DIPLOMARBEIT

Development of a short-time-implementation tool for FMC

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter der Leitung von

ao. Univ.Prof. Dr. Burkhard Kittl
E311
Institut für Fertigungstechnik

eingereicht an der Technischen Universität Wien
Fakultät für Maschinenwesen und Betriebswissenschaften

von

Roland Schmalhofer
Matnr. 0226033
Esterhazyg. 31/1/11; 1060 Wien

Wien, am 2. Juni 2008

Contents

List of Figures

1 Introduction

1.1 Common

Manufacturing (lat. *manu facere* = "making by hand") includes all necessary processes for the production of a component. Generally, manufacturing processes intend to achieve added value for a product in order to finally sell it as dearly as possible. This is just one of the reasons why our economy is inconceivable without manufacturing.
Most of the production processes are closely connected with the need of many human resources, which however, renders manufacturing fairly expensive. Consequently, many companies attempt to automise production processes to a great extent. Therefore automation and manufacturing engineering have become more important than ever before.

The interaction between mechanical engineering and modern information technology offers a set of new possibilities. So, novel control tools for manufacturing systems can be developed that are not just flexible in their applicability: modern software systems should be user-friendly and easy to use in the first place.

The demand to build up a user-friendly control system for manufacturing cells is the focus of my thesis and the related software system named ProCO (ProductionControl). The aim of this study is to create a software tool that enables us to control the activation and deactivation of the single components, i.e. robots, turning machines, conveyor belts, while the concrecte programming of the machines, e.g. CNC programmes or robot movement control, is done on the machines themselves. So, the main task of the tool is confined to the execution (start) and the recognition of the termination of the machines. The tool does not interfere with running programmes on the machines or with the information flow of the manufacturing cells. The activation/deactivation commands themselves are mainly done by simple digital input/output signals. Almost every machine supports this kind of control innately. The start signals can also be replaced by user defined DLL functions so that solutions for special cases (e.g. when machines provide other ways of control that digital I/O's) can be realised, too.
The source code of ProCO has about 6000 lines of code. The development of ProCO is the first part of this study. The second part of this thesis is based on an exemplary realisation which shows the use of ProCO in practice.

The main aim of ProCO is that the building of an execution process should be kept as simple as possible. The user is not required to have any experience in common programming languages. The only disadvantage is that such a system does not cover all scenarios. However, this is not the aim of the system developed. It ideally fits for ad-hoc solutions which should be implemented in a short time and is used for a small or medium batch size. One important characteristic of ProCO should be that the software-sided design time lasts less than one hour for a medium complex production flow. This will be realised by an intuitive design which keeps the adjustment time short for a new user.

1.2 Application areas

The main application areas are small systems whose single components were used up to now in a manual and not connected way. So, a high level of automatisation can be reached especially in the field of work piece handling. A first impression of the usage of ProCO should be given in the following example.

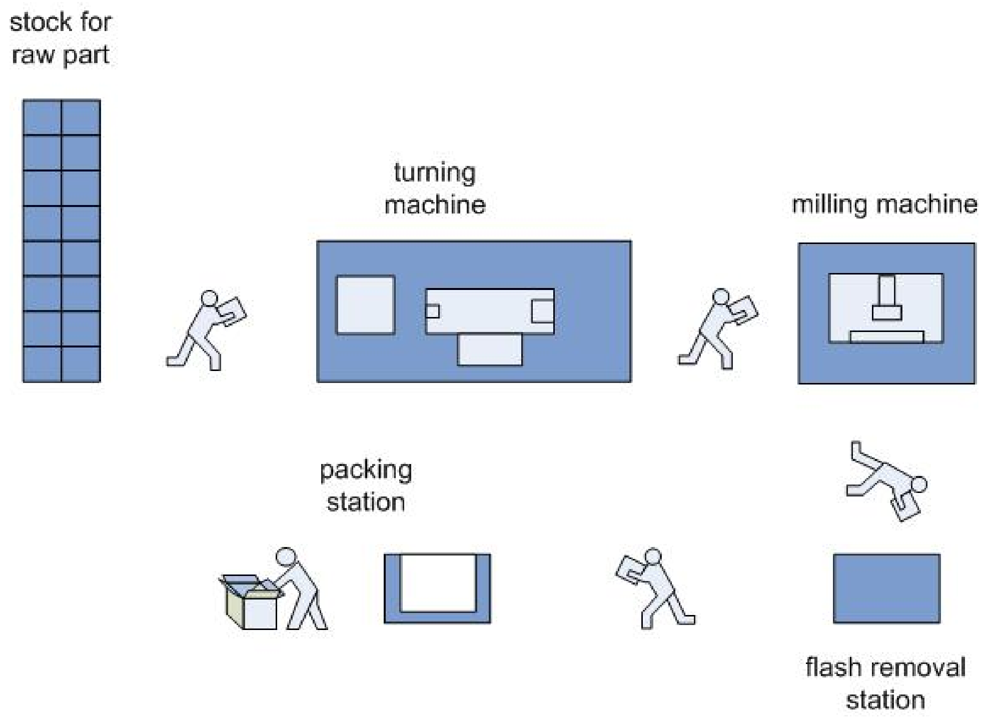


Figure 1: Manual connection

The figures 1 and 2 show what such automation could look like. In the example above a company produces parts which are first of all cut by different chip removal - machines. In a second step the flash will be removed and the finished parts will be brought to a packing

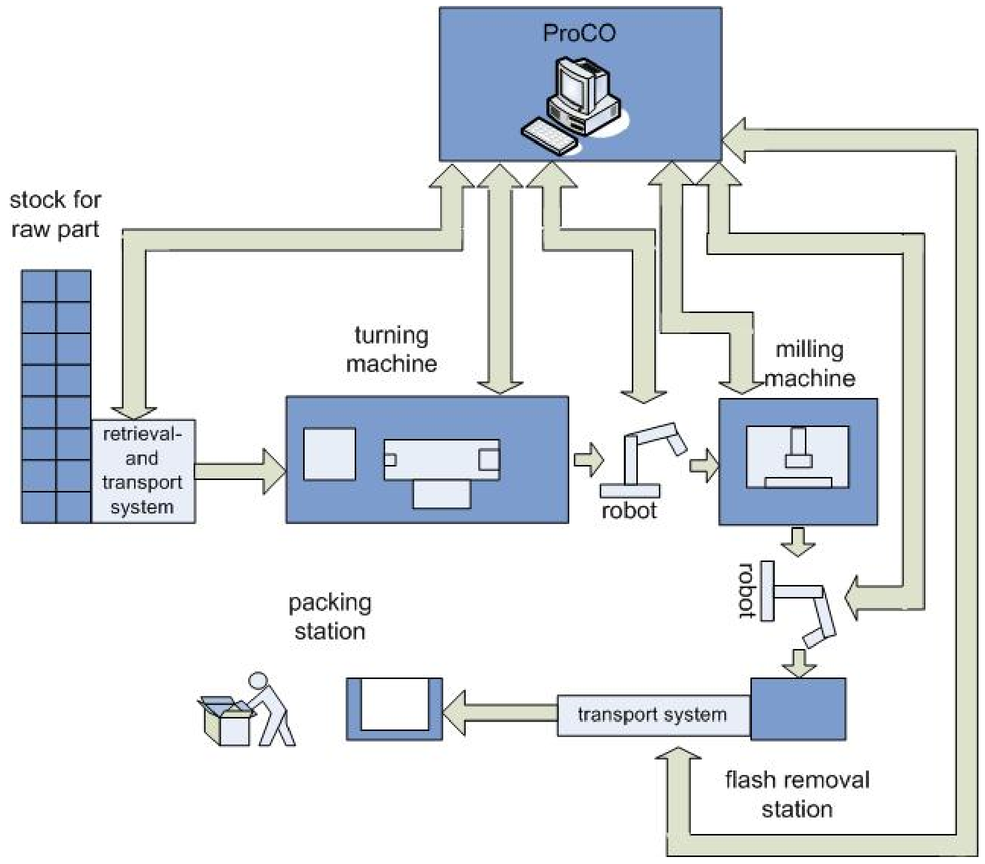


Figure 2: Automated connection

station, where they will be boxed for shipping. All these actions were done manually (See *figure 1*). Now, the company wants to automate this process. Figure 2 shows the possibilities of automation. The arrows from the host computer to the components illustrate, that the component is started by the host and receives signals from the components telling the host computer that they have completed their execution. Hence, the communication is bi directional. In this example, two robots and two transportation systems are pictured. With a clever arrangement of the components it is mostly possible to get on with just each one of them. So it could be used just one robot for the loading of the milling machine and the transport to the flash removal station.
This system is a simplified model. In fact, there must be systems to span the workpieces for the machining at the milling and the turning machine. Such systems could be controlled by ProCO as well.
It is well known that not every task can be done in a cost-effective way by automated systems. The packing for instance could be a too complex activity to be managed by a robot. In this case special systems would be too expensive for the small batch size.

