sequences. The sequence HomeToStand -> StandToPutChk -> PutChkToBtw is used for returning the tool, while the sequence BetwToStand -> StandToGetChk -> GetChkToHome is used for fetching the tool. [9]

2.1.2 Sensors

Two sensor types in PS are used at VCC, the proximity sensor and the joint value sensor. The proximity sensor registers if any object is close to or in collision with the sensor. To detect objects they are specified in a list in the properties of the sensor. The sensor must be graphical represented in the simulation model in order to work. It must also be shown in the model (not blanked). The joint value sensor is used to make sure that a kinematic device is in a specified given state. This sensor is most used to get signals when clamps or other kinematic devices are open or closed and it has no graphical representation in the model. For a sensor to be initiated and work properly, it must be used as a transition condition in the Sequence Editor. The simulation must also be executed at LineOperation level for the sensors to make an impact. The sensors are important in building the model using an event-based approach since sensor signals are of great importance to make the model work properly. Without sensors, the model will suffer in events triggering the different operations. [4]

2.1.3 IPA

An IPA is a collection of assembled parts that all together make up a compound of parts. For example, if a car is an IPA, a door and a window can be parts in collection of that IPA. VCC largely use IPAs when building up simulation models, since handling of plain parts has an ability to be severe to manage as parts can be small and insignificant for the simulation model as bolts and screws. To set up an IPA, PD is used to control which parts that are added to the IPA. When the IPA is created and contains the wanted parts, its content can be seen in PD under the IPA Viewer (see Figure 2.1).[4]

More specific information how IPAs have been used in this project can be read in Section 3.1.3 and 4.4.

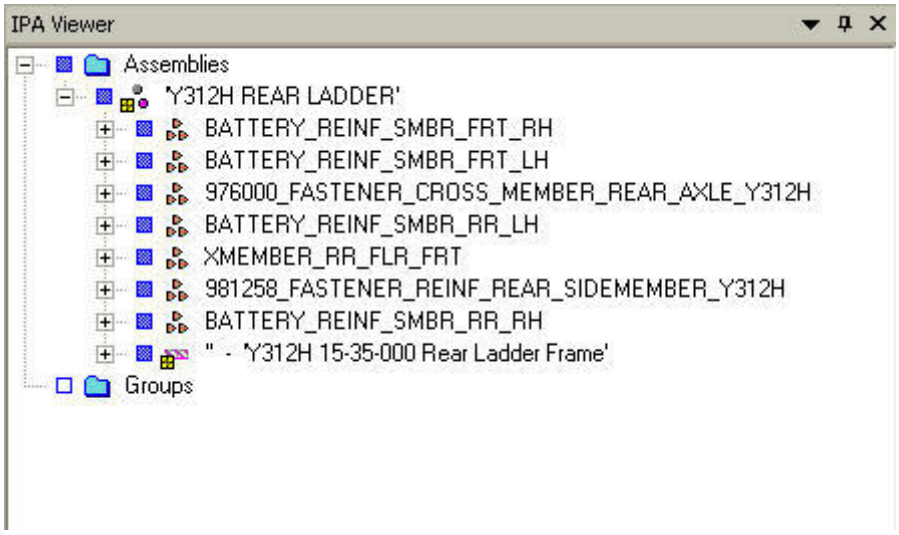


Figure 2.1: Content of an IPA shown in the IPA Viewer in PD.

2.1.4 Non-sim operation

A non-sim operation is a specific type of operation that are used to perform different things that not are directly connected to a resource. One typical thing to use a non-sim operation for is to add instances of material in the simulation model. Even though the name non-sim the operation is still simulated, but it does not perform anything directly to the model. [4]

2.2 Sequence Planner

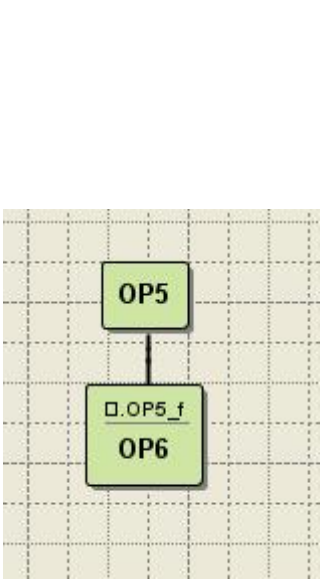
Sequence Planner was founded by two master's thesis students at Chalmers. It has since then been developed at Chalmers with the ambition to be a visualization and an optimization tool for sequences of operations. Regarding the visualization part it is at the moment possible to model a clear view of a sequence. The modeling of the sequence is realized by inputting all resources and operations into the program. When all operations and resources have been defined the relation between them has to be defined. This is done by specifying three parameters:

- Precondition
- Postcondition
- Resource needed

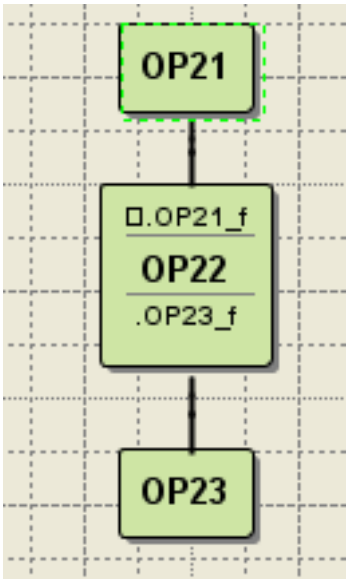
A precondition represents a condition that needs to be fulfilled before an operation being realized. Figure 2.2a shows a very simple relation between two operations, 5 and 6. For operation 6 to be realized, operation 5 needs to be finished. The specification OP5_f is therefore placed on operation 6. A postcondition is contrariwise what needs to be fulfilled after an operation has been realized. Figure 2.2b shows a simple relation where both a precondition and a postcondition are placed on operation 22. As seen in the figures the conditions are placed on operations. When placing the conditions there are three different types of conditions that can be placed:

- _f - Another operation needs to be finished for this operation to be realized.
- _e - Another operation needs to be executing for this operation to be realized.
- _i - Another operation needs to be idle for this operation to be realized.

It is also possible to specify which resources that are needed for an operation to be realized. An example is if a plate is to be welded. In that case both a welding gun and a robot is needed to perform the task, so these two are therefore specified as resources and used by the operation. [6]



(a) A precondition (OP5_f) for an operation to occur.



(b) Pre- and postconditions (OP21_f and OP23_f) to an operation to occur.

Figure 2.2: Simple visualization examples.

2.2.1 Import of data

Sequence Planner has the ability to import data from external programs. This can be done in two ways, XML- and text file . When importing the data Sequence Planner is very strict when it comes to the structure of the imported file. Therefore, XML files must usually be heavily edited before imported into Sequence Planner from a external simulation software like Sequence Planner. There is no standard converter for this to happen automatically.

When it comes to the import of the text file it is a bit of a workaround for simplifying import of data. It has a very simple format that facilitates the understanding for insertion of data for an user. A lot of information might get lost if the format is too simple and excludes a lot of important data. According to the department of Signals and Systems at Chalmers, this issue is under development.

The structure of the text file looks as in Figure 2.3. The idea of Sequence Planner regarding structure is that instances of operations build up the entity framework and that every instance are given a number of attributes.

```
ID:001::NAME:R02_Program_1::PRECON:null::POSTCON:null::DESCRIPTION:null
ID:002::NAME:WaitCase::PRECON:001,2::POSTCON:null::DESCRIPTION: WaitSignal AllocateStation2
ID:003::NAME:R2_Home::PRECON:009,2 OR 015,2::POSTCON:null::DESCRIPTION:null
ID:004::NAME:WaitSignal ReleaseStation2::PRECON:003,2::POSTCON:null::DESCRIPTION:null
ID:005::NAME:WaitSignal LockSignal31::PRECON:004,2::POSTCON:null::DESCRIPTION:null
ID:006::NAME:ToolChange Tool_m02_313::PRECON:002,2 AND 010,0::POSTCON:010,0::DESCRIPTION:Turna
ID:007::NAME:R02_d31120258_PANEL::PRECON:006,2::POSTCON:null::DESCRIPTION:null
ID:008::NAME:R02_d31120214_PANEL::PRECON:007,2::POSTCON:null::DESCRIPTION:null
ID:009::NAME:R02_d31120260_PANEL::PRECON:008,2::POSTCON:null::DESCRIPTION:null
ID:010::NAME:ToolChange Tool_m02_312::PRECON:002,2 AND 006,0::POSTCON:006,0::DESCRIPTION:Turna
ID:011::NAME:R2_CPL_spots_d31120260_CPL::PRECON:010,2::POSTCON:null::DESCRIPTION:null
ID:012::NAME:ToolChange Tool_m02_313::PRECON:011,2::POSTCON:null::DESCRIPTION:null
ID:013::NAME:R2_rear_section_d31120214_CPL::PRECON:012,2::POSTCON:null::DESCRIPTION:null
ID:014::NAME:R02_d31120258_CPL::PRECON:013,2::POSTCON:null::DESCRIPTION:null
ID:015::NAME:R02_d31120258_battery_spots_CPL::PRECON:014,2::POSTCON:null::DESCRIPTION:null
```

Figure 2.3: Instances of operations in the text file imported into Sequence Planner.

The attributes are as seen in capital letters; ID, NAME, PRECON, POSTCON, DESCRIPTION. ID represents an unique id that every instance of an operation is assigned. NAME is the name of the operation. Note that several operations can have the same name. PRECON consists of preconditions and POSTCON consists of postconditions (see Section 2.2). DESCRIPTION describes in pure text the transition that needs to be fulfilled for the operation to occur. Since this is a workaround solution, the postcondition attribute represents not reality. Neither the description attribute represents the reality because the information included is actually precondition information. But this solution takes care of the visualization, which was desired.

Besides the XML and text file import/export Sequence Planner can import and export files to another Chalmers developed software, Supremica, to check for interlocks. In Supremica every state is represented which makes it less suitable for visualization, but in collaboration with Sequence Planner both interlocks and visualization can be done. [6]

2.3 Automation Designer

Automation Designer is a Siemens developed software and is mainly an engineering platform for the manufacturing industry. The main purpose of Automation Designer is to reduce the time needed to import information from the virtual model of a factory to the real factory and also to move the virtual world closer to the reality. The idea of using Automation Designer in this thesis work was to review the possibility of importing operations and their belonging logic from PS into Automation Designer. A previous made master's thesis project investigated if PLC programs could be automatically generated with information from PS in Automation Designer, so that analysis was left outside the scope of this thesis.

2.3.1 Sequences of operations

As in PS, Automation Designer contains a navigation tree that enables the user to navigate between objects in the hierarchy of the project. Unfortunately, operations can neither be created nor imported from another application. According to developers at Siemens, this function will be implemented in the next big release that is calculated to be released in mid-2012. Though, in Automation Designer there is a feature called Sequence Designer where it is possible to create actions with signals, but exports from PS can not be imported here.

To make the imported operations and their logic in Automation Designer useful for VCC, it must not only be possible to import operations from the Gantt chart in PS but also the schedulers which contains information about the sequence of operations for each robot. [3]

2.4 XML handling

A XML document is a well structured document which represents data. The XML document it self does not do anything; it is only a container for data. The benefits of the XML structure are many. It is a industry standard recommended by the W3C organization. Compared to other formats like CSV the XML format is quite self-explaining. The nodes in the document can be added, removed and named in any way you like. This is also a great advantage compared to other formats that often have a specific number of node types available. Because it is an industry standard it is widely used and a beneficial format to exchange data between platforms. [5]

2.4.1 Parsing

There are two approaches to use when parsing XML documents. Either the data is processed in the memory (DOM) or just scanned (SAX).

DOM This parser is the most common and popular one. It reads the entire XML document making it possible to access and change the data within the memory. Because the entire document is read into the memory random access is possible. Random access means that the nodes do not

have to be read in any specific order. A drawback of this method is that it consumes a lot of memory and can be hard to use when dealing with a large amount of data. [5]

SAX This parser is the other method to use when parsing XML documents. A SAX parser scans the whole document sequentially and raises events that then can be handled. Unlike a DOM parser a SAX parser is a read only application. Therefore it is not a good solution when the document has to be changed. The benefit of using SAX parsers is when handling a huge amount of data that can not be read into the memory. [5]

2.4.2 XSLT

XSLT is a stylesheet method of restructuring data. The language can be used in many different ways; for example transform one XML document into another, generate a HTML document from a XML document etc. When restructuring the information with XSLT often XPaths are used. XPaths are explained in Section 2.4.3. The main idea is that a stylesheet is created in XSLT format. The XSLT documents refers to the information in the XML document needed to restructure. This stylesheet is processed together with the source XML-document to generate the desirable output. Figure 2.4 shows an overview of the XSLT concept. [2]

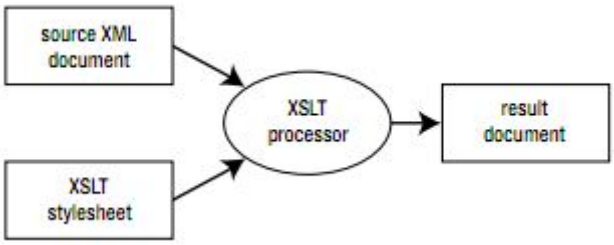


Figure 2.4: Description of XSLT processor.

2.4.3 XPath

XPath is a common method to access data within a XML document. The XPath looks for a defined pattern in the file and selects the nodes that fulfill the pattern. For example if the XPath looks like "*Data/Objects/*" the sub-nodes that are children to *Data/Objects* will be selected. When performing the selection of data logical conditions and functions can be added to filter the data. Logical conditions are added by putting brackets in the search string. The bracket can be placed either inside or in the end of the search string depending on which node to filter on. A XPath like *"Data/Objects[id='123']*" would select the sub-nodes of *Data/Objects* where id is equal to "123". Functions are applied to a specific search string. The expression *count("Data/Objects")* would return an integer value that corresponds to the number of sub-nodes in *Data/Object*. Both these methods can of course be combined in many different ways to get the desirable data. [7]

2.5 The manufacturing station (Y312H)

The manufacturing station in the simulation environment which is analyzed in this project is named Y312H. The station assembles the rear floor for a specific car. The rear floor consists of several sub-parts which are glued and welded together in different steps. To execute these steps, different fixtures, weld guns and grippers are used. In the cell there are a lot of manual work performed by an operator. These operations will not be covered in this project because the focus is on the automatic equipment. A graphical representation of the station can be seen in Figure 2.5.

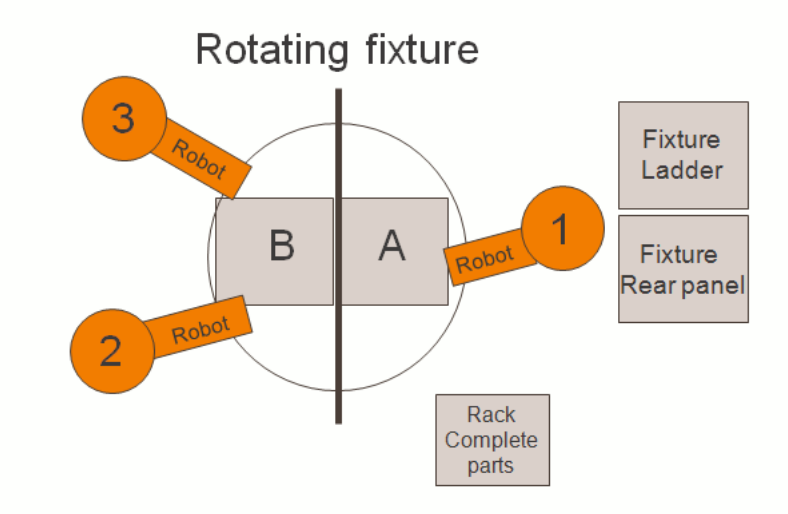


Figure 2.5: Simple layout of the Y312H cell.

2.5.1 Layout of the station

The station consist of a number of resources and sub-products to produce the rear floor.

Resources

The station Y312H consists of the following resources:

- A rotating fixture table with two sides where different sub-parts are glued and welded together in different steps.
- Three robots that perform all the operations in the station:
  - Robot 1 handles the gripping and the gluing operations.
  - Robot 2 and 3 performs the spot welding operations.
- Two manual fixtures that are loaded by an operator.
- Tool stands where the different grippers and weld guns are stationed when not used.
- Racks that contains sub-parts and one rack where the finished rear floor parts are placed after all performed operations.