1.3 Requirements

The intention is to develop a control system for small manufacturing cells. It has to be possible to store, reorganise and execute the process flow of machine control. It should be possible to execute more than one process simultaneously so that an optimal use of the whole cell can be ensured. To achieve this, it is essential to implement a waiting queue logic that ensures that no overlapping occurs.
The communication with the components is confined to the sending of a start command and the recognition of the end of the component execution. This shall be realized with digital inputs/outputs.
The whole project, including the components and process orders, should be saved in a single file. The installation of an additional database system should not be necessary; it should simply be possible to transfer projects from one computer to another.
A further important aspect is the possiblity to customise the control so that other ways of initial methods in addition to digital I/O's can be used. An example would be RS 232 C. This can be realised by the implementation of DLL interfaces. Thereby, users would have the option to write own function and include it in the execution processes. Another demand is that errors must be handled.

The execution process itself would ensure an automatic process flow by activating the single components in a linear or a parallel way. In addition, good visualisation, which is the active component and which shows what actually happens in the cell, is paramount. The system should be programmed in a way to make future enhancements easy to implement.

1.4 Rentability

As described in *Common* above, the time available to produce/develop new products is getting shorter and shorter so that the demand for flexible and fast-production structures increases steadily.
The most important aspect is the fact that, because of changes in the market, enterprises can modify very quickly their production programme. It is in general not very expensive or time-consuming to implement these changes.
Another advantage is that if the production volume increases, a manufacturing cell can be easily extended. New CNC machines, for example, can be included in the existing system. Furthermore, different products can be manufactured without basic changes in the arrangements of the respective configuration.

Further positive and measurable side effects are:

- shorter cycle times
- smaller stocks
- less and shorter downtimes, in a technical and organisational context
- lower costs for human resources
- higher period of use of the machines because of 2nd and 3rd shifts of production
- and, as a result, lower production costs

The importance of such a manufacturing cell has to be checked in every case:
If the number of pieces produced is low while the flexibility demanded is high, it will perhaps be more useful to work just with stand-alone CNC machines.
The other case is that almost no flexibility is needed, because just one specific product is produced and the output is very high. A manufacturing cell is the best solution, provided that the above criteria are taken into consideration.

1.5 Standard components

The main components of an MC are in general the same. (*q.v.[Kie98]*)

1.5.1 Machining Units

The most commonly used variants are:

- CNC lathe
- boring and milling machine
- assembly stations
- grinding machine
- special purpose machine

Chip removing machines are a special field of interest in this study. Particularly in their application in flexible manufacturing cells, it must be verified that the machine fits for manufacturing cells. There is a wide range of criteria which must be fulfilled. Two of them are of special interest for the software developed:

Programme Calls The machine must be able to start a specific CNC (DIN 66025[1]) programme by an initial signal (eg. RS 232 C, 20 mA ...) from the start to the end. Another very practicable variant would be for a CNC programme to run in an endless loop and for the machine to stop at a specific point that has been pre-defined in the programme and that is usually realised by a special M-code command. The exact number of that command depends on the type and the producer of the machine.

Possibility of loading It must be possible to load the machine in an automatic way. Variants could be the loading by a pallet changer or via a robot. Automatic door-opening is a prerequisite for this solution. Size compatibility is another need. The geometric preconditions, such as the height or width provided for the pallet places must be the same as for the single components (e.g. conveyor belts and turning machine). The supply of material via rod loaders for turning machines is also a very practicable solution.

1.5.2 Work piece transport and handling

The chaining of the single stations is a basic demand of manufacturing cells. There is a wide range of implementation areas of such systems, depending on the size of the work pieces and the number of parts produced.
The main issue is the transportation of the work pieces. It must be possible too to start transportation processes via an initial signal by the host computer on which the developed software is running. The solution is mostly reached by an SPS-controlled system that has digital inputs/outputs and that can be set in order to start a specific action or to confirm the completion of the respective order.
The common forms of realisation are rail-bounded transport carts, robots or pallet handling systems.
The aim of these transport systems is the reduction of the cycle-time and the production costs by avoiding interim storages, by employing better feeding-conceptions at the machines, as well as by a better machine-usage quota.

The transport within the system, in general, is mostly handled with pallets. A changing pallet, for example, is the top part of a handling table with a fixed chuck device. It can be completely moved into the machine, so it does not have to be spanned anew before the machining. An alternative is to use the pallet just as an underlay. One or more pieces are put onto it and are then moved into the machine via robots or changing devices.

[1]English-speaking: G - code

Robots are a further basic component in handling materials and work pieces in a very flexible way. They can be programmed and adapted for various problems, because it is possible to move in six axes and to put different grippers on the robot arm. Thereby, small and large, as well as light and heavy pieces can be moved this way. Robots can be generally well controlled by host systems. Programmes which are programed at the robot itself can be called and executed by start signals (eg. DI/DO, ...).
A distinction is drawn between articulated robots(*fig. 3 Articulated robot*[2] and *fig. 4 SCARA robot*[3]) and portal robots:

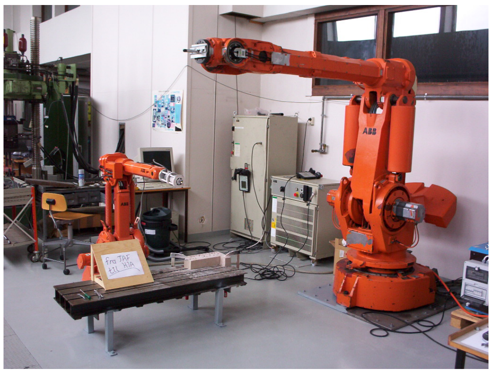


Figure 3: Articulated robot [oA08]

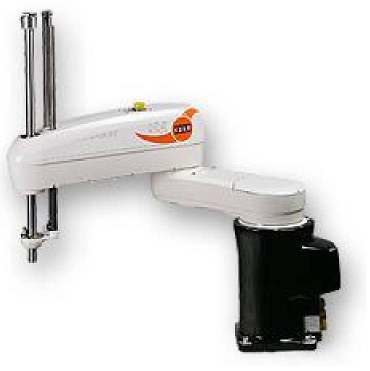


Figure 4: SCARA robot [KR08]

Portal robots have a wide action area. Furthermore, the accessibility of the machines is very good, because the setting is effected from the top.

[2]6 - axes robot

[3]A special variant of robots. It can only handle in 4 axes. Lifting can be done with just one axis. Ideal for handling in one axis.

2 ProCO - Functional specification

2.1 Goals

The software system that has been developed in the course of this research project provides for creating, managing, reorganising, storing and executing production processes within a manufacturing-cell.
It allows to rebuild the process as a computer model within little time. Another important aspect is the simplicity of these design activities. It can be reached by the orientation by the common windows behaviour and an intuitive design that makes it possible to learn the functioning of the system without reading the manual first. Methods like drag-and-drop and navigation within descriptive lists replace the necessity of programming-knowledge. This method enables nearly every computer user to build a manufacturing-process.

2.2 Separation from other systems

There are many systems on the market that allow us to control a production process. However, well-experienced experts to handle such systems are needed, which leads to the question: What if the user wants to implement a small system with simple and defined parameters?
In this case, the result of such software packages would be large models that need much time to be developed. So, the aim of this study should be to build a system that purposefully leaves out some options, with the advantage of having a smaller and clearer system.