Bill of Materials

To produce the rear floor, several sub-parts are used:

- Rear panel
- Cover lids
- Ladder frame
- Rear section

Reachability

The robots in the station has several reachability issues aspects to consider. First of all, Robot 1 must be able to reach a lot of places that are deployed too far away from each other for the robot to reach. Therefore, Robot 1 is placed on a rail that is able to move along a track. The following applies to the robots:

- Robot 1 must reach the fixture table, the tool stands linked to that robot, the manual fixtures and the racks.
- Robot 2 and 3 must reach the fixture table and the tool stands linked to those robots. [1]

2.5.2 Operations

Even though most of the operations in the study are event-based, it is still appropriate to sketch a sequential chart that shows which order the operations needs to be done in. Figure 2.6 shows a sequential chart of one product cycle where all resources are included. There are a number of operations omitted in the chart e.g. manual operations performed by an operator and tool changing activities. A full description of the robots and the fixture table's individual sequences can be seen in Appendix A.

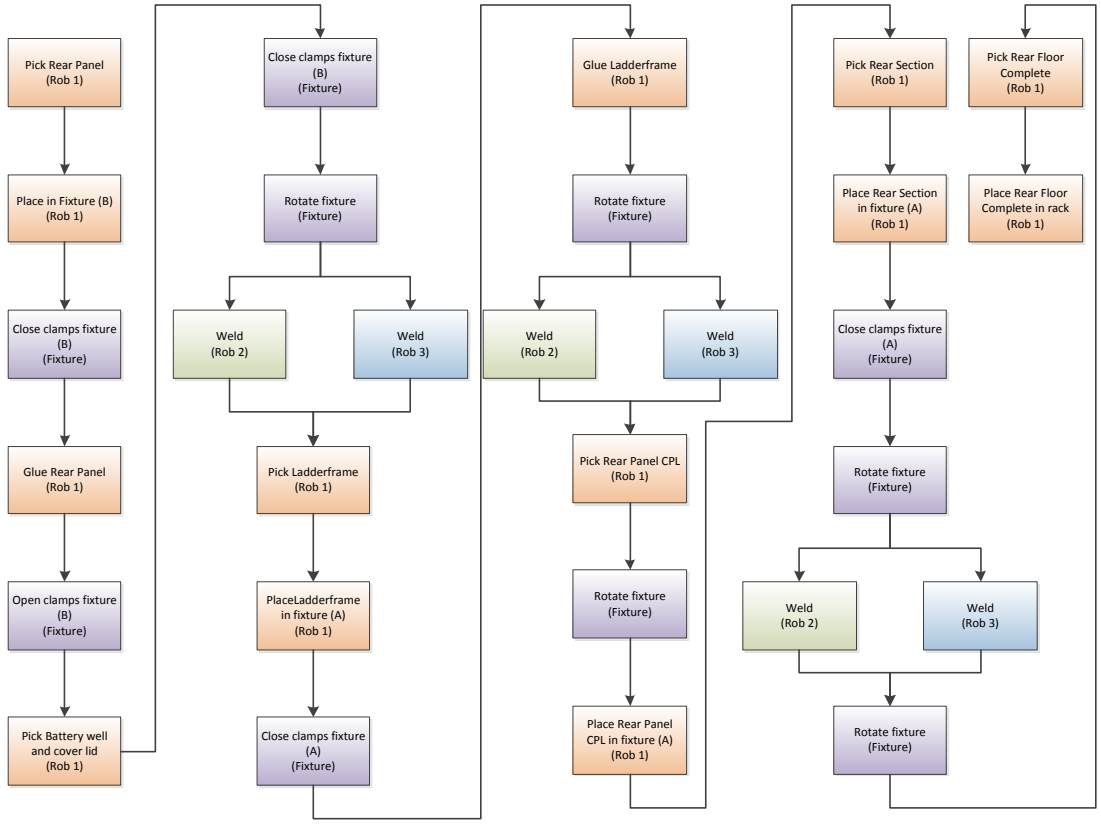


Figure 2.6: Sequential chart for one finished rear floor cycle.

Chapter 3

Issues to consider

Within the project there are a number of different approaches that need to be considered. The structure of this chapter is based on the aims for this thesis work. For every problem associated solutions is presented. All problems are presented in the decision matrix in Appendix B, where the chosen solution is highlighted. The chosen solutions will then be more thoroughly described in the following chapters.

3.1 Problem: Preparation of the simulation model

Early in the project instructions regarding the simulation model were set. The instructions are based on matters that needed to be done in order to get a good-looking simulation model and to be able to find and extract necessary information to visualize operations with associated logic. The aim for building the model was to complete it from a SE-phase to a fully functional discrete event model. How the chosen solutions regarding the simulation model was used will be more thoroughly described in Chapter 4.

3.1.1 Subproblem: Definition of kinematics

Defining kinematics of a simulation model can be done in several steps. Common for all kinematics is that it should be as much a like the reality as possible. One problem in the model was how to define the fixture. The fixture can either be defined as individual clamps or as a gripper. The intention of the gripper type is to apply it to the grippers of the robots. But since the functionality of the fixture is very much alike a robot gripper, it also applies to the fixture in this case.

If the fixture is defined as a number of clamps another action has to be added when the clamps are closed, a attach action. An attach action defines which part to be attached to the clamp (see Figure 3.2).

Defining the fixture as a gripper makes the attachment of the part to occur once the clamp is closed. A gripper has gripping entities and a list of parts to grip. The gripping entities are the parts of the clamp that should interact with the parts. The list of parts can be set to all parts, meaning that all parts touching the gripping entity when it is in closed state will be attached. [10]

As stated before the goal is to achieve a model as much a like the reality as possible. Therefore the choice was to model the fixture as a gripper tool although this is not the intention for this tool type.

3.1.2 Subproblem: Operation of sequences

Operations in a simulation model done in PS can be structured in several ways. As stated in Section 2.1 there are two main concepts of structuring the operation, time- and event-based. All operations are placed in a Gantt chart. When building the model in a time-based manner all operations are added after each other with transition restricting the sequence to continue until a condition is fulfilled (see Figure 3.1). This method of building the model restricts operations

to happen before the previous operations have completed its task. When applying the event-based approach the operations are placed in a flat structure where every operation has individual transitions restricting the operation to start before a specific condition is set. [4]

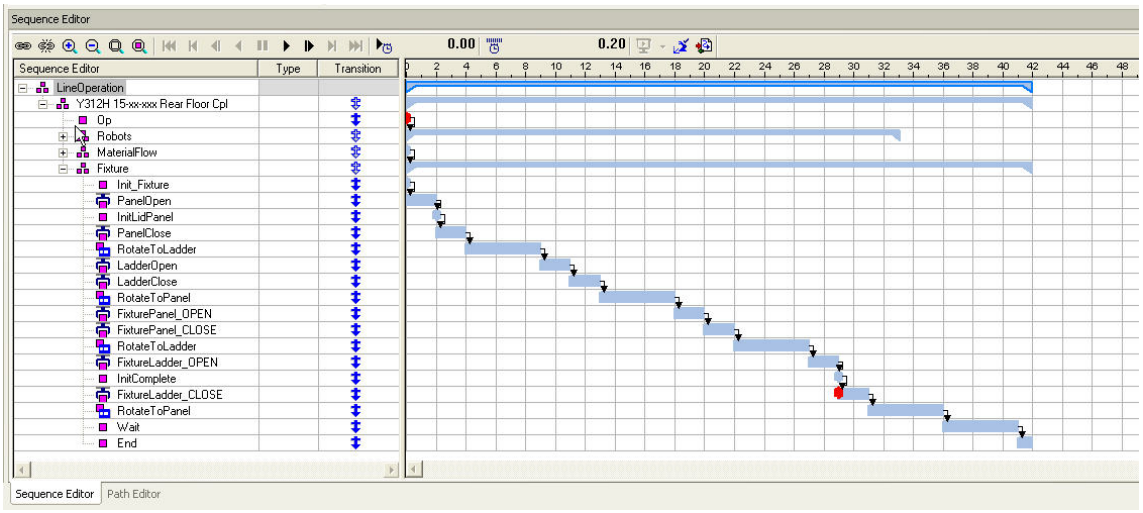


Figure 3.1: An example of sequenced operations due to a time-based approach.

To decide which approach to use information about the intention of the model must be known. In this project the intention is to make a visualization of the logical flow in the cell. This puts a demand that as much realistic logic as possible should be defined in the model. An aspect to consider when building a model in PS is the material flow. The material flow is described in Section 3.1.3. To have a continuous flow of material and to be able to decide when the material should enter the simulation it has to be included in a sequence that follows the complete sequence of the cell. Both the event-based and time-based concept has pros and cons. The time-based way of working is very advantageous when the project is in the early stages (the SE-phase). In the SE-phase the product is still being changed which also continuously changes the production line. Changes in the production line are more easily handled when working with a time-based simulation. The reason for this is that in a time-based simulation less logic has to be added to create a model. Instead the simulation runs by the specified order. The order is specified in a Gantt or a Pert chart which easily can be redrawn. On the other hand if the goal is to make the simulation as alike the reality as possible the event-based way of working is preferable. When working with events the model can be triggered on the same signals as the real cell is. This makes it possible to extract more information from the model and the possibility to test a PLC program. [11]

Since the examined robot cell is a complex cell with a lot of interaction between the robots and the rotating fixture, it became clear that an event-based approach must be used as a starting point. Thus, with a rather complex material flow with a lot of joining, the time-based approach was also used. A combination of the two approaches made up the simulation and the main reason for that is that the simulation became a lot easier when not trying to stick just to one approach.

3.1.3 Subproblem: Material flow

To know how to deal with the material flow it is necessary to know how different parts will interact with each other in the simulation model. Two different solutions was examined for this case; attach events for connecting parts to each other and merge parts with the help of IPAs.

The first approach, attach events, is about adding events to operations in the Gantt chart which makes a specified part to be connected to another part (see Figure 3.2). Thus, this will still make the two parts to be separated from each other in the Object Tree. The advantage with this approach is that parts easily can be disconnected from each other with a detach event. The disadvantage is most visible when creating a simulation of a model which shows more than one product cycle and when the cycles are partly merged. Then, with two instances of the same part,

only one of the instances will be active and have the ability to for example be gripped (see Figure 3.3a).

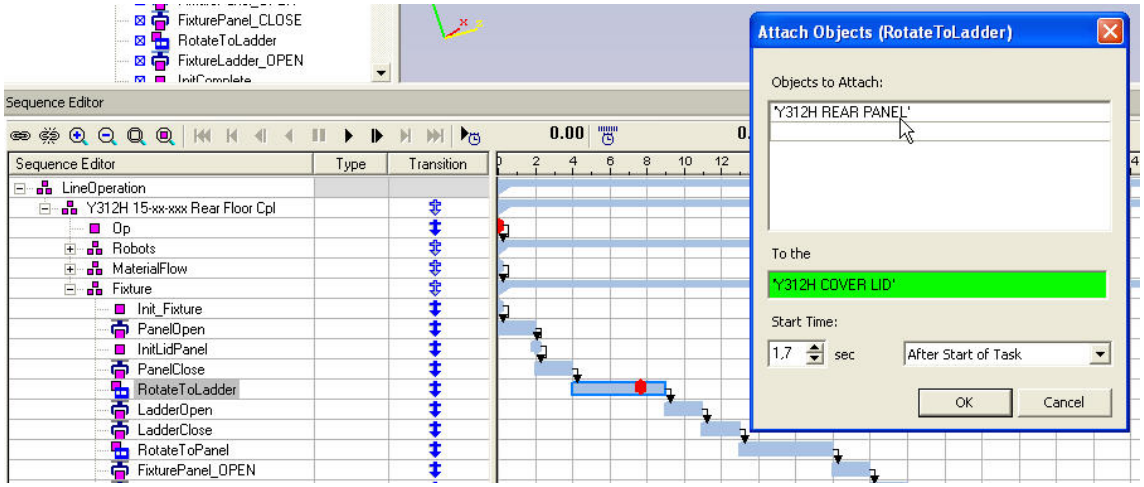
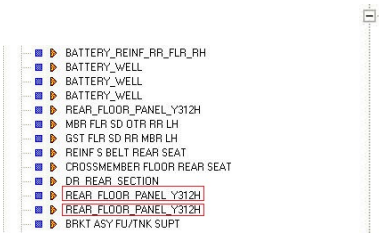
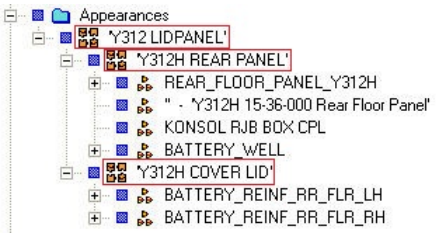


Figure 3.2: An attach event in the Sequence Editor.

The second approach, merge parts to an IPA, includes a bit of work in PD. Two or more parts must be connected in PD in order to form a higher level IPA that then contains the old parts. The old parts will then disappear in the object tree and be replaced by the new IPA (see Figure 3.3b) and the new IPA will then contain the old parts. The pros and cons for this approach are roughly the opposite of the other approach. Since the examined simulation model assembles several parts but never disassembles any parts the second approach preferable.



(a) Two instances of the same part in the Object Tree.



(b) A higher level IPA containing the old parts.

Figure 3.3: Parts in the appearance tree.

3.1.4 Subproblem: Insertion of logic

The logical expressions for operations can be inserted in several places in the simulation model. Three different solutions have been examined; logic blocks (see Section 2.1.1), modules (see Section 4.5.2) and logic in the Gantt chart. If there are logical expressions in the model that often is used in other models as well, for example zone allocation or robot start signals, logic blocks are good to use. However, if the logical expression is peculiar for the given model, then logic blocks often are too complex to use and it will be more work than is achieved. In that case, modules can be used. Both modules and logic blocks are good logical expression sources to use when building the model due to an event-based approach. Though an event-based approach is to prefer, some parts of the model is often built due to a time-based approach. The time-based parts of the logic are best placed in the Gantt chart.

Regarding where preferably one wants to insert logic in a simulation model it is different from case to case. Inserting logic in modules is rather simple and most events can be simulated with a couple of if-else statements. Unfortunately it tends to be lots of if-else statements and it is difficult

for an external user to understand how the logical expressions are supposed to work. The Gantt chart is also simple to use as a logic source, but since the aim in many cases is to keep the model as event-based as possible the amount of logic inserted here should be kept to a minimum. [4]

For this project, decisions have been made to use a combination of these three solutions. This has been done since all these solutions have advantages and it is very complex to stick just to one logic source. To be consistent, it was decided to use modules only for distinguish different cases in WaitCase operations from each other, the Gantt chart when operations were connected according to a time-based approach and the logic blocks for standard station logic. Another important point for this problem was to make sure that logic was inserted in the model so all important logical expressions for the visualization could be exported from PS. By making it clear where to insert different the different parts of the logic a working solution for extraction can be guaranteed.

3.2 Problem: Extraction of information from Process Simulate

A project in PS contains a massive load of information and the extraction of operations and logical expressions is therefore crucial for the later work with the visualization to run as smooth as possible. In an early phase of the project several ways to solve the extraction were discovered. During the project, four possible solutions have been discussed and examined to find out how the extraction should be done in order to find all relevant information. A few requirements were set on the extraction:

- The size of the extracted information should be kept to a minimum.
- It should be as easy as possible to extract the correct information from PS.
- It must be possible to, from the extracted data, be able to automatically generate a visualization of a logical flow of operations for all robots, but also for other selected resources (fixtures etc.) in the model.

Since operations and logical expressions have been partly extracted from PS with the help of an API in a previous made thesis work, it was natural to examine that solution. The application built in that thesis work was tested with the simulation model created in this project to see if the application managed to extract the wanted information as supposed. Unfortunately, the application was built on a previous version of PS, which means that it could not handle all types of operations and nodes that the simulation model of this project contained. Discussions with one of the developers of this application were held and ideas to construct a new application with help of an API came up. However, since the aim of this project not was to look into the usage of an API, it was considered that the time frame for such an implementation would be too narrow. [8]

eM-Planner is a tool developed by Siemens that is used for modeling and analysis of manufacturing processes and lines, just as PD/PS. There are also possibilities to do queries on the database which the simulation model is built on. The advantage of deducing information this way is that the graphical interface in eM-Planner does it easy for the user to traverse the database and find the information without knowing so much about how a database works. The disadvantage is that the extraction of information only could be presented in an Excel-formatted file, which is not a big issue in itself, thus still a delimitation in the extraction of information.