2.3 Hardware

ProCO is programmed in the .NET Framework in the version 1.1 of Microsoft. Therefore, it is runnable on Windows XP and Vista-based computers. The requirements in the CPU power and the main memory are not complicated. It works with every computer on which the OS's listed above are running.

For the communication with the production-components, the systems works with nearly every Digital-I/O-Card of NI (National Instruments), one of the market leaders for such peripheries. It does not matter if it is a PCI, PCI Express or an USB component. ProCO can be configured with parameters for each of these products. It will be described in great detail in the following research how to do this.

2.4 Activities

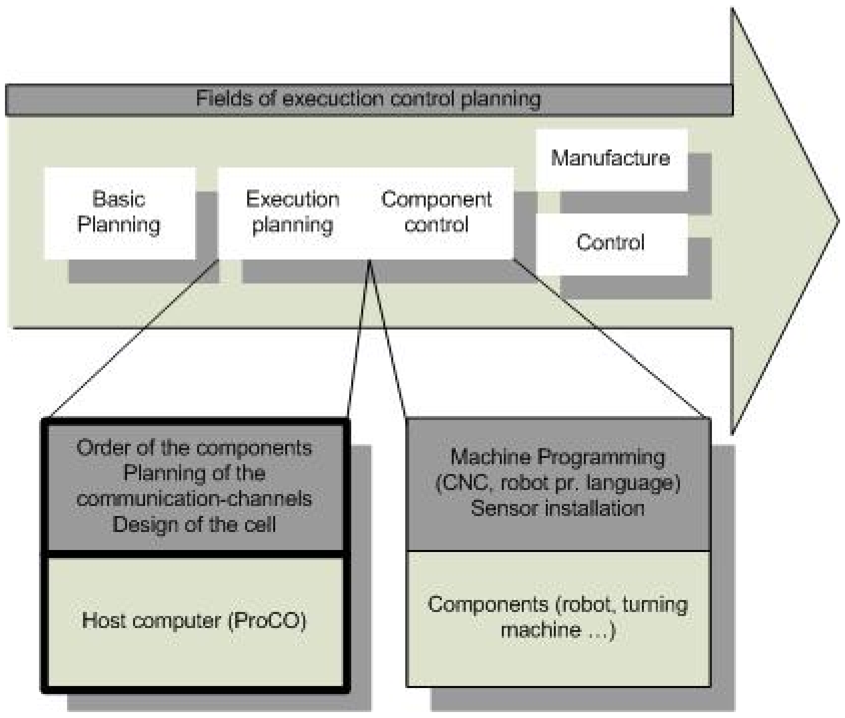

Fields of execuction control planning
Basic Planning
Execution planning
Component control
Manufacture
Control
Order of the components
Planning of the communication-channels
Design of the cell
Host computer (ProCO)
Machine Programming (CNC, robot pr. language)
Sensor installation
Components (robot, turning machine ...)

Figure 5: Activities

The figure above demonstrates the common order of the required cycle in realising a manufacturing cell.

Basic planning The first step is the design process. It must be defined which product/product group will be built/assembled and which actions have to be taken to realise it. Furthermore, it must be clarified which components, such as turning machine(s), robot(s) or cleaning stations, to name but a few, are necessary. The last step is to design the layout, i.e. which component is to be placed where. This is a fundamental contribution to the efficiency of such a system. Transport systems can be made more effective through a smart assignment of robots or other handling systems, for example.

Execution planning This part includes all planning activities which must be done on the host computer, especially on the corresponding software tool (ProCO). **This is the part that will be executed by ProCO, and it is the central part of this work. The bold border in the figure underlines this.**
Specific details of this step will be given in the following chapters, in particular when the example of a realised flexible manufacturing cell is discussed (at a later stage). The essential points are:

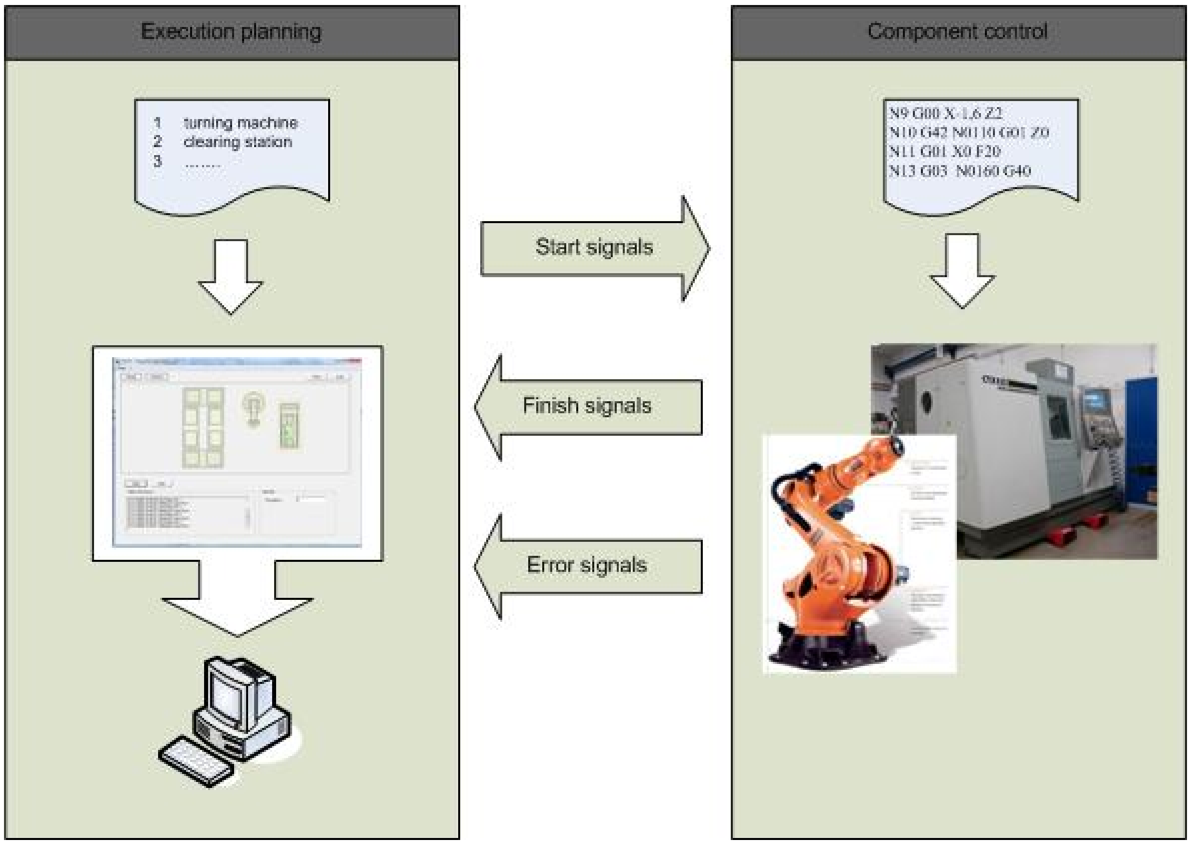


Figure 6: Connection Hard-, Software

- definition of the order of the single machines
- determination of the communication-channels (eg. Digital I/O or via special DLL functions: RS 232, COM-calls via TCP/IP - Networks ...)
- determination of stop criterias (eg. emergency stop, failure in the process, ...) which must be observed during the production process
- determination of parameters which can be defined later in the control section

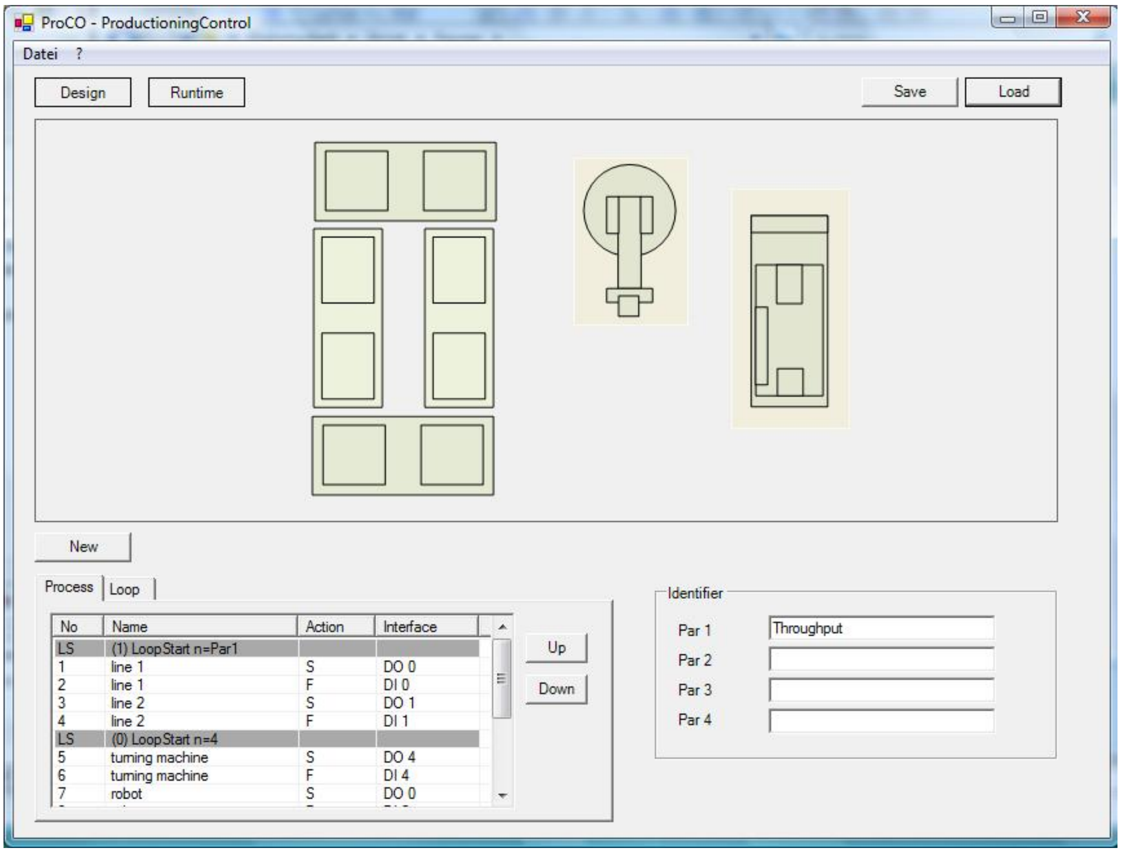
Component Control The activities in this part are mainly taken on the components themselves. These are the programming of the machines (DIN 66025, robot control language, SPS control cycles, ...) or installing and connecting sensors.

Manufacture Describes the process of the real production sequence. It is the realisation of the two parts *Execution planning* and *Component control*.

Control At this stage, the software tool is responsible for the appropriate execution of the programmed sequences. Multi-threading must be managed, too, which means that multiple orders can be delivered to the manufacturing-cell and the software has to manage

- by the help of queues - which component can be used by which order process. An efficient implementation algorithm can help to save time. Error-handling will also be of great importance in this respect. So the ports which are defined by the user as error ports must be checked for incoming error signals.

2.5 Functions and design



No	Name	Action	Interface
LS	(1) LoopStart n=Par1		
1	line 1	S	DO 0
2	line 1	F	DI 0
3	line 2	S	DO 1
4	line 2	F	DI 1
LS	(0) LoopStart n=4		
5	turning machine	S	DO 4
6	turning machine	F	DI 4
7	robot	S	DO 0

Par 1	Throughput
Par 2	
Par 3	
Par 4	

Figure 7: Main-mask

Figure 7 shows the main-screen. It pictures the most relevant control elements. The system is divided into two parts:

- Design
- Runtime

These areas can be reached via the two buttons on the top of the screen (fig. 8)
The design mode allows to create the process sequence. In the runtime mode the developed

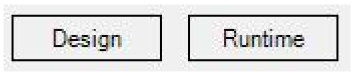


Figure 8: Modes

process and the manufacturing system can be started. The control elements in the rest of the screen change depending on the current mode.

2.5.1 Workflow

Figure 9 shows an overview of the steps to take to build a complete model. The process *Adding the components* is pictured in great detail in figure 10. The single steps and the meaning of the attributes will be described in the following chapters.
The label *XOR* next to the branchings means that just one of the following steps can be chosen. All other branchings are seen as conditional.