The last solution examined is the export of an XML file directly from PS. PS has the possibility to export information largely about anything in the simulation model. For example, data regarding physical and virtual resources can be extracted and also information about operations and logic can be found through the XML export. The advantage with XML export through PS is that it is possible to find a lot of information about the model just by traversing through the navigation tree. The risk using this approach is the amount of information that can be extracted if not holding information about exactly what is needed to be extracted.

In this project, information about operations and which resources that are performing respectively operation needs to be extracted, but also data regarding signals and sensors to build up information regarding the logic in the simulation model. Since a solution including working

against an API not was realistic given the time frame of the thesis project, emphasis was on the latter solutions. Concerning information extraction with eM-Planner, it must be possible to do queries directly on the database instead of using the graphical interface in eM-Planner. If that is not possible, it will be harder to find a way to create auto-generated flows since the user manually must extract the correct information from inside eM-Planner. Also, according to simulation experts at VCC, eM-Planner is a tool that is being phased out from the activity at VCC as PD is on its way to be developed to get the same functionality as eM-Planner. The XML export solution was thoroughly discussed, since the risk with a huge amount of extracted data could easily lead to misunderstanding of the data. Decisions were taken to keep the extracted data to a minimum and examine possibilities to reduce the exported information as far as it goes.

Summed up, the only realistic and valid solution to extract data from the simulation model in order to visualize auto-generated logical flows is the XML export. The XML export will be more described in Chapter 5.

3.3 Problem: Restructuring of the information

Once the information has been extracted from the simulation model it has to be restructured in some way to fit the demands of the visualization program. In Section 3.2 the method to extract data was chosen to be XML. XML is a good format because it can quite easy be restructured and most programs can import it, in their specific format. To restructure a XML file there are several ways to do it. Two main concepts have been examined; the stylesheet method XSLT or to process the data within a own developed application.

When using the XSLT solution a stylesheet has to be created. The stylesheet contains information about how the data should be restructured. Combining this document and the XML data creates an output document in the specified structure. The second approach is to process the data within a application. When processing the data inside a application the data is stored inside objects in the application and then outputted in the desired format.

The data that is extracted from PS is largely in a flat structure. It also contains different XML documents with different structure. This makes it quite hard to use XSLT for restructuring of the information. Of course it is possible, but the gain is quite small. The desired output to the visualization software demands a lot of information which is not included in the XML data outputted from PS. Therefore the easiest way to handle this is to generate templates containing this information. With a C# application an output file can quite easily be generated when having all the data placed in objects within the application and the templates as separate files. A advantage of a application solution is also that it easier can be implemented into softwares like Sequence Planner to get a overall solution to the visualization problem. Therefore the approach with the C# solution was chosen. Since the C# solution consists most about programming code, this has not been thoroughly described in the report, since this not is an direct focus of the thesis, but only a way of reaching the focus. Though, it can be read in Chapter 6 how the thoughts behind the application are.

3.4 Problem: Visualization

As it is today, the sequence of operations is shown in an item list and the logic for a station is just defined in PS. There is no good way to connect the operations with the logic in a good visualized way. The first question to ask is in fact how the visualization should be made. Visualization is a very wide expression and can be done in a lot of ways. Since the aim is to visualize a manufacturing process with several different operations and conditions, a decision were made to put up the process as flow charts. This decision was made together with the supervisor for this thesis and no deeper investigation regarding visualization was made because VCC was satisfied with the proposed visualization method.

The second question to ask is how the flow chart should look like and what information it will contain. The base for the flow chart could be set in two different ways:

- Resource-based flow chart, which shows flow charts divided in resources, for instance one flow chart for robot 1, one for robot 2 etc.

- Operations-based flow chart, which shows flow charts divided in cells or stations, for instance one flow chart for cell 1, one for cell 2 etc.

The two ways have both pros and cons. A resource-based flow chart is beneficial when there is an intention of focusing on a specific resource to see exactly which operations the resource performs. It can also be good to use if there is a need of evaluating the utilization of a resource. On the other hand, it could be rather non-orderly if the intention is to see the complete cycle of a station. This is the great benefit of a operations-based view. To get a complete understanding of a station or a cell, this view can be advantageous because it is possible to follow the whole sequence of a station cycle. However, it can be difficult to keep track of what happens when there are a lot of resources and operations involved. After discussions with the supervisor for the project and staff working with PLC communication for fixtures and robots, it was decided that a resource-based flow chart was the best way of visualizing the operations.

Preferably the visualization should be easy to monitor for a wide spectrum of people, therefore the level of details must vary depending on whom the visualization is presented to. For example, staff that work with PLC signals will probably have greater benefit of conditions connected to the handshaking between PLC and the operations.

Throughout the thesis project, Sequence Planner was not functional in visualizing the logical flows from the simulation model in PS. Though, during the last month of the project major improvements by the department of Signals and Systems have been done to Sequence Planner. It is in the time of writing able to import information from PS regarding operations and logic and visualize them in an approved way. A major advantage with the visualization in Sequence Planner is that the operation blocks and the transitions are visualized clear from the beginning. There is no need to drag around blocks to make the flow chart look good as Sequence Planner is built up so operation blocks will not collide with each other. The disadvantage with Sequence Planner as a visualization tool is that transitions do not fall between operations in the visualized picture, but as a pre- or postcondition in the operation block (see Figure 2.2b). Regarding to the authors, it is better to have transition conditions between operations to make the visualization easier to overview.

The intention with Automation Designer regarding visualization was to get a better overview of the sequences of operations than PS can present. Since it was concluded that there are no good way to import extracted information regarding operations and logic in Automation Designer, focus has been taken away from doing this and instead emphasis has been on producing a thorough specification for Siemens regarding VCC´s preferences on next versions of Automation Designer.

Since neither Sequence Planner nor Automation Designer initially were as functional as wanted in visualizing the extracted data from PS, decisions were taken to try other softwares for visualization. Since a flow chart was the way the visualization should be structured as, the number of softwares available was reduced. Visio is a software developed by Microsoft that uses vector graphics to create and present diagrams of different types, so the opportunities to visualize flow charts in Visio is great. The advantage of Visio is that the vision of the visualization was made in Visio, so it was already concluded that such flow charts would be possible to do. Visio has also the opportunities to read files which structure is similar to a XML structured file which makes it a lot easier to restructure the extracted information into a structure Visio is able to read. Thus, a disadvantage with Visio is that the XML structured file has a very complex appearance with a lot of code that needs to be screened out to understand which parts of the code that are crucial to deal with to get a general implementation of the imported data.

In an early phase of this project, Visio was not even in prospect as an option for visualization. However, it was concluded that Visio were as functional as intended and gave a clear picture of the logical flow, so decisions were taken to select Visio as the main track and to work proactive with Automation Designer and Sequence Planner to hopefully give good input to the developers so the softwares can evolve in the right direction. The chosen visualization method can be read about in Chapter 6.

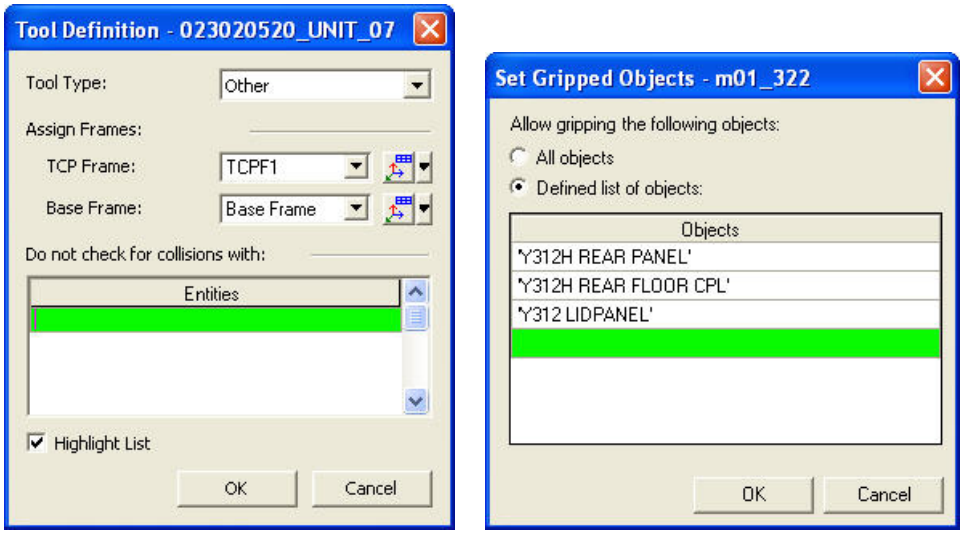
Chapter 4

Building the model

To fully understand which information that is wanted and possible to extract from PS, a station (see Section 2.5) was examined. The station had been preparatory worked by a simulation engineer, for example all welding points were already defined. Since the scope of the thesis concerns just the logic and the sequencing in a station, focus was to define everything that relates to this. For example, collisions between entities when simulating the model have not been fully taken into consideration, though it is important that the entities do not collide too much so it can be presentable in an eventual demonstration of the model.

4.1 Definition of kinematics

According to knowledge and information available, decisions were made to start defining kinematics. The kinematics for several entities (e.g. robots and the track) was already predefined from the import. For every fixture and gripper, the kinematics needed to be defined. The fixture has two entities, clamps and pins. Both these had to be given two different poses. The clamps got OPEN and CLOSE, while the pins got RETRACT and EXTEND. This is needed when invoking signals linked to the resources with the mentioned poses. All clamps were also defined as grippers, which facilitates the process of implementing a correct material flow (see Section 4.4) and a more realistic behavior. A gripper can be defined to grip several entities and an OLP command can easy be established for a gripper to grip or release any entities in the model (see Figure 4.1b). If the clamps of the fixture were defined as "Other" (see Figure 4.1b), then attach and detach events in the Gantt chart must be used which is an unnecessarily difficult solution. The kinematics was implemented in the structure tree according to VCC standards. [4]



(a) Clamps defined as "Other". (b) Clamps defined as grippers.

Figure 4.1: Different methods of defining the clamps.

4.2 Implementation of operations for different resources

All welding points, glue spots and a few pick-and-place locations were predefined by a simulation engineer, so the job was to add operations for the robots including via locations to avoid most collisions, pick-and-place operations and tool change operations (see Section 2.1.1). One tool change module were defined for each tool and was constructed according to VCC working methods document to make them work correctly. Thus the aim of the project was to extract information from the model, the robot operations was built according to an event-based concept. This way is also more connected to reality, where robot operations in most cases are triggered on signals. All robot operations were placed separate without any connection between. The connection between the robot operations will instead be done through OLP commands.

Operations for the rotating fixture in the model (rotating and clamp operations) were also defined. These operations were decided to be sequenced in the Gantt chart, mostly because it is an easier way to build the simulation model regarding the material flow (see Section 3.1.3). There was no attempt to construct the fixture operations non-sequenced, however there are doubtless possibilities to define the operations that way given the few types of operations available for the fixture in the model for this project. Device and gripper operations were used to build up the fixture sequence. Decisions were also taken to include the material flow to the fixture sequence (see Section 4.4). This was done in order to get the material flow to work correctly and to add the right material at the right time in the simulation model.

4.3 Adding OLP commands

OLP commands are internal orders for a robot that with advantage are used in a LineSimulation study, where it is common that the robots not has a time-based approach, but instead decoupled operations. In other words, every robot operation runs simultaneously and is triggered by different signals.

To get the most out of the extraction of information, the intention was to implement a few predefined VCC OLP commands, even if it was not completely necessary. The reason why it was done was particularly to give VCC a more general template of the visualization where the most common used commands were taken into account. Especially the WaitCase command (see Section 2.1.1), which is used for branching between different choices depending on what DirectionOutFrom (see Section 2.1.1) is set to. In the model used for this thesis, WaitCase is not necessary because of the straight sequence that means that operations must always happen in a specific order.

4.3.1 Schedulers

A scheduler should be assigned to a robot and consists often of several operations and OLP commands. It is usually the scheduler that is triggered by a defined robot program and the other robot operations are then triggered by the scheduler. The main idea is that it should be one scheduler per robot, but when using cases as in WaitCase there also needs to be one scheduler per case, which then are controlled by the robot´s main scheduler (see Figure 4.2). To make a scheduler work as it is supposed to work, every transition leading to any operation that is included in the scheduler must be set to FALSE in the Gantt chart (see Figure 4.3).

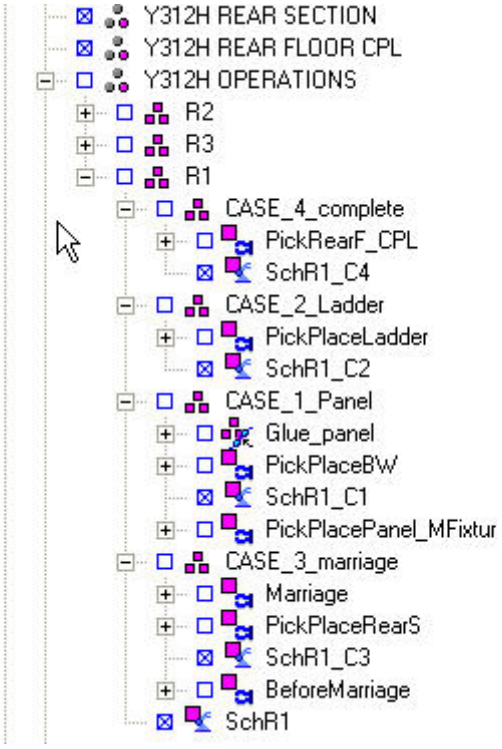


Figure 4.2: Several schedulers for one robot controlled by a main scheduler.

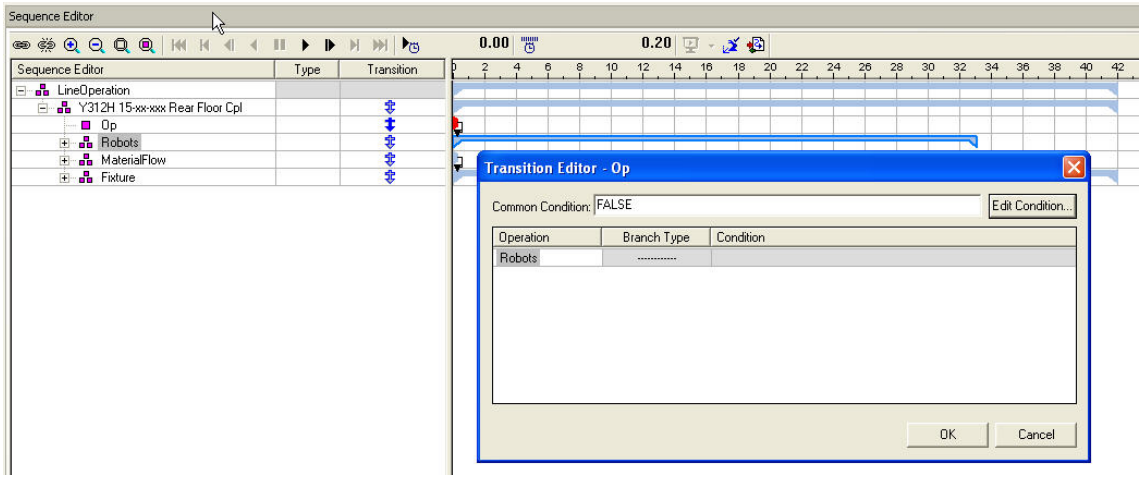


Figure 4.3: The sequence editor with conditions leading to scheduled operations set to FALSE.

Every robot in the model has been assigned one main scheduler with a WaitCase command. In this case, it means that all robot operations are part of case statements, where every case contains

a certain number of operations. Therefore all robots have underlying schedulers (one for each case) ,which all are connected to the main scheduler through the WaitCase command. To control the main scheduler to select a specific case, modules have been created which sets an integer value depending on what conditions are met. How the modules was constructed with associated if-else statements can be seen in Section 4.5.2.

To create a scheduler in PS, the segmentizer function must be used for the compound to be scheduled. The segmentizer is used to create correct OLP commands according to VCC´s own implemented signals and commands. After segmentizing, a scheduler will be created with the content of the compound and its associated OLP commands. [4]

4.4 Material flow

In a LineSimulation study, it can be tricky to make the material flow work in a correct way. Compared to a RobCad study, there exist no parts in a LineSimulation study. Instead, appearances are used. An appearance is an instance of a part that appears in the simulation if the part is connected to an operation. The connection to operations is done in PD. As mentioned in Section 2.5, Y312H produces a rear floor that consists of four major parts that in turn consists of smaller sub-parts. To better manage all these sub-parts, IPAs are connected to the operations instead. An IPA is a collection of parts which all together make up a larger part. For example, if a car is an IPA, a door and a window can be parts in collection of that IPA. IPAs are created by adding stations in PD. Every station become an IPA with the corresponding name.

The approach in this model has been to merge IPAs in PD to then get a better appearance implementation in PS. The model in PS has been created as a LineProcess, which means that the model is built on line level with underlying stations. This setting can be changed through an admin console, which is an application for administrative usage of PD/PS. One station has been created per insertion that are added into the model. In this way, one IPA is created for every station in the model. Figure4.4 shows a Pert chart of an IPA, “Y312H REAR PANEL”. The Pert chart contains three different types of objects. (1) is the station which will appear as a IPA. (2) and (3) are the parts that will be added to the IPA. In this case there are two different types of parts. (2) is individual parts while (3) is a previous assembly made in another line.

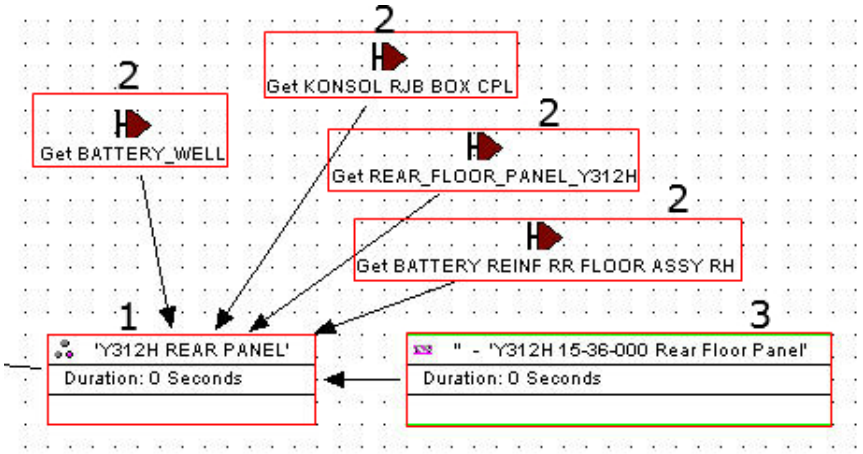


Figure 4.4: Adding parts to a station node gives an IPA.

The idea is that the model should run an infinite number of cycles with new material added to the simulation model to be assembled, which means that material needs to be continuous added to the model. To generate material non-sim operations with associated IPAs are used (see Figure 4.5). In this and other models, it is important that material is generated at the right time. The problem if the material is sourced in at wrong time is that there can not exist two instances of an IPA at the same time. Having many instances will make the newest instance active and the others inactive. That means, among other things, that grippers not will be fully functional as grippers

because they can only grip the active instances in the model. To solve this, IPAs has to be sourced into a higher level of IPA containing the old IPA. With the previous mentioned example with the car in mind, so if the windows and the doors are individual IPAs they first have to be sourced into a "complete-car-IPA" before another window or door can be added to the model.

To merge IPAs, PD is also used. Two or more stations are then connected to another station, which means that an IPA will be created containing the old IPAs. Figure 4.6 shows a station (1) which will contain many IPAs. For example it contains the IPA generated in Figure 4.4. Once the higher level IPA is generated into the model the sub-IPA will be a part of it (see Figure 3.3b). Worth noting is that to make the IPAs appear in the Assembly Tree in PD, a new Assembly Tree must be generated after changes have been done. This is performed by choosing *Tools -> In-Process Assembly -> Generate Assembly Tree*.

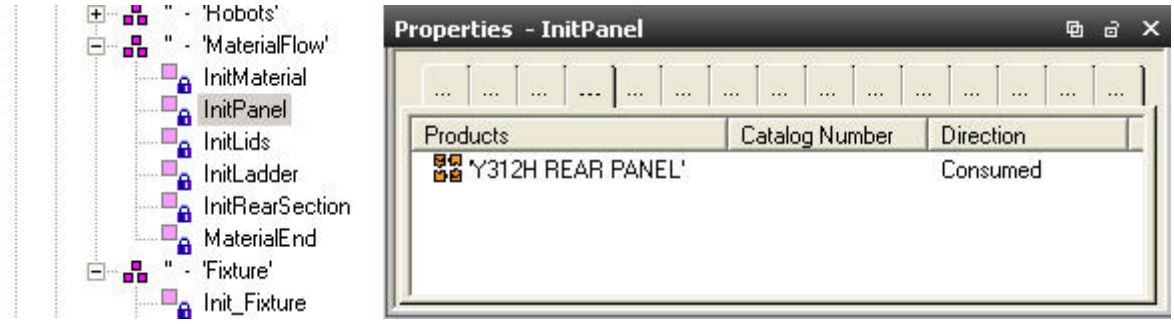


Figure 4.5: A non-sim operation with associated IPAs.

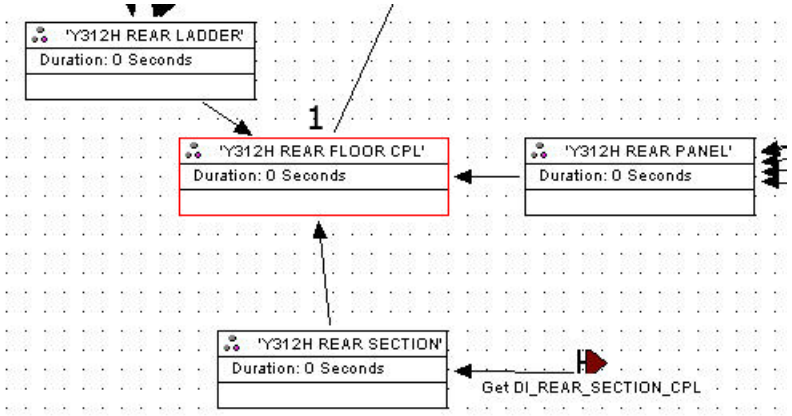


Figure 4.6: Several stations connected to another station to form a higher level IPA.

4.5 Logic insertion in the simulation model

As mentioned, all robot operations were placed separately in the Gantt chart with no connections between. Though, the fixture and the initialization of the different parts of material were connected in the Gantt chart. This was done in order to initiate the material at the right time in the simulation model, since the current version of PS lacks in function regarding material treatment. Therefore logic needed to be inserted in several locations in PS due to the combination of time-based and event-based approaches.

4.5.1 Logic blocks

As the intention was to incorporate as many commonly used functions in PS as possible when building the model, several standard logic blocks are used. The fixture is seen as two station where the robots must allocate a station to be permitted to work and therefore the logic block

Allocate_Zone (see Section 2.1.1), which also except zone allocation contains station allocation, is used for this. In combination with the standard logic block WeldCompleted_ML, which keeps track of whether the robots have finished their work or not, the fixture is prohibited to rotate when the robots are working. To make the robots functional and initiated, the block RobotStartBlock_Rx, where x is the selected robot, is used.

The standard logic blocks are made available by importing them from a server at VCC where they are located. After imported, the logic blocks can be found at the node EngineeringResourceLibrary in the Navigation Tree. To structure the model in the right way, the logic blocks and any sensor signals imported should be placed in a resource compound under the station resource node to avoid getting shortcuts of the logic blocks spread out too much in the project structure (see Figure 4.7). When importing logic blocks, standard signals are in most cases included. It was discovered after a few attempts with the standard signals that they did not work properly every time. In that case, it might be a good idea to delete the included signals and create new signals with the same name to be sure that the imported logic blocks will be fully functional.

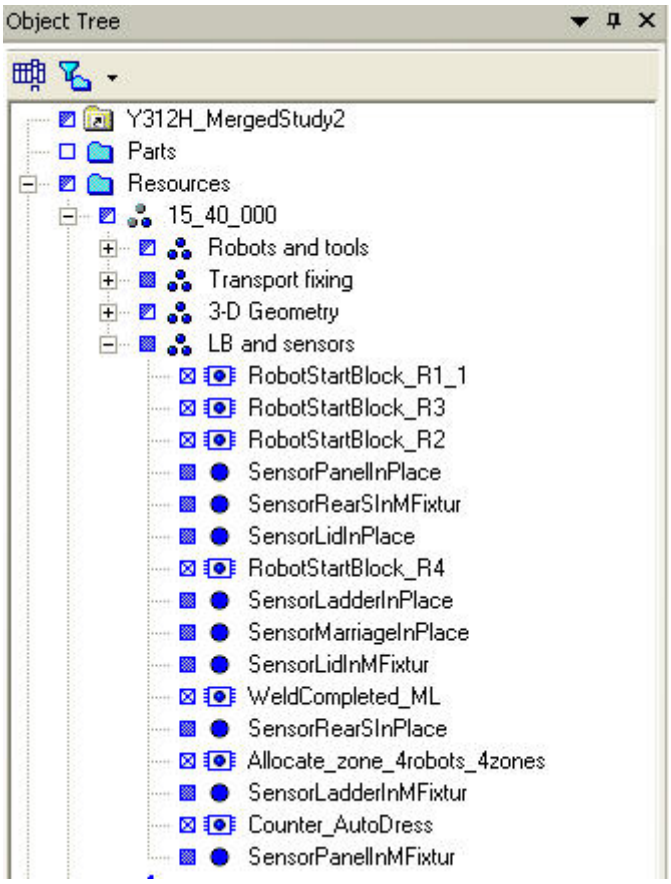


Figure 4.7: Logic blocks and sensors gathered in one resource compound.

4.5.2 Modules

To avoid too nested if-else statements, the Modules Viewer was used to a small extent. It was only used to simplify the use of WaitCase, with one if-statement for every case. Every if-statement then contains modules that sets the robot input signal DirectionIn_rx, where x is the selected robot, to a given integer value that decides which case the robot should execute. The integer values represents the different cases associated with the robots. There is also a module for each robot that resets the value of DirectionIn_rx after each performed case (see Figure 4.8).

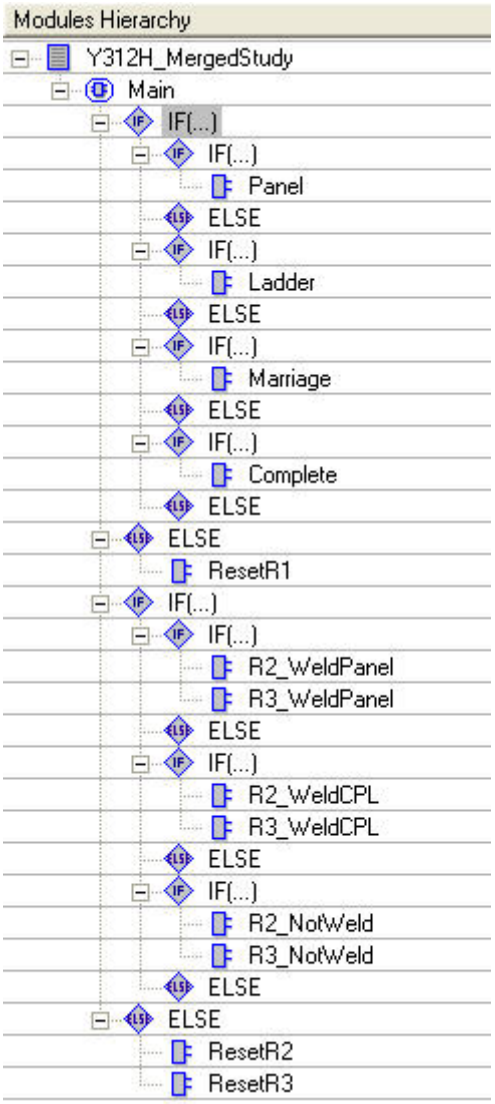


Figure 4.8: The modules hierarchy showing how the different modules are used.

4.5.3 Transitions in the Gantt chart

In order to make the rotating fixture table work properly, transitions must be defined in the Gantt chart due to the sequencing of operations for that resource. The transitions are mostly based on sensor signals, thus a number of sensors must be triggered for a fixture operation to occur. The usage of sensors in the model is kept to a minimum and are placed to make the simulation easier, not to mimic the reality. In the real robot cell, every single clamp often have a sensor. To make the sensor signals functional in the simulation model, two main issues need to be fulfilled. First, under *Tools -> Options -> Motion*, the checkbox *"Ignore signals in robotic simulation"* must be unchecked. Secondly, as mentioned in Section 2.1.2, all sensor signals must be specified somewhere as a transition in the Sequence Editor in order to work properly. Since the model is supposed to run continuously, the conditions for the fixture took some reflections to make it work correct. It is for instance not the same scenario the first and the second product cycle, since the cycles are merged into each other and different sensor signals are set depending on the cycle.

Chapter 5

Extraction of information

In this chapter the XML export solution will be more thoroughly examined and it will be explained where the data is positioned in PS.

5.1 Extract from Process Simulate

To get the correct information from PS it is important to export the correct files. PS stores the information on different places depending on which information it is. For the visualization in this project four files are required to get all the information. All files from PS are extracted by choosing *File -> Project Management -> Export selected eBOP to file...* (see Figure5.1).

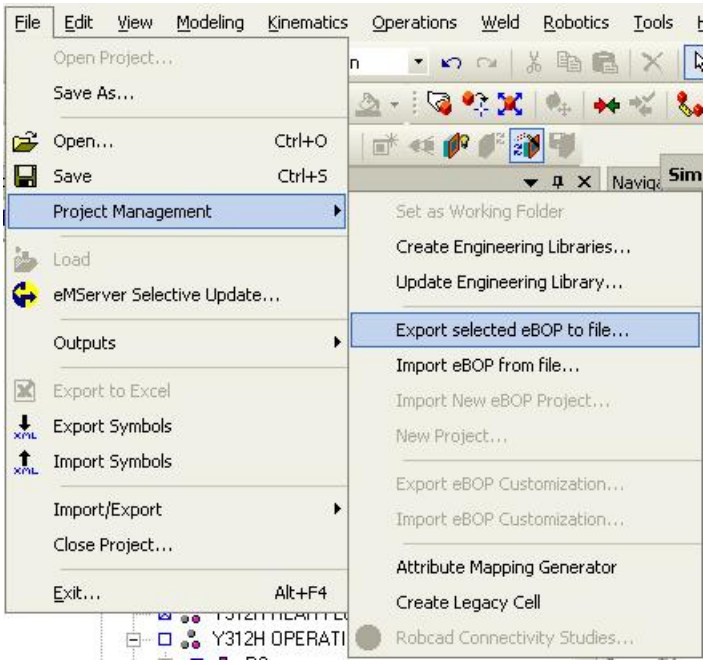


Figure 5.1: Extraction of XML files from PS.

The four files that needs to be extracted are:

- Operations
- Resources
- PLCProgram
- Simulation Panel

Operations The operation file contains information of all operations in the simulation model; their names, under which compound are they placed etc. To extract this information the node in Figure 5.2 is marked and extracted to a file. When extracting the XML files with operations it is important that only the line/station that is supposed to be visualized is marked. If for example the top node is chosen the XML file will become to large for the program to handle. This was one of the major problems regarding XML-wise extraction.

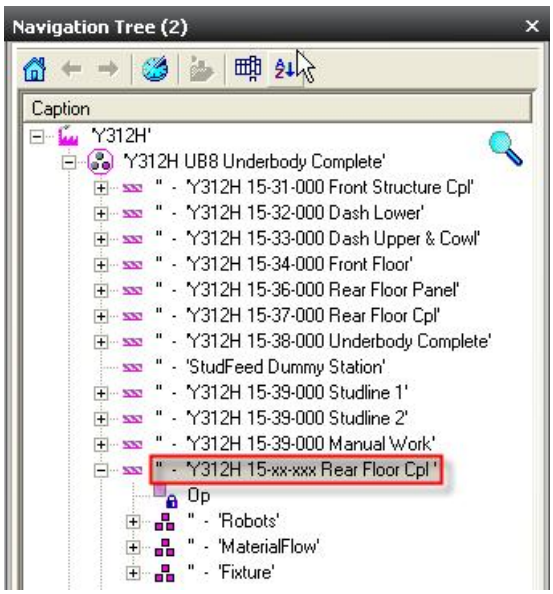


Figure 5.2: The node to extract to achieve information regarding operations.