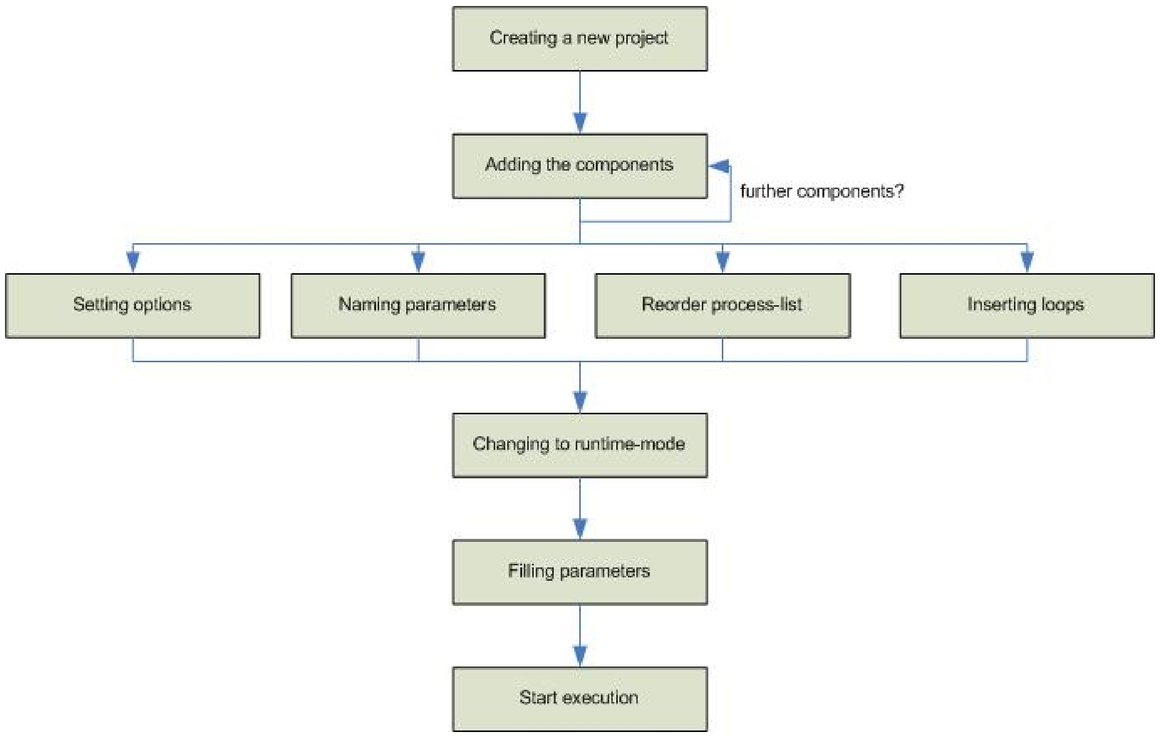


Figure 9: ProCO - Workflow

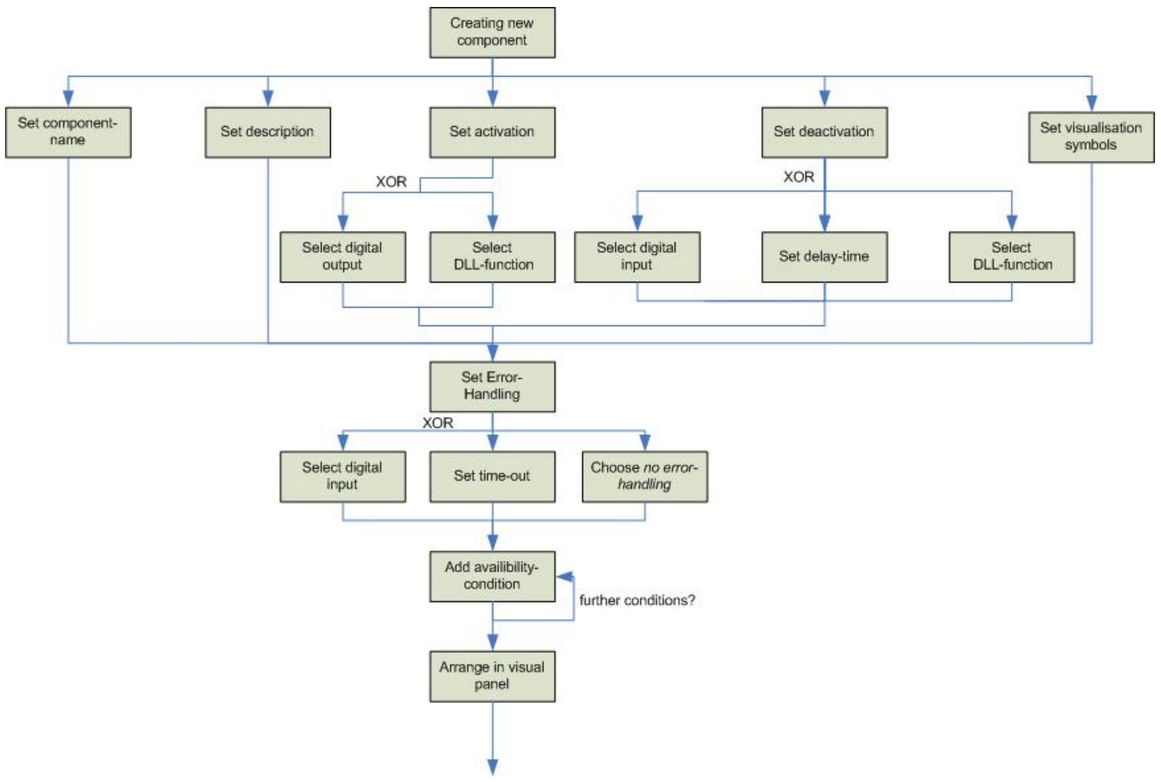
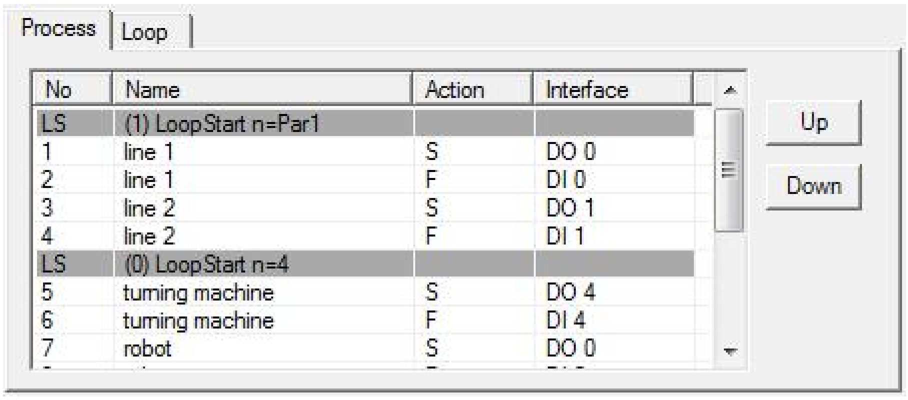


Figure 10: Adding the components

2.5.2 Design - mode

In the *figure 7* on *page 16* the design mode is illustrated. There are two large areas: the visual panel which allows us to arrange the single components in a graphics panel via drag and drop and the second in which the execution order of the activation/deactivation of the machines can be determined.

Visual panel The visual panel is shown later in the runtime mode. This view is intended to give a better overview during the execution in the runtime mode. Thus, the user can quickly control which components are activated.
The pictures which represent the components can be determined in the relevant property sheet (q.v. Chapter *Adding a component*). The most popular graphic formats can be used (eg. .jpg, .gif, .bmp, ...). Nearly every graphic programme is able to create such pictures. Illustrations for the commonly used components (e.g. turning machines, conveyor belts, robots ...) are included in the software package. As mentioned above, personalised symbols can be created very easily.



No	Name	Action	Interface
LS	(1) LoopStart n=Par1		
1	line 1	S	DO 0
2	line 1	F	DI 0
3	line 2	S	DO 1
4	line 2	F	DI 1
LS	(0) LoopStart n=4		
5	tuming machine	S	DO 4
6	tuming machine	F	DI 4
7	robot	S	DO 0

Figure 11: Process-order-determination

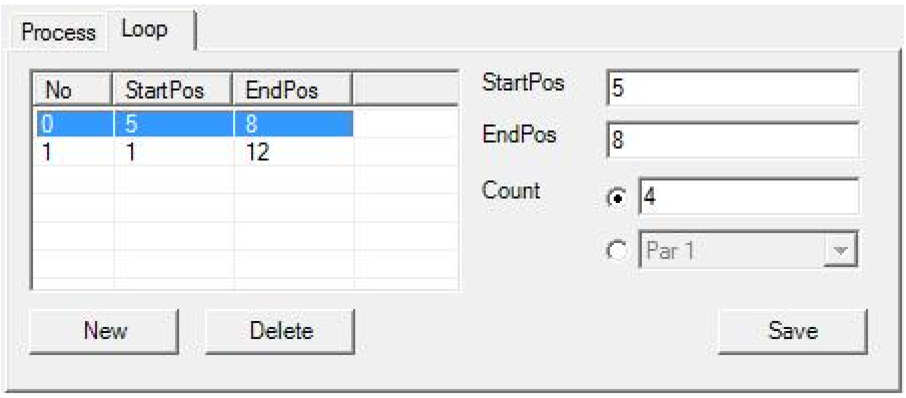
Process - order - determination Every component (in general) has two entries in the list of the property sheet Process. An exception are components which are finished with a delay time. For this case just a start - entry exists. The list shown in *figure 11* has four columns:

- No
  This represents the position in the order of the process chain. Alternatively, this cell contains the markers LS (loop start) or LE (loop end). These markers either describe the start of a loop in this line (LS) or that the end of a loop is reached (LE)

- Name
  Gives the name of the component. Alternatively, the data of a loop are listed in this cell. In this case, the index of the loop is put in bracket, so that it is easier to find the corresponding start-and-end entries. A further piece of information is the count of the loop. There are two possibilities: via a parameter or via a fixed number. These options are delineated in the next chapters.
- Action
  This marker determines whether it is the activation (S...Start) or the deactivation (F...Finish) of the component. It is empty, if the entry is a loop definition.
- Interface
  It displays the interface that is used for this action (Digital I/O, COM ...). It is empty if the entry is a loop definition.

The two buttons *Up* and *Down* are intended for the reorganisation of the order. When a line in the list is selected, reorganisation is executed by clicking on one of the buttons pushed one position higher or lower in the sequence. The numbers are automatically re-counted. The order can be changed by the property sheets, too (see chapter *Adding a* *component*).

The second tab is named *Loop*. It creates the definition of a specific area which is to be repeated several times.
The list contains three columns:



No	StartPos	EndPos
0	5	8
1	1	12

StartPos 5
EndPos 8
Count (● 4) (○ Par 1)

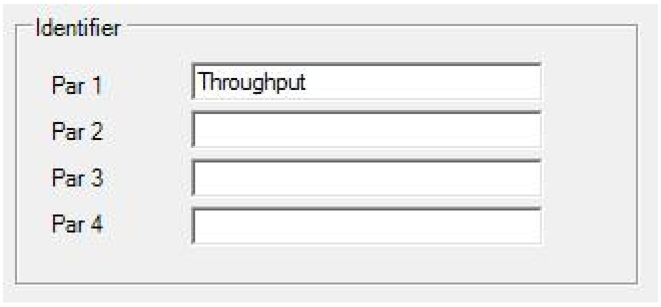
Figure 12: Loop

- No
  This is the identifier of the loop (Loop - ID). It is given in the process list (see column name in brackets).
- StartPos
  It is the first position which will be repeated in the execution mode.
- EndPos
  This is the last postion of repetition.

The button *New* creates a new loop. With the button *Save* the changes of an already existing or a new entry are written into the data set. *Delete* removes an existing loop. The parameter *Count* represents the number of times a loop is repeated. There are two options: The first one is to define a constant number, the second one is to determine a parameter.

In ProCO four parameters are provided. If the users decides to choose the option parameter, they have the chance to enter the number of repetition in the runtime mode. This can be used, for instance, to determine the exact number of parts that shall be produced. Every parameter can be given a meaningful caption. It will be shown in the runtime - mode.