Resources The resource contains information about all resources in the model; robots, fixtures etc. Besides the wanted information this file also contains a lot of information about all 3D representations of the resources. This information is not needed in this exact case but is also hard to sort out. To extract the resources the node in Figure 5.3 should be extracted.

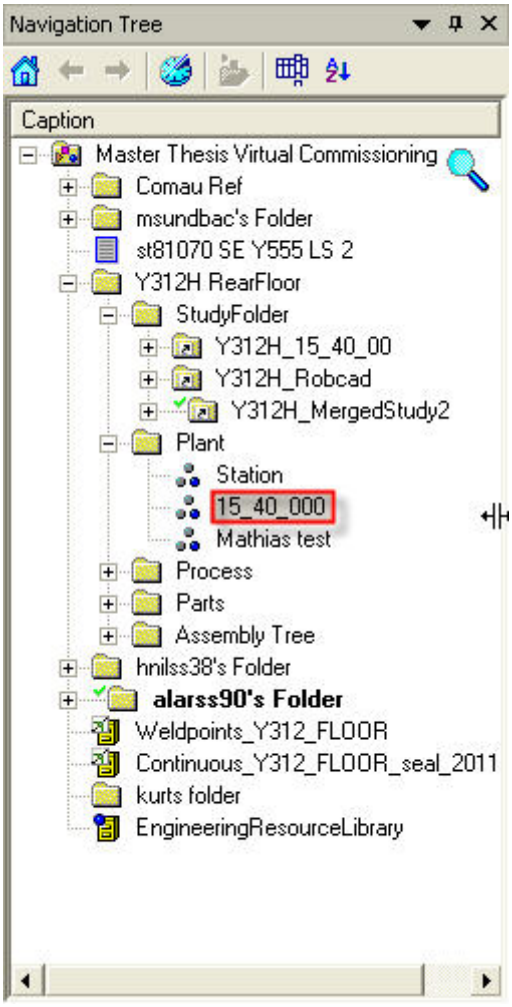


Figure 5.3: The node to extract to achieve information regarding resources.

PLCProgram The PLCProgram file contains information about all the logic in the cell. For example the logical transitions in the Gantt chart and also the expressions inside the Modules Viewer. Unlike the extraction of the operations and transitions the extracted file containing the logic does not contain a fully correct XML structure. Some parts of it have to be decoded to be able to read it. Table 5.1 shows a list of all decodings that needs to be done.

Table 5.1: Decoding needed to be done on the PLC program file

Value in the file	Replaced by
##	“
&gt;	>
&lt;	<

The PLCProgram file is extracted by selecting the node in Figure 5.4.

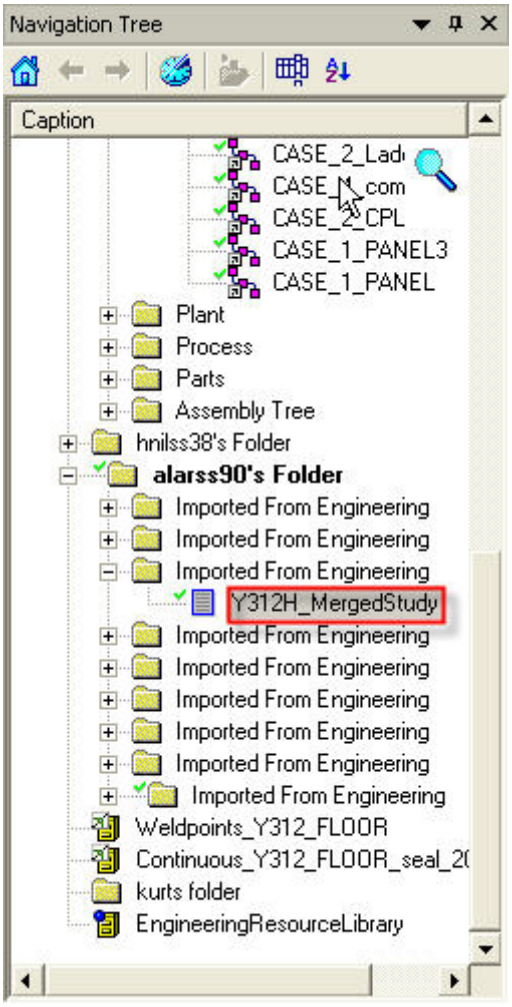


Figure 5.4: The node to extract to achieve information regarding PLCProgram.

Simulation Panel The last extraction is the simulation panel. This file contains static information about variables that need to be set for the simulation to be able to run. For example, PS does not contain any static variable declaration for integers so if the simulation needs a variable with an integer value to run this has to be assigned in the simulation panel. Extraction of this is simply done by copying the saved .spss-file from the simulation panel (see Figure 5.5) and then change the extension to .xml.

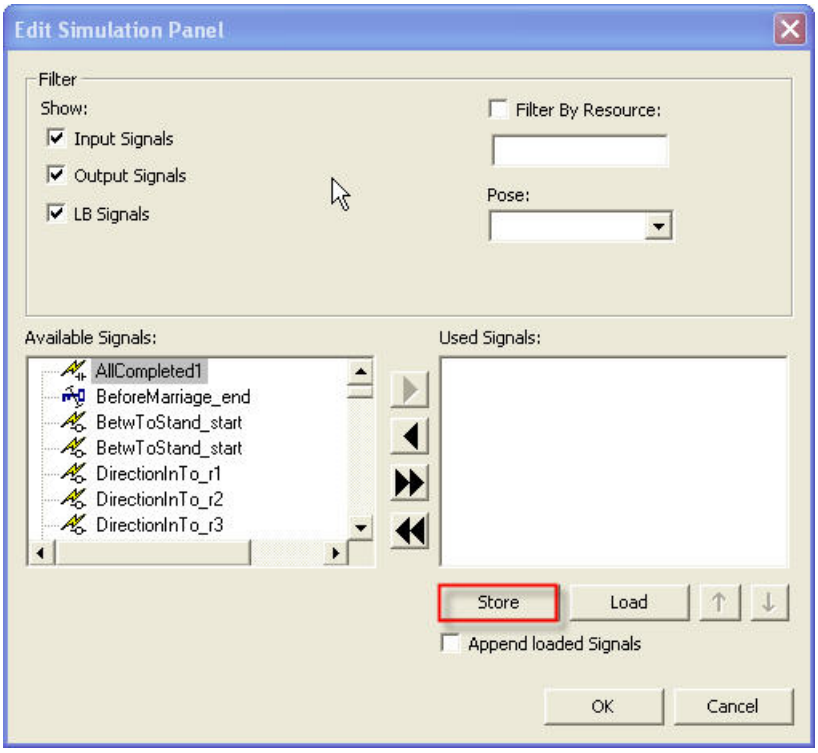


Figure 5.5: Information regarding the simulation panel can be extracted by saving the .spss-file.

5.2 Extracted XML documents

The understanding of the extracted XML documents is a time-consuming work. It is important to understand the overall structure to be able to extract the exact information needed. All objects in the XML files have an external ID which is created inside PS and applies to everything, from a small bolt to a robot operation. There is no way to separate the different objects from each other by their external ID; instead they are classified in groups depending on what type of object it is. The groups can be found as sub-nodes to the top-node (see Figure5.6). Every object in the cell gets the name of the group it belongs to. Therefore the XML file contains many nodes with the same name as its individual objects. Common for all nodes that are some sort of operation is that operation nodes always have the same two sub-nodes, *outputFlows* and *inputFlows*. These nodes contain information about which operation that are performed before (*inputFlows*) and which operation that is performed after (*outputFlows*). These nodes can be used to find the logical flow for everything that exists in the Gantt chart.

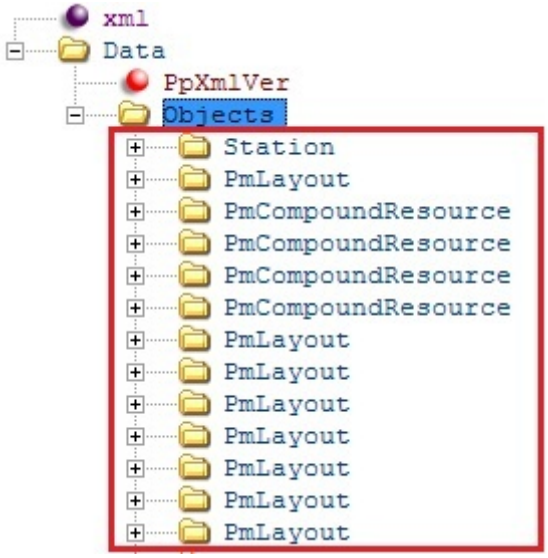


Figure 5.6: Tree structure of one extracted XML file.

Different information can be found in the four different extracted documents. It is important to know which document and node to access for the correct information. The most important nodes and a description of their most important information will be described in this chapter.

5.2.1 Operations

PmCompound This group contains all operation compounds in the model. The group has a list named *children* that contains all external ID:s to the sub-operations that are gathered in the compound.

PmGenericRoboticOperation This group contains all schedulers. The important node in this group is the *simulationInfo* node. In this node the information inside the OLP command window can be found. Figure 5.7 shows an example of a typical *simulationInfo* tag.

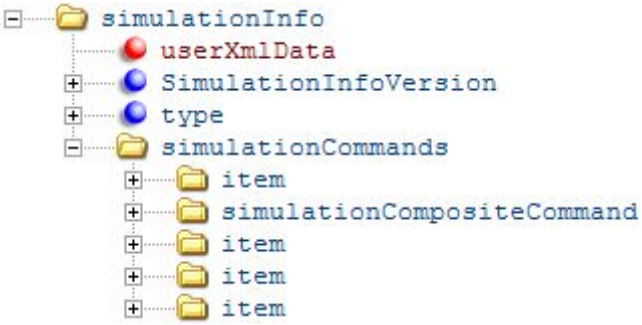


Figure 5.7: An example of a typical *simulationInfo* tag.

The *simulationInfo* tag contains two different sub-nodes, item and *simulationCompositeCommand*. These nodes are ordered in the order that they appear in the OLP-command. The sub-node *item* is largely straightforward. It contains operations like WaitSignal, Weld operations etc. (see Figure 5.8). It only contains two parts, an external ID and a string describing the operation.

Figure 5.8: Content of the node *item*.

The sub-node *simulationCompositeCommand* is more complex and is used for the WaitCases. To generate the WaitCase OLP commands it contains several sub-nodes named *simulationParameter*. Every *simulationParameter* node contains information about the WaitCase operation. The number of nodes depends on the number of cases in the operation (see Figure 5.9). The first two *simulationParameter* nodes declare that it is a WaitCase operation and which lock signal that is connected to it. The following nodes declare the cases.

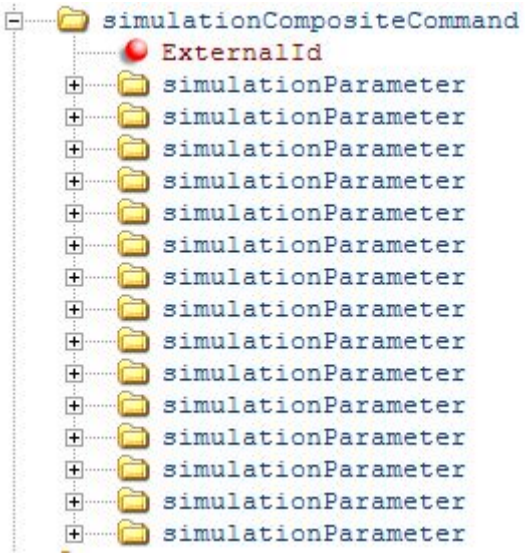


Figure 5.9: Structure of *simulationCompositeCommand*.

Every *simulationParameter* node look the same having two parameters, *simulationParamType* and one other parameter that varies depending on what is described (see Figure 5.10). The *simulationParamType* declares what data that is contained in the second parameter. All different data types used in this project can be found in Table 5.2.

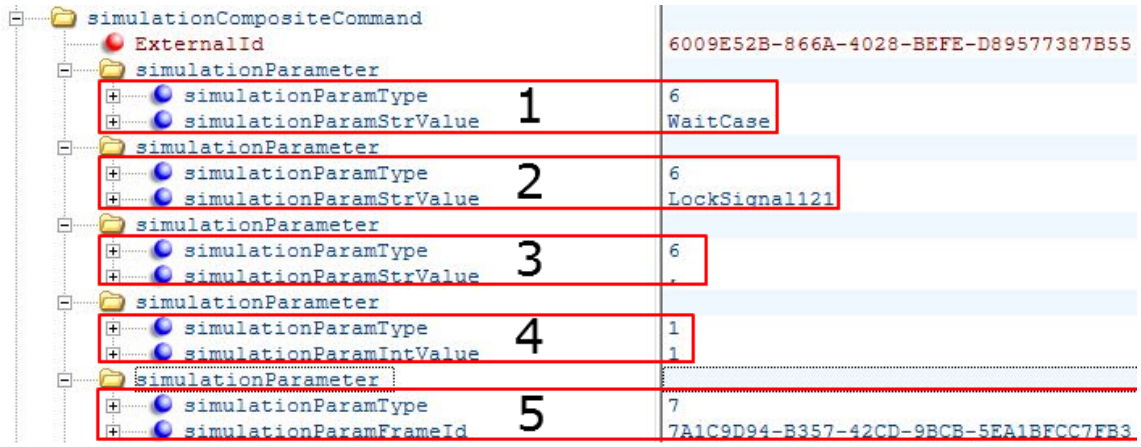


Figure 5.10: Structure of a WaitCase.

Table 5.2: The different parameter types used in this project.

Value	Type
1	Integer
6	String
7	Reference to an external ID

As stated before the first two nodes are declaration of the WaitCase operation. In Figure 5.10 five nodes are marked. These five nodes all together declares the WaitCase and one of the cases. The list beneath corresponds to the five marked nodes.

1. WaitCase declaration.
2. Declares which lock signal to be used. In this case the value of LockSignal121 (121) is communicated from the robot when it asks for which case to choose.
3. Declares a comma sign to the string in the OLP-command. Has nothing to do with the actual case.
4. Declares that if an arbitrary integer value is returned to the robot it should perform the case assigned to the given integer value.
5. Declares a reference to the case that should be performed if the integer value in point 4 is returned.

5.2.2 Resources

PmToolInstance These nodes contains all instances of resources. For example a specific type of robot can appear many times in the simulation. The robot basic information is defined in one group and the unique instances of the robot is defined in *PmToolInstance*. For example if there are two robots of the same type in a model the definition will only appear once but the resource instance will appear twice. Under this node the resource gets a name and an external ID. Which resource that is the base for the instance is defined in the node prototype (see Figure 5.11).

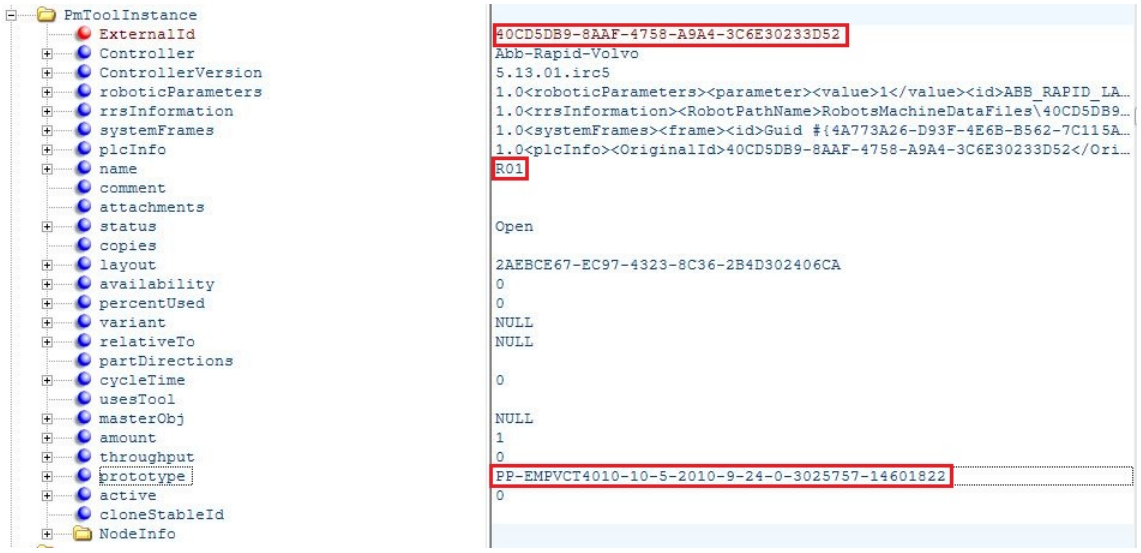
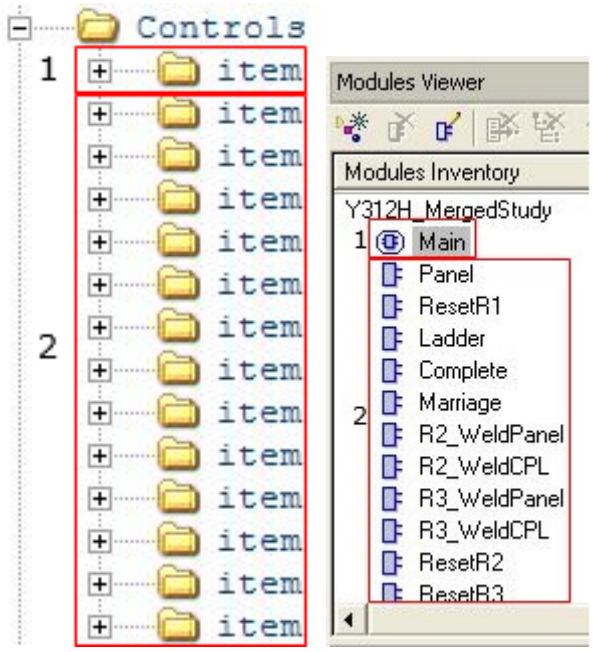


Figure 5.11: Structure of the node *PmToolInstance*.

5.2.3 PLCProgram

As stated before this extraction contains all information regarding the logic in the cell. The logical information can be found in the sub-node *plcInfo/plcInfo*. This sub-node contains a number of important nodes.

Controls In this node the Modules Viewer is defined. It contains a number of sub-nodes called *item* (see Figure 5.12a).



(a) The nodes shown in a XML document.

(b) Modules Viewer in PS.

Figure 5.12: Structure of the Controls node.

The item marked with (1) in Figure 5.12a and Figure 5.12b is the Main module. It defines all the if-statements with their conditions and what they should perform. To be able to do anything inside the first node all module blocks has to be defined. This is done in all items marked with (2) in Figure 5.12a and Figure 5.12b. An item in the second group (see Figure 5.12a) is either a module block in Figure 5.12b or a declaration made directly in the main module. Every item contains a source signal (1) and a target signal (2) (see Figure 5.13). Both the source and the target signal contain a reference in form of a external ID to the involved signals. The source signal can either be one or more signals depending on the expression and the target signal is only one signal. If the item is a module there is only one source signal for every target signal. The signals have to be of the same type. For example in the WaitCase the source signal is an integer signal with a defined value and when the module is called the target signal, which is a input to the robot, is set to the value. If the item is a declaration written in the Main module the source signal can contain many signals. When having many signals Boolean expressions is added to know when to set the target signal to true or false. One example is if you have one signal that is to be set if either signal A or signal B is true the source signal is "signal A OR signal B".

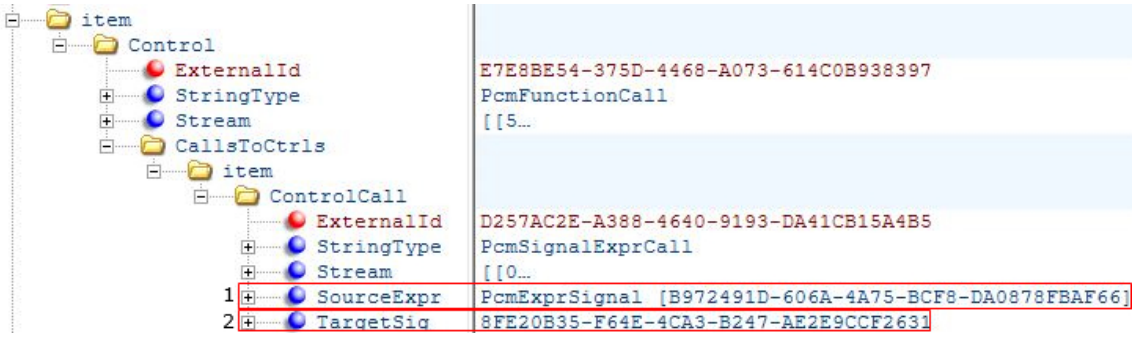
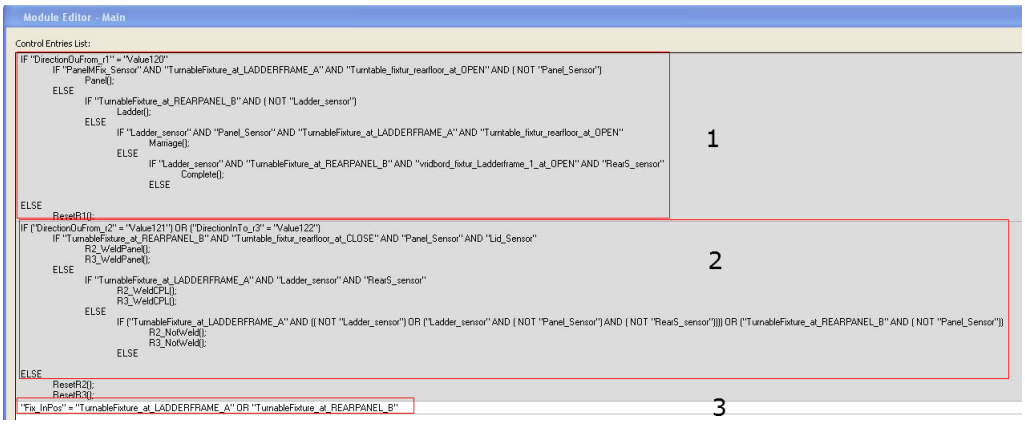


Figure 5.13: Definition of a modules block.

When all the modules blocks are defined they can be called from the Main module. The convocation of the modules are done inside the if-statement (see Figure 5.14b). Each if-statement corresponds to a node in the XML file (see Figure 5.14a). The third *item* node is a declaration of a signal made directly in the Main module. In the XML structure every if-statement is iteratively built, which means that if the logic contains a nested if-statement there will be a sub node inside the item-node looking exactly the same as the top one.

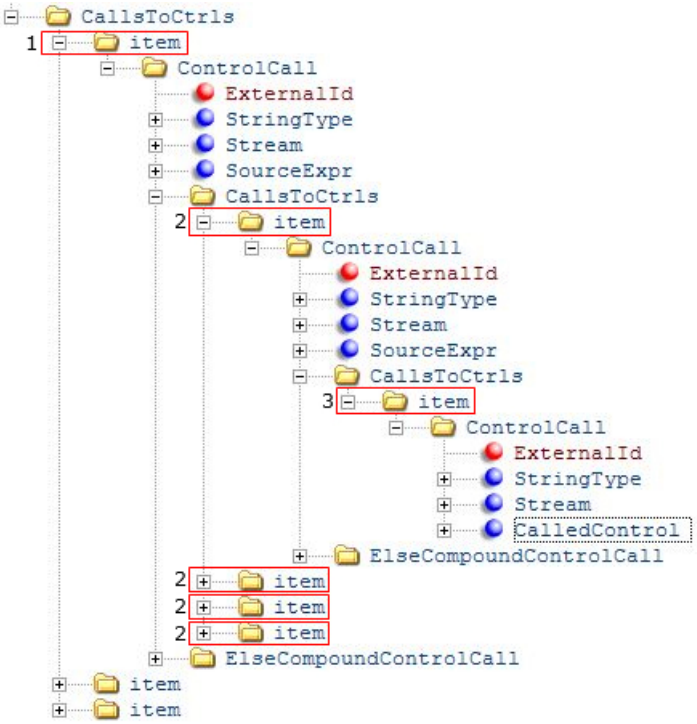
(a) The Main module shown in a XML document.



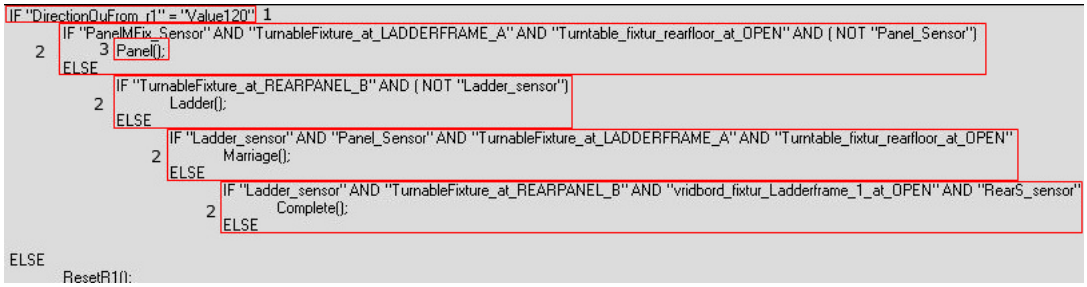
(b) The Main module shown in the Modules Viewer in PS.

Figure 5.14: Structure of first *item* node (Main).

To be able to understand the structure the first if-statement is expanded in Figure 5.15. The numbers in Figure 5.15a corresponds to an element in Figure 5.15b. Looking at Figure 5.15b, it can be seen that the first if-statement contains four sub statements. Of those the first one is expanded to see the underlying structure. This is where the actual call to the module is performed. The function call is marked with (3). Just as with the if-statements the function calls are built exactly the same. It contains a call to the external ID of the module that should be called. A module with the corresponding external ID is found in the defined modules blocks (see Figure 5.13 and the action is performed. A note is that the else-statement is placed as a sub node inside the if-statement it belongs to.



(a) If-statements shown in a XML document.



(b) If-statements in the Modules Viewer in PS.

Figure 5.15: Structure of if-statements.

Chapter 6

Building the application

To be able to visualize the sequence an application was built. The aim with the application was to fill the gap between PS and the chosen visualization tool Visio. Visio can read a file format named .vdx. A vdx-file is a XML structured file formatted in a special way. Therefore the application should be able to read XML data exported from PS and convert it into a XML structured file in form of a vdx-file.

In order to read the extracted documents the program first have to restructure the data according to Table 5.1. After this decoding has been done the documents contain a valid XML structure. To ease the use of XPaths when extracting information the documents are merged into one document that is saved in the location of the extracted files. When the program has the merged and decoded file it can extract the wanted information.

6.1 Data extraction

The data extraction is done in many different steps to get the desirable information. General for all data extraction is that XPaths are used to get the data. The XPaths is used to find the correct nodes for the data. All XPaths used in this project can be found in Appendix C. When extracting the data only the raw data is extracted. By raw data means that if there are many programs that runs the same scheduler the schedule will only be extracted once. To handle the extracted data the application creates objects that contain the wanted information. These objects can then be used when generating the output.

6.2 Crucial settings in the simulation model for the application to work properly

The extraction of the four XML documents in Section 5.1 are needed to visualize the logical flow of all robot programs and compounds with operations. In order for the application to work properly, some crucial requirements are set on on the built simulation model:

- The Modules Viewer must only contain modules that set DirectionIn for the different robots to a given value. More exactly, the if-statements must only deal with WaitCase commands.
- The robot programs must be done with a main scheduler for the program and then one scheduler for every case in the program.
- Compounds that should be visualized must contain operations that are fully or partly sequenced.
- Compounds with only non-sim operations will not be visualized in a correct way.

6.3 Generate output

As stated in before the output file created is a XML structured file with the extension .vdx. To get as logically flows as possible the output from the application is a number of files. The application creates one file for every resource (Rob1, Rob2 etc.). Note that if a robot has many robot programs these are also placed in different files. The files are named according to *[resource]_Program_[ProgramNr]*.

Every vdx-file looks largely the same with a lot of nodes looking the same. The only nodes added/removed are the ones containing information about the operations and transitions. Therefore a couple of templates were used when creating the final output:

VisioTemplateFile - This file contains the main structure of a vdx-file.

CodeForCreatingProcess - This file contains information for creating one operation.

CodeForCreatingTextInProcess - This file contains information for adding text to a operation.

CodeForGeneratingDynamicConnector - This file contains information for creating one transition.

CodeForDefiningConnection - This file is needed for the transition to connect. It defines between which operations the transition should be connected.

Using templates like these makes the generation of the vdx-file largely straightforward, once the data is extracted. For example when creating the operations the templates *CodeForCreatingProcess* and *CodeForCreatingTextInProcess* are just merged, changed and inserted into the *VisioTemplateFile* as many times as it exists operations.

Chapter 7

Result

A simulation model was upgraded from early SE-phase to OLP level using PS and PD. The upgrade consisted of implementing operations, logic and other necessary functions in order to make the simulation model work properly. When the model was found complete, information from the simulation model was extracted. This was to be able to visualize logical flows containing operations and their associated logic. Information regarding all operations in the model, all transition conditions that shows which signals are needed to be set in order for an operation to happen and all resources in the model was extracted as XML files.

To get any benefit out of the extracted XML files, they were thoroughly analyzed to understand how the different tags in the files were connected. By defining XPaths, the XML structure could be traversed easier to understand which parts of the XML files that were most important. The important XPaths found in this project can be found in Appendix C. After all important information was traced, an application was programmed in C# which uses the defined XPaths to generate a output which can be read in both SP and Visio. The application has a graphical interface (see Figure 7.1) where the user can chose which files to analyze, which logical sequences to visualize and where to save the files. There are instructions in the menu which facilitate the use of the application. These can be seen in Appendix D. The visualization of the generated output was done in Visio and SP. Both with flow charts that consist of operation blocks and transition lines to easy show the logical flow. Every compound and robot program is visualized in separate files. The visualized flows can be seen in Appendix A.

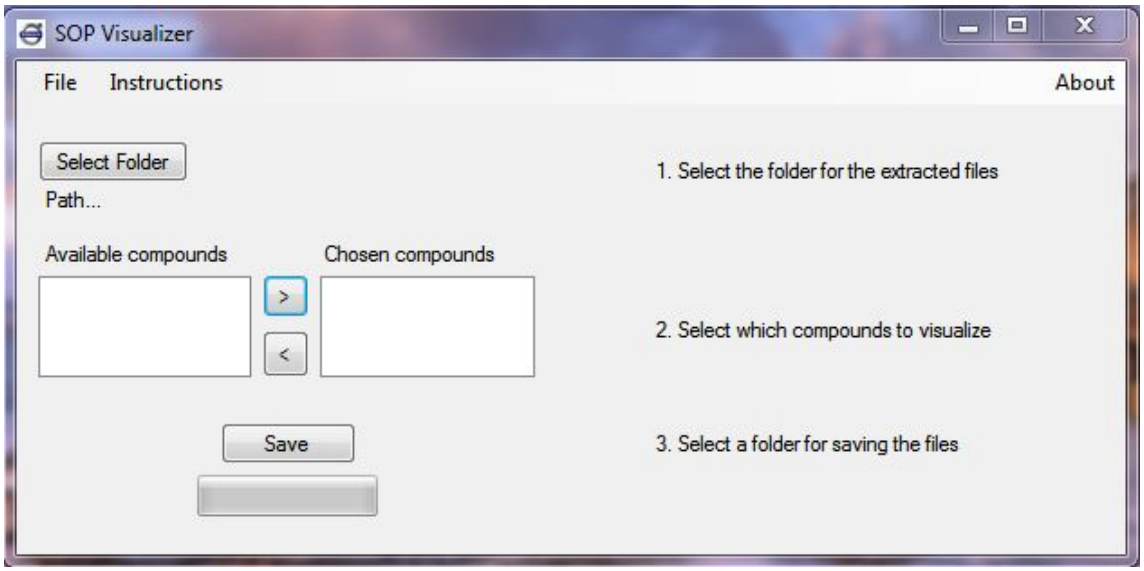


Figure 7.1: The graphical interface of the application.