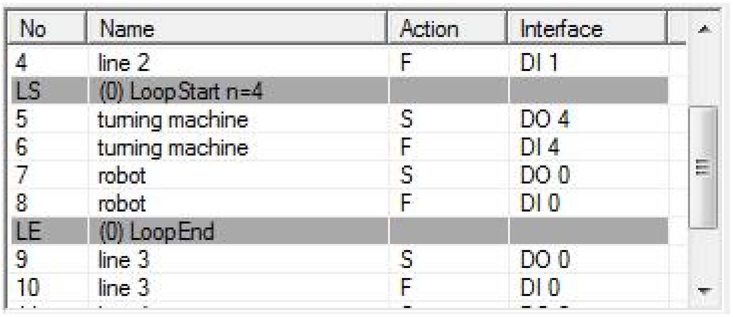
The loops created are shown on the right position in the process - order - list in the



Identifier

Par 1	Throughput
Par 2	
Par 3	
Par 4	

Figure 13: Parameters - Identifier



No	Name	Action	Interface
4	line 2	F	DI 1
LS	(0) LoopStart n=4		
5	turning machine	S	DO 4
6	turning machine	F	DI 4
7	robot	S	DO 0
8	robot	F	DI 0
LE	(0) LoopEnd		
9	line 3	S	DO 0
10	line 3	F	DI 0

Figure 14: Loop - Entry

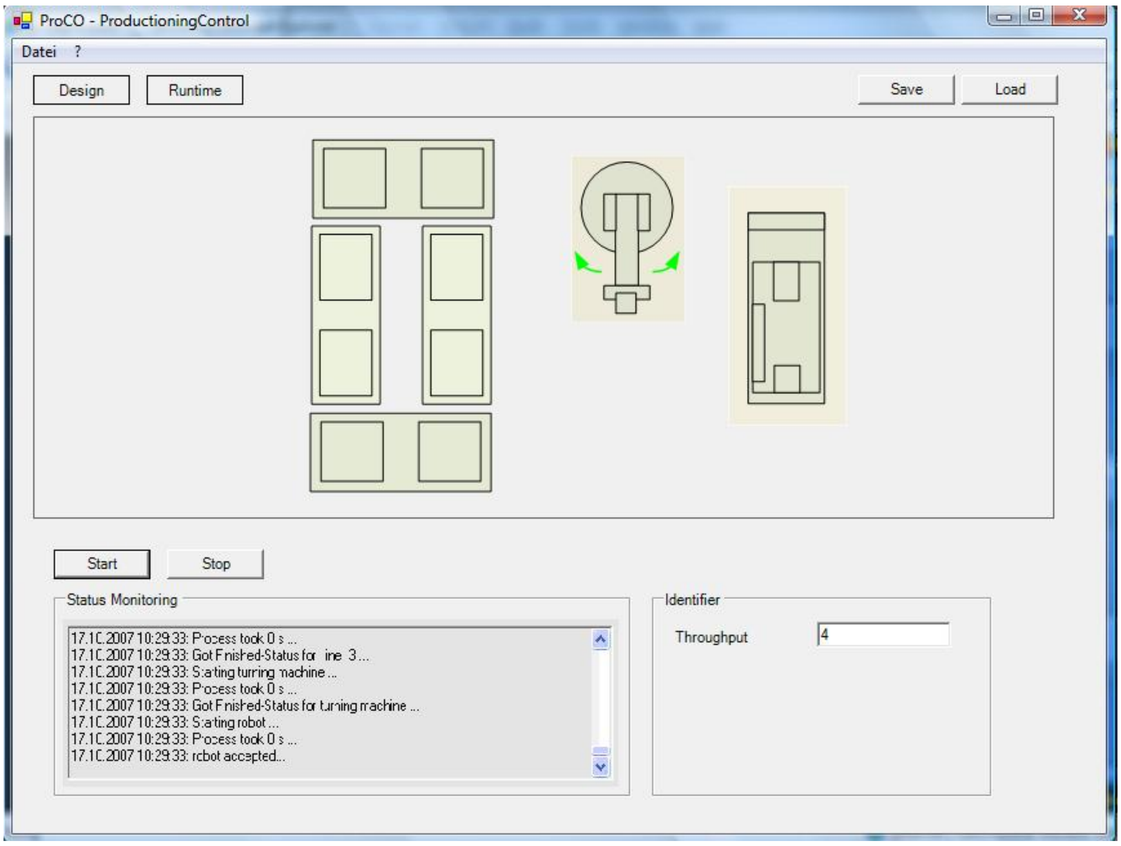
tab *Process*. The number in the list is replaced by the marker *LS (loop start)* or *LE (loop end)*. The column *Name* contains the loop ID in brackets and the kind of count. This is n=Parx, if the option Parameter was chosen or n=x if a constant count was chosen (x stands for the number of counts).
The lines which include loops are furthermore marked with a gray background - colour. These lines can be moved via the Up/Down - buttons, too. The informations about the start-/end-postion is synchronised with the tab *Loop*.

2.5.3 Runtime - Mode

To execute the production sequence created, the user must switch to the runtime mode. Instead of the process list and the parameter identifiers, two new control elements appear. The first is the status panel. It shows:

- the current date and time
- which component is started and which output signal is set or
- for which component the procedure is waiting (including the corresponding input channel) to be finished

The second control elements are the text fields, which can be used to enter the values for the parameters that were used in the process order determination. The label for the text



```
17.10.2007 10:29:33: Process took 0 s ...
17.10.2007 10:29:33: Got Finished-Status for line 3 ...
17.10.2007 10:29:33: Starting turning machine ...
17.10.2007 10:29:33: Process took 0 s ...
17.10.2007 10:29:33: Got Finished-Status for turning machine ...
17.10.2007 10:29:33: Starting robot ...
17.10.2007 10:29:33: Process took 0 s ...
17.10.2007 10:29:33: robot accepted...
```

Throughput: 4

Figure 15: Runtime - Mode

fields are the identifiers which were given in the design - mode. Only those parameters that were really used are shown.

After the declaration of the parameters the sequence can be started with the start button. The process can be stopped at any time with the stop button. Attention: when a robot is started, it will start the programme that is programmed at the robot and will finish it!

The visual panel looks very similar to the one in the design mode. The difference is the fact that it is no longer possible to change the arrangement via drag-and-drop. During the execution, the active component picture is replaced by the image, which is defined for the active state (also see chapter *property sheet*)

2.5.4 Store and recall of configurations

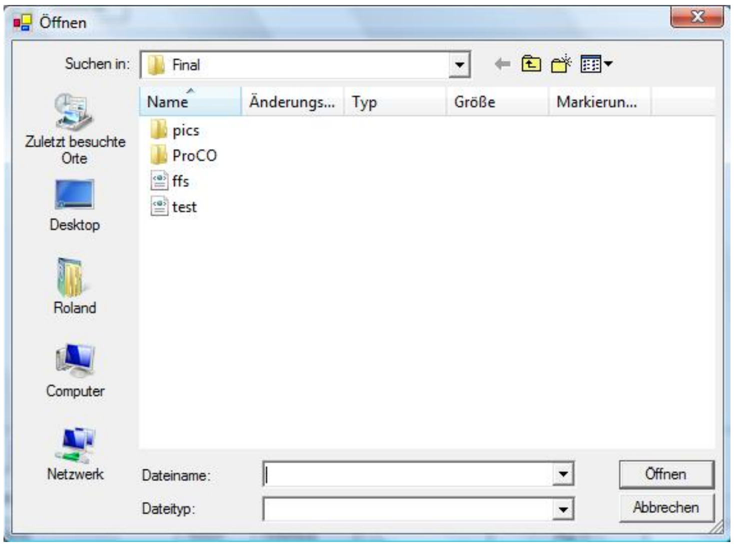
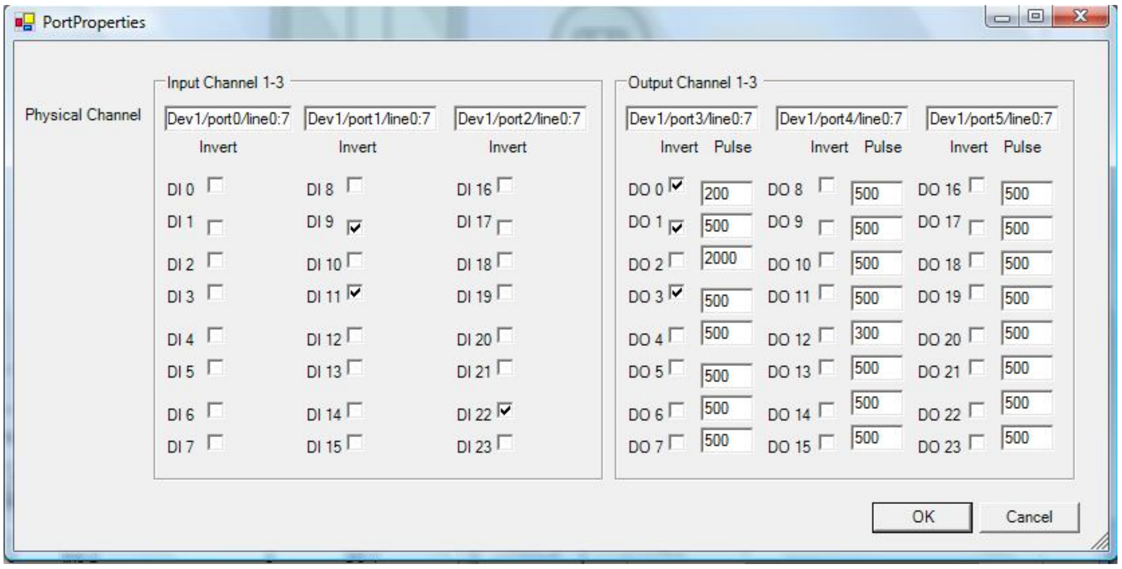


Figure 16: Open - Dialog

The whole configuration (process order, content of the visual panel) can be saved in a file. The format of the file is XML (eXtensible Markup Language). Is is designed to be human-legible. Therefore, the user can gain a first impression of the set-up by opening the file in a simple text editor. Actually, it would be possible to write the whole configuration in a text editor. This, however, is not recommendable because of the bad overview and the complexity of these files.
The files can be easily saved and opened by the standard windows dialogues (see *figure 16*). It can be activated by the menu entry *File*.

2.5.5 Settings



Physical Channel

Input Channel 1-3

Dev1/port0/line0:7	Invert	Dev1/port1/line0:7	Invert	Dev1/port2/line0:7	Invert
DI 0	☐	DI 8	☐	DI 16	☐
DI 1	☐	DI 9	✓	DI 17	☐
DI 2	☐	DI 10	☐	DI 18	☐
DI 3	☐	DI 11	✓	DI 19	☐
DI 4	☐	DI 12	☐	DI 20	☐
DI 5	☐	DI 13	☐	DI 21	☐
DI 6	☐	DI 14	☐	DI 22	✓
DI 7	☐	DI 15	☐	DI 23	☐

Output Channel 1-3

Dev1/port3/line0:7	Invert	Pulse	Dev1/port4/line0:7	Invert	Pulse	Dev1/port5/line0:7	Invert	Pulse
DO 0	✓	200	DO 8	☐	500	DO 16	☐	500
DO 1	✓	500	DO 9	☐	500	DO 17	☐	500
DO 2	☐	2000	DO 10	☐	500	DO 18	☐	500
DO 3	✓	500	DO 11	☐	500	DO 19	☐	500
DO 4	☐	500	DO 12	☐	300	DO 20	☐	500
DO 5	☐	500	DO 13	☐	500	DO 21	☐	500
DO 6	☐	500	DO 14	☐	500	DO 22	☐	500
DO 7	☐	500	DO 15	☐	500	DO 23	☐	500

Figure 17: Port - properties

The basic technical settings can be set up in the dialogue *port - properties* (*figure 17*). The possible modifications are:

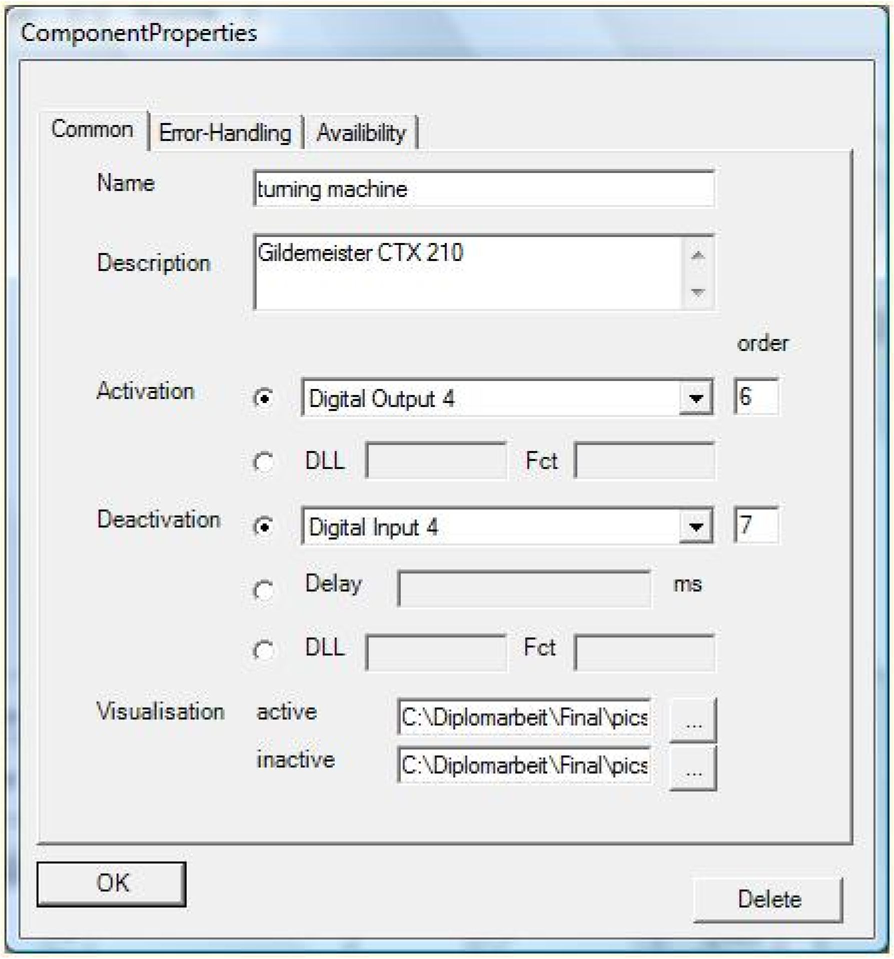
Physical Channel It is necessary to adapt these points to the user's NI DI/DO Card. Every channel has eight ports. There are cards with just 8 or 16 ports; here, the user just enters the physical channel address for the first and the second one listed in the manual of the card. Also, if more than one card is installed on one computer, this property must be adapted as well as if it is the same procedure for the output channel.

Invert channels In general, the activation of a component is started when the channel signal is switched from 0 to 1, but some machines give the signal the other way round. The component is kept inactive by choosing the signal 1 and is working by the decreasing of the signal level to 0. This can be achieved by marking the checkbox of this specific port.

Pulse time The last property is the pulse time. It determines how long the level is kept to 1 (or 0; see above). For special applications, it is sometimes necessary to prolong or shorten the period.

2.5.6 Adding a component

A component can be added to the data set by clicking the *New* button, which is positioned over the process list. A dialog called Component properties (see *figure 18*) will be opened. This is the same dialogue that appears after a double click on a component in the visual panel. In this case, the dialogue is used to edit the component properties.


ComponentProperties

Common | Error-Handling | Availibility

Name: turning machine

Description: Gildemeister CTX 210

Activation: (●) Digital Output 4, order 6; ( ) DLL, Fct

Deactivation: (●) Digital Input 4, order 7; ( ) Delay ms; ( ) DLL, Fct

Visualisation: active C:\Diplomarbeit\Final\pics; inactive C:\Diplomarbeit\Final\pics

Figure 18: Component - properties 1

The dialogue in question contains the attributes for a component. The input can be finished by clicking on the *OK* button. The property sheets contain three tabs, *Common*, *Error - Handling* and *Availability*.
The attributes are listed in detail in *table 1*.
The second tab contains the elements for the error handling. Generally speaking, there are three ways of handling an occurred error which are shown in *table 2*.