Since the application automatically creates flow charts of the logical sequences, it can replace

manual created flow charts as it shows basically the same information. VCC does this manually today and their flow charts are called Marguerites. Since this is done automatically in this application, a lot of manually work is saved. The communication between simulation engineers and PLC engineers can also be improved by the auto-generated flow charts since visual information is easier to communicate through. By take this into consideration; more efficient engineering can be achieved.

A lot of knowledge regarding LineSimulation mode in PS has been achieved throughout the thesis work and the intention is that VCC should be able to use that knowledge in their future work in robot simulation. Focus during building the simulation model has naturally been to get the model ready to OLP level as smooth and fast as possible. Though, several attempts of handling issues in the simulation model a bit different than VCC ordinarily do have been done in order to improve methods in simulation that VCC use, for example the material flow.

Chapter 8

Discussion

During this thesis project, it has been discovered that documentation regarding how information in PS is structured and where to find it is inadequate. Thus, VCC have not lacked in documenting this kind of information, but have not be able to access all necessary information needed to get a full understanding of the structure. Discussions have been made with staff at VCC and also staff working at VCC production site in Gent. Since Siemens develop PS and PD they have lot of information about this issue. This information has, as far as the authors know, not reached VCC to a large extent. To be able to develop new skills in this area, a better understanding of the structure in the data handling of PS must be achieved.

The solution presented in this thesis work is by all means meeting the visualization requirements stated initially. Though, the authors are aware of that this solution must be developed much more for VCC to use it to a full extent. Since the application built is detached from PS and the visualization tool, a change in any of these softwares could make the application unusable. To make a visualization of the sequence of operations more sustainable, a more static connection between the simulation software and the visualization software would be preferable to make sure that the functionality is maintained. A better connection between PS and the developed application would maybe also make it possible to add many interesting features. One could for example be to implement a in real time visualization of the sequence, meaning that the current status of the robots could be seen in the visualization by having the current operation highlighted. Sequence planning feels like a natural matter that Siemens could implement in the Siemens softwares to optimize the usage of robotic simulation tools like PD/PS.

The exported format from the application can also be discussed. As this application is built the exported visualization is static and does not have any reverse connection to the model, meaning that changes in the visualization can not be imported to the simulation model. As it is now there are no standard for which format information regarding logic should be in when exchanged between different parts of the production. Having a standardized format that has a reverse connection would be the best solution. This would make it possible for data exchange earlier in the project and a more simultaneous way of working could be implemented. As it is now one department performs its tasks and then sends it to the next department. A simultaneous development would drastically decrease the development time for a production line.

To get the best solution there should be much more exchange of information between VCC, Siemens and Chalmers. Having all these three working together on a unified solution would bring a lot of different knowledge into the result. VCC have of course the knowledge about parameters that are desirable to achieve short development time. Siemens has the knowledge about how to develop the software in a efficient manner. Chalmers can contribute with a lot of theoretical knowledge about how to optimize the sequence of operation. If all this would be put together into one common solution the result would be significantly better than everyone creating their own solution.

Chapter 9

Future recommendations

The future recommendations for continuing on this project covers three main parts:

- Process Simulate
- The built application
- Sequence Planner
- Automation Designer

To facilitate the use of the built application it would be good to have a function in PS that extracts the necessary XML files automatically, without the need to find the information self. The four XML files could then be automatically extracted, placed in a folder, and then be ready for use. A recommendation for VCC is to develop such a function in PS to remove the need of traversing through the nodes in the project.

The application should be improved to be able to generate the logical flow directly in .pdf format or a picture file without the need of Visio. Some functionality regarding visualizing parts of sequences could also be preferably. This is useful when parts of a robot program are to be sent to another department. The visualization should be sent with a document describing all the signals in the visualization. In order to get a standard document that easily could be understood a template should be generated for this. Important when creating the template is that a thorough discussion is done with the other departments to see what problems they have when trying to understand the visualization.

The department of Signals and Systems needs to continue their work with Sequence Planner to get a better visualization functionality than it is today. The authors think that transition conditions should be visualized as a line between the operations blocks instead of including it in the operation block. Another recommendation is that Chalmers keep up the cooperation with VCC regarding connecting Sequence Planner with PS to remove the need for workarounds. XML files could then be directly imported from Sequence Planner to PS.

Finally, people at Siemens involved in the development of Automation Designer have been informed by the authors what functionalities that are needed in Automation Designer in order to make it work with exported XML documents from PS. A recommendation for VCC is to give Siemens continuously feedback so they can develop new functions regarding the import of operations and logic from PS.

Bibliography

[1] Sigun Edbäck. Y312h timeplan. PowerPoint presentation, 2010.

[2] James Robert Gardner. *XSLT and XPATH: a guide to XML transformations.* Prentice Hall, 2002.

[3] Mikael Andersson Erik Helander. Automatic generation of plc programs using automation designer. Master's thesis, Chalmers University of Technology, 2010.

[4] Magnus Jivefors. *Volvo Cars working methods.* Volvo Cars Corporation, 2.1 edition, 10 2010.

[5] Bipin Joshi. *Beginning xml with C# 2008 From Novice to Professional.* Apress, 2008.

[6] Patrik Magnusson. Interview, May 2011.

[7] Jim Melton. *Querying XML: XQuery, XPath, and SQL/XML in context.* The Morgan Kaufmann series in data management systems. Morgan Kaufmann, 2006.

[8] Fredrik Westman Christoffer Modig. Olp verification - linking eventbased simulations in process simulate with supremica to perform verification of sequences of operations. Master's thesis, Chalmers University of Technology, 2008.

[9] Tord Nordin. *Robot Programming Specification for ABB S4C+/IRC5 and Comau C4G Controllers.* Volvo Cars Corporation, 3.3 edition.

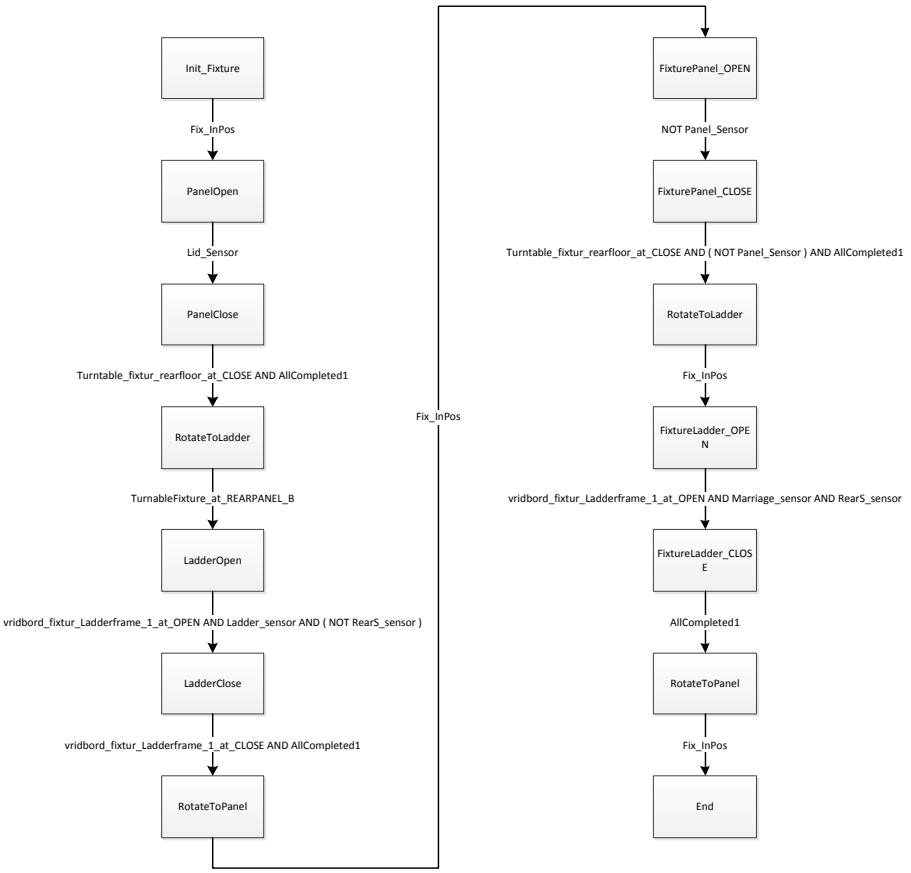
[10] Samir Delali Marcus Persson. Development of an event based robotic simulation implemented in process simulate. Master's thesis, Chalmers University of Technology, 2009.

[11] Tecnomatix. *Process Simulate CEE and Robotics Basics Student Guide.* Siemens, 8.2 edition, August 2008.

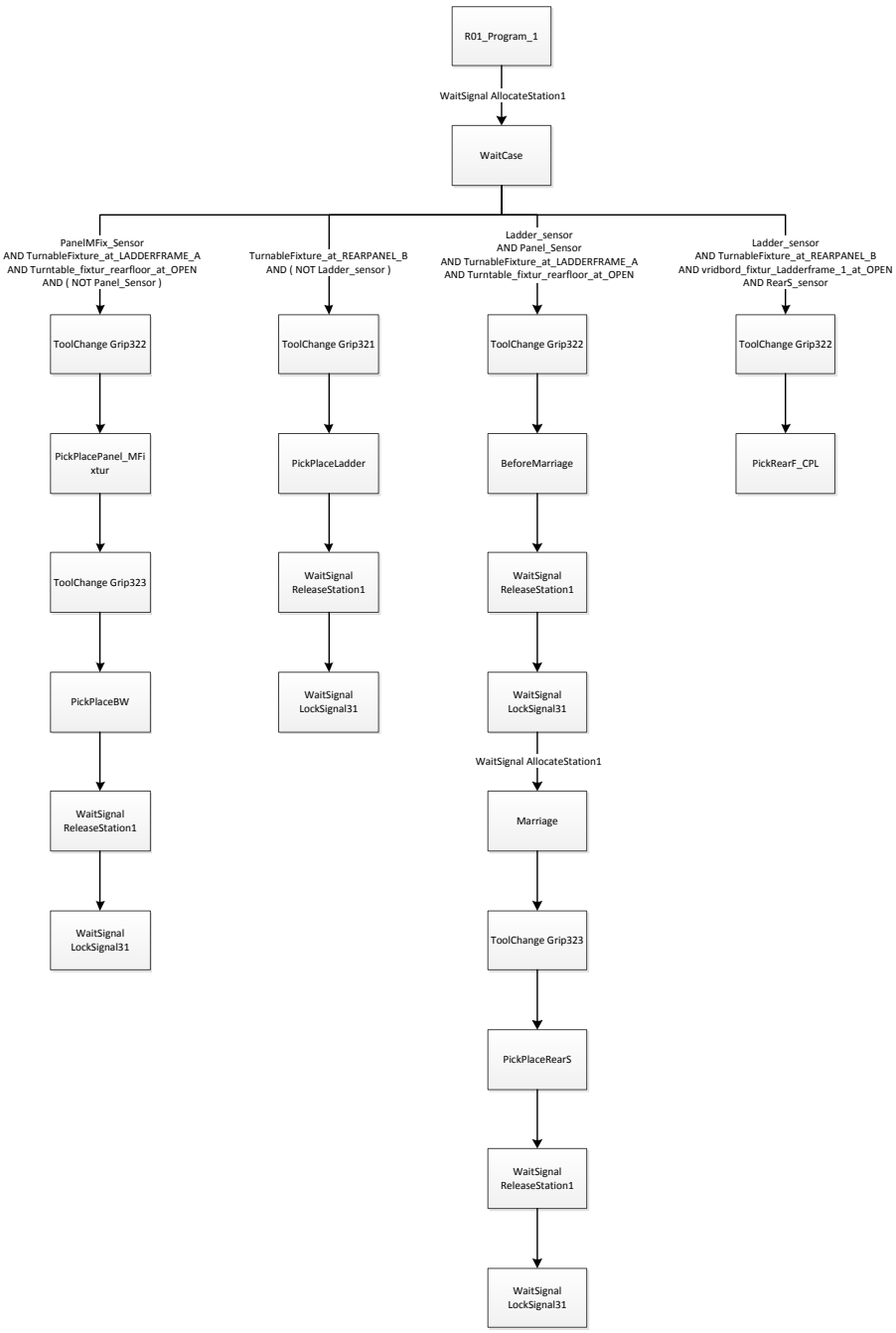
Appendix A

Logic for important resources in the simulation model

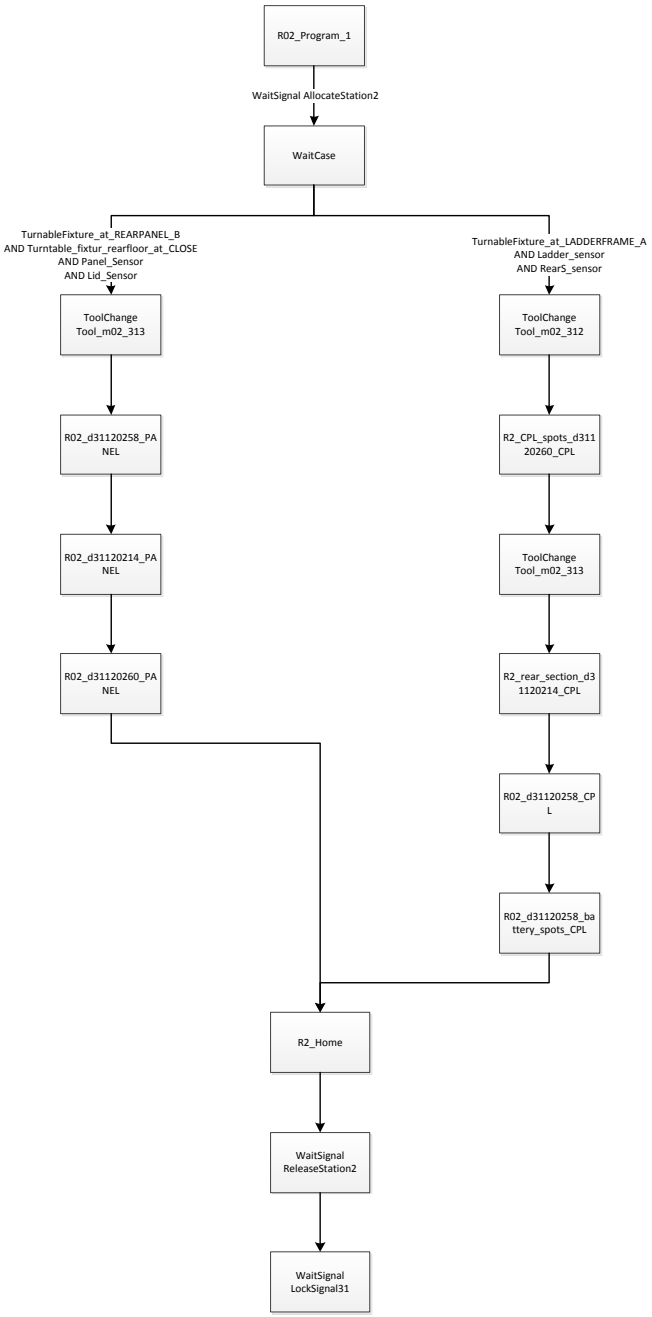
A.1 Logical flow for the rotating fixture



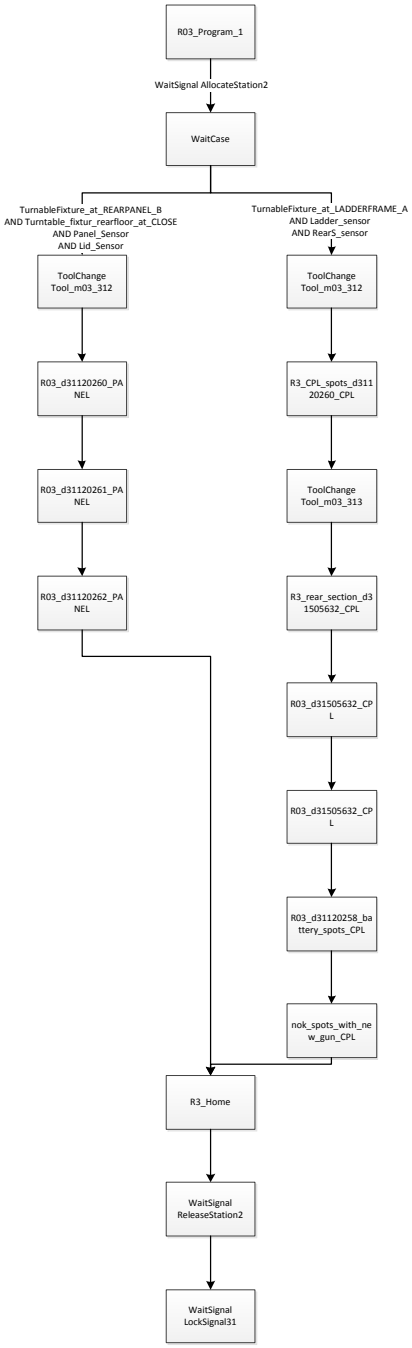
A.2 Logical flow for robot 1



A.3 Logical flow for robot 2



A.4 Logical flow for robot 3



Appendix B

Decision matrix showing problems and associated solutions

Subproblem	**Solution**			
Simulation model				
- *Define kinematics*	**Turn the fixture into grippers**	Seperate gripping entities		
- *Operation sequences*	Time-based concept	Event-based concept	**Partly event-based**	
- *Material flow*	Attach parts to other parts	**Merge IPAs**		
- *Insertion of logic*	Use logic blocks	Use modules	Gantt chart	**A combination of the three**
Extraction of information	Already developed API	Design a new API	eM-Planner	**XML export**
Structure of the information	Restructure with XSLT	**C# solution with XPaths**		
Visualization	**Sequence Planner**	Automation Designer	**Visio**	

Appendix C

XPaths used for data extraction

XPath	Description
/Data/Objects/Robot/@ExternalId	Returns the external id of all robot types
Data/Objects/PmToolInstance[prototype= /Data/Objects/Robot/@ExternalId]	Returns all instances of the available robot types
Data/Objects/PmToolInstance[@ExternalId= '{1}']/plcInfo/plcInfo/*/item/Signal	Returns all signals belonging to the robot instance with the external id {1}
Data/Objects/*/plcInfo/plcInfo/Sensors/ item/*/SensorSig/Signal	Returns all signals from the sensors in the model
Data/Objects/PLCProgram/plcInfo/plcInfo/ */item/Signal	Returns all signals that not belongs to the robots or the sensors
Data/Objects/PmGenericRoboticOperation[operationType = 'SCHEDULER_OPERATION']	Returns all schedulers
Data/Objects/PLCProgram/plcInfo/plcInfo/ Controls/item/Control	Returns all modules blocks, including the one with the main sequence
Data/Objects/*[@ExternalId=/Data/Objects/ PmCompoundOperation[name='{1}']/ children/item and not(starts-with(name,'Get'))]	Returns all operations in the compound with the name {1}. The XPath automatically removes operations created to source material into the model
Data/Objects/PmCompoundOperation[children/item = /Data/Objects/*[outputFlows != ']/@ExternalId]	Returns all compounds which has some logical flow inside

Appendix D

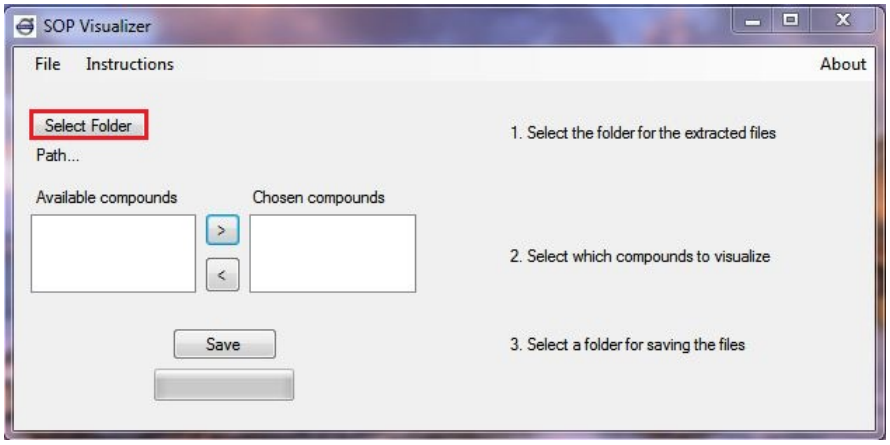
Instructions in the application

D.1 SOP Visualizer

To be able to use the application SOP Visualizer the steps extracting information from Process Simulate has to be done. These steps create four files, named 1.xml, 2.xml, 3.xml, 4.xml. When the files have been extracted the following steps have to be followed to generate visualizable files:

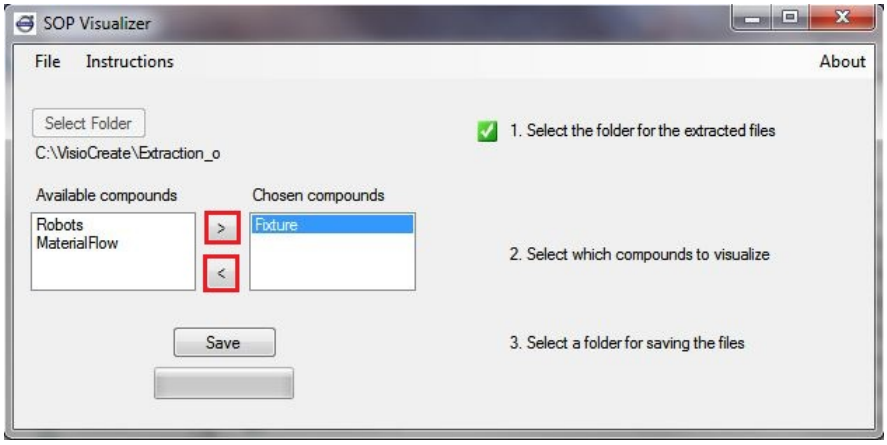
Select the folder where the files were extracted

Select the folder where the extracted files from Process Simulate were placed by pushing the *"Select Folder"* button.



Choose which compounds to visualize

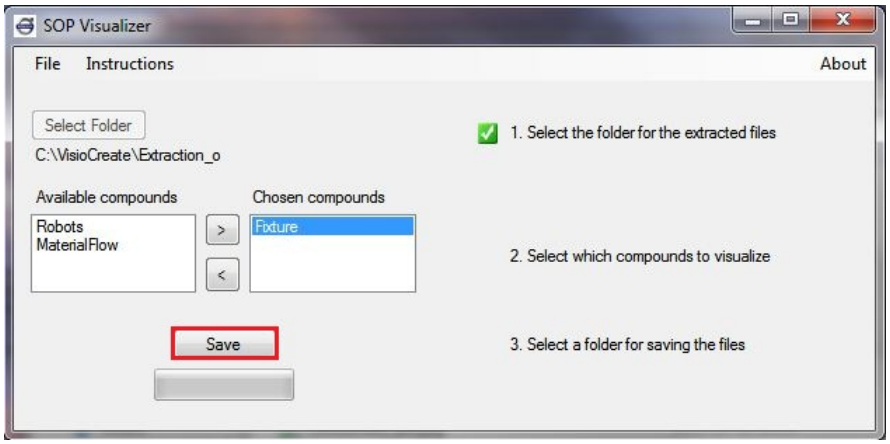
The application lists all compounds containing any logical sequence. The compounds are listed in *"Available compounds"*. All compounds that should be visualized have to be placed in the list *"Chosen compounds"*. This is done by marking it and click on the arrows marked in the picture. Compounds only containing one logical flow where all operations are sequenced are automatically placed in *"Chosen compounds"*.



Note that logical flows of robot programs will be automatically visualized and does not have to be considered in this step.

Choose location for saving the files

Select the *"Save"* button and choose where to place the visualization files. The files are divided by which compound/robot they visualize and also by their robot program number. So for a robot containing two programs there will be two files.

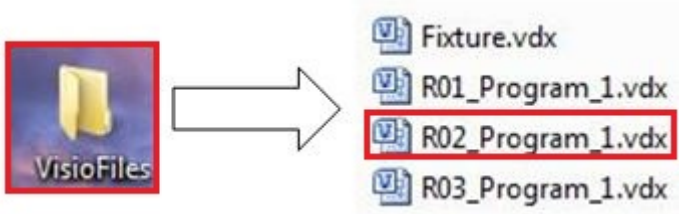


D.2 Visio

The visualized sequence outputted from SOP Visualizer is not arranged in a viewable way. All operations are presented on top of each other and the chart must therefore be re-arranged. To re-arrange the flow some simple tasks have to be done in Visio.

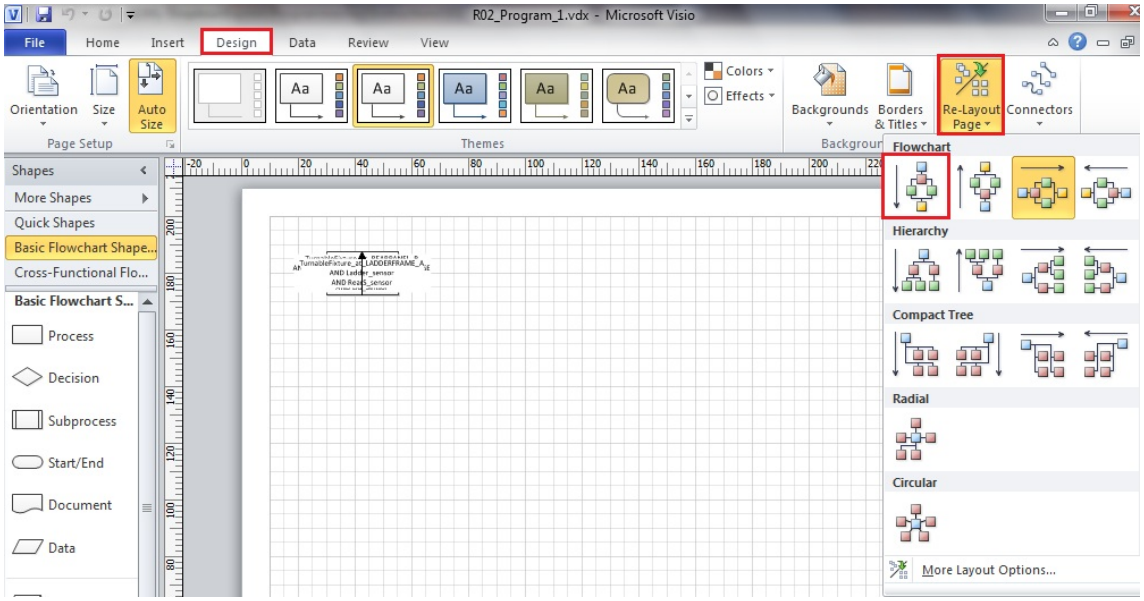
Open the file

The Visio-files are outputted in a folder named *VisioFiles*. Open the folder and select the file you want to visualize.



Re-arrange the operations

When the file is opened the blocks are re-arranged by using a built-in Visio functionality. Start by selecting the tab *Design* and press *Re-Layout Page*. There are a number of options of how the sequence can be arranged. To get a standard top-bottom flowchart, select the option marked.



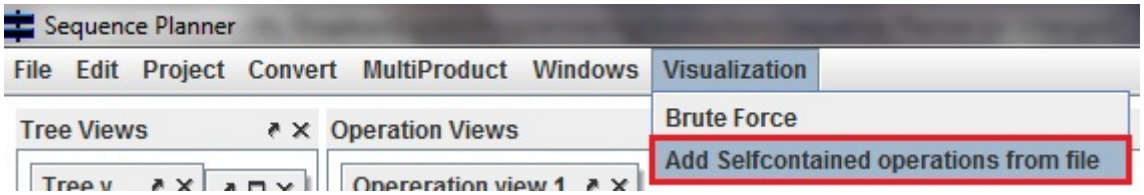
This will re-arrange the operations in a top-down manner. Some manual arrangements may have to be done if the transitions contains too much information making them overlap. To manually re-arrange some operations select the operations and drag them until all transitions can be seen.

D.3 Sequence Planner

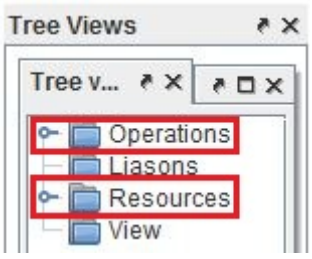
To get the sequence visualized in Sequence Planner the outputted files have to be imported into Sequence Planner. The files outputted are in txt-format and placed in a folder named *SPFiles*. Start by opening Sequence Planner and follow the steps.

Import sequence

In the menu, choose *Visualization -> Add SelfContained operations from file.*

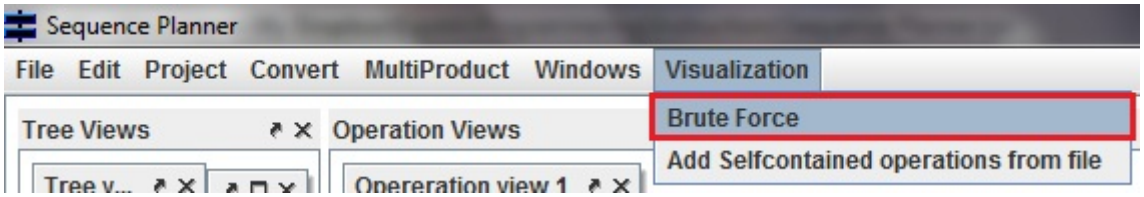


A dialog will pop-up, choose the sequence you want to visualize. The operation should then be added in the Tree View, seen by the circle symbol added before the operation and the resource nodes.

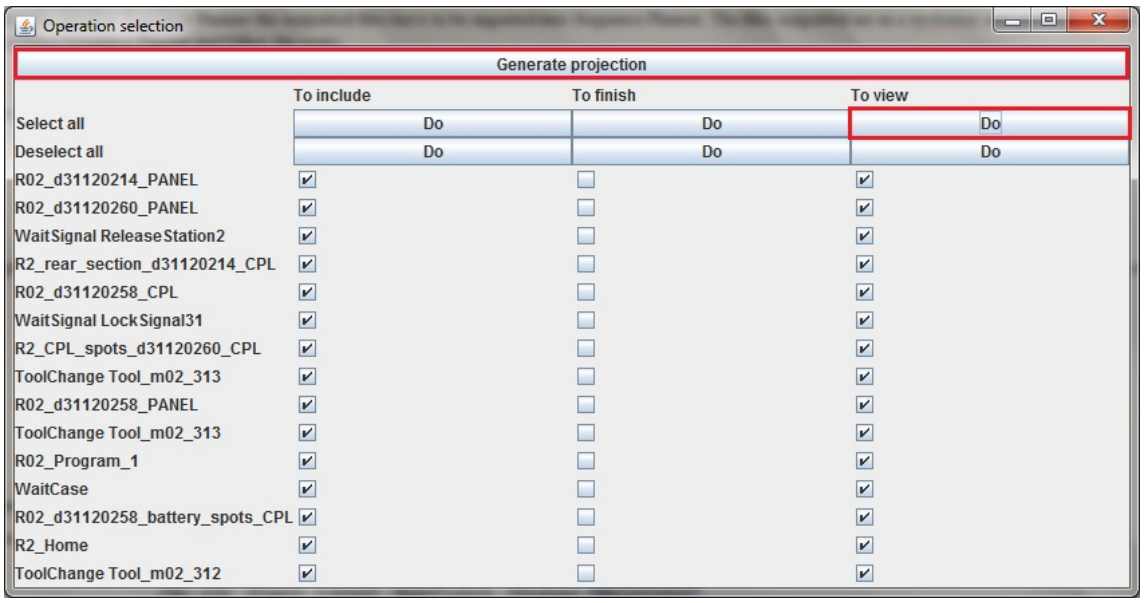


Visualize sequence

When the sequence is imported it is visualized by choosing *Visualization -> Brute Force*.



When selecting Brute Force the *Operation selection* window pops up. This window has three columns; *To include*, *To finish*, *To view*. The important column for visualization is the *To view* column. In this column specific operations can be chosen or as in most cases the whole sequence (all operations). To visualize the whole sequence choose *Select all* and press *Generate projection*.



Close the Operation selection window to see the visualization.