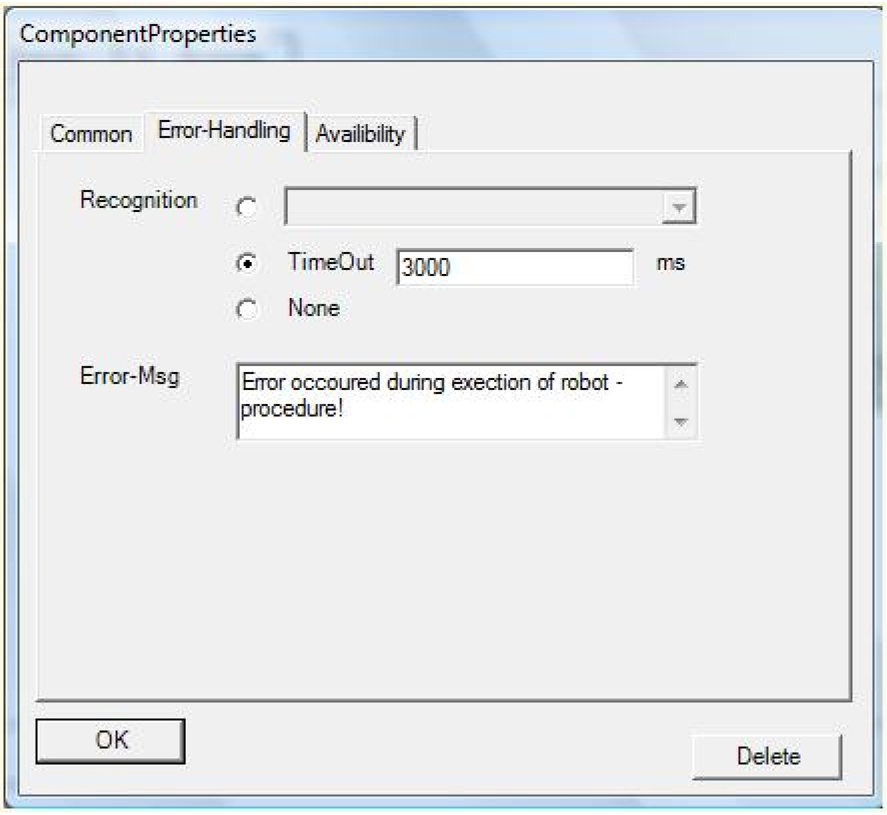
The third sheet (see *figure 20*) represents the availability configuration. It allows to define conditions that must be fulfilled so that the system may recognise the component as free during the runtime mode. This is necessary for conveyor belts, for example. It

Name	Defines the name of the component. This string will be used in the process list.
Description	Further information about the machine used can be saved in this area. For example, the exact type data or special data about the usage.
Activation	Defines the interface which is used for the activation. Next to the combo box, the position of the component in the process order is listed. It can be entered directly via the position number. It will be inserted at the given position. Later reordering can be done via this field or via the up/down - buttons in the process order list. The data are automatically synchronised. The activation can be done by a digital output or via a function in an DLL file written by the user. For this reason, two text fields are intended. The first one contains the file and the class in the file (eg. *dllfile.Class1*) separated by a dot. A file can contain more than one class. The second field contains the name of the function (eg. *robotact*). All these data are case sensitive. This means that it is important to pay attention to writing uppercase letters in these fields. The functions of the dll files are functions with no return values (respective *void* in C or C#) and no parameters.
Deactivation	Defines the interface for deactivation. The order position has the same behaviour like the one in the activation area. Alternatively to the possibility of defining a digital input for the recognition of the finish status the user can define a specific *delay* time. The component is started. The system waits for the time defined and will then continue with the sequence. The unit for this attribute is ms. When checking the option *Delay*, the order position field becomes inactive. The activation can be triggered either by a digital output or via a user written function in a DLL file, too. The conditions are the same like in the activation area. The function has no parameters but a return value of the type *integer*. When the return value becomes 1, the sequence is continued. The system executes the function continiously until the return value becomes 0.
Visualisation	This attribute contains the path of the pictures which are shown in the visual panel. One for the active and one for the inactive status.

Table 1: Attributes: Component

Recognition via DI	The combo box allows to select an Digital Input Port which tells the control system when an error occurred by setting the signal to 1.
Recognition via Timeout	When the time between the start (attribute Activation) and the finish (attribute Dectivation) is longer than the given timeout an exception with an error message will be shown and the execution will be stopped.
No handling	The option *None* ignores errors.
Error - Msg	This text area contains the message that is shown when the error occurs.

Table 2: Attributes: Error - Handling

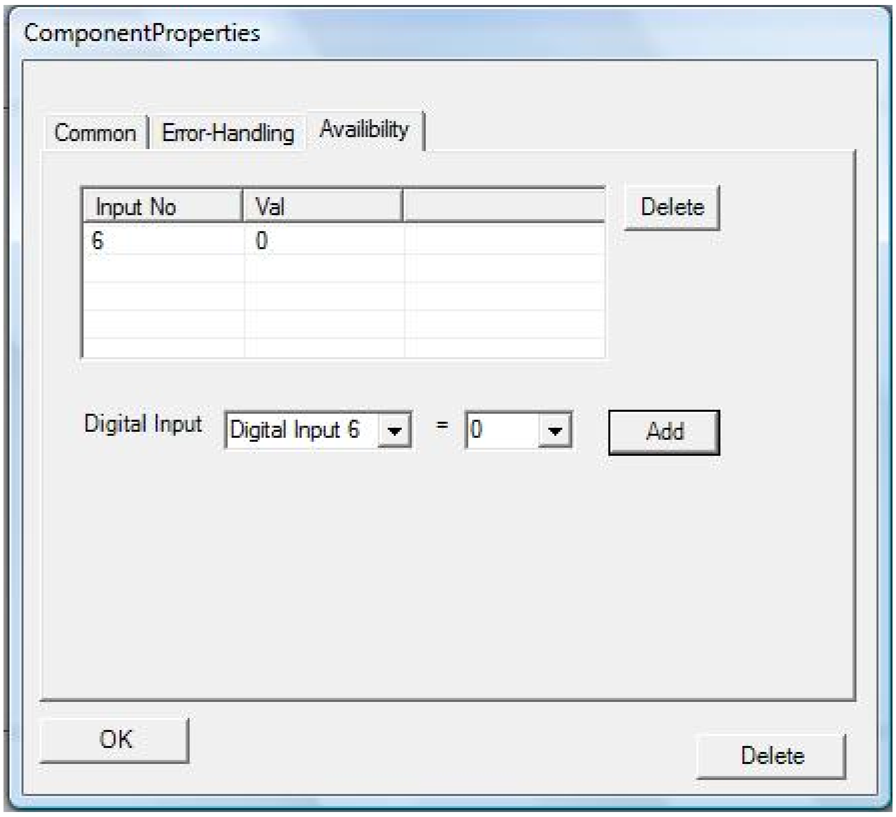


Recognition ( ) [✓ TimeOut] 3000 ms ( ) None

Error-Msg: Error occoured during exection of robot - procedure!

Figure 19: Component - properties 2

must be ensured that the position after the belt line is free for using it.
The first combo box in the mask allows for selecting the input channel, the second represents the value . By clicking the Add Button a condition, can be added.



Input No	Val	
6	0	

Digital Input: Digital Input 6 = 0

Figure 20: Component - properties 3

The basic necessity of this function results from the possibility of the system to execute more than one process at the same time (Mulit - threading[4]). So, many process procedures can be started. Consequently, it is important to determine when a component can be used by a process.

[4]Threads make it possible for a computer program to split itself and generates so further processes, which are running autonomously.

3 ProCO - Technical specification

This chapter will give insight into the technical background of the development of ProCO. It exclusively presents its basic functions. The whole source code has about 6000 lines of code. For detailed information, consider the source code itself.

3.1 Programming - framework

The realisation, of the software that was described in the last chapter is done in the programming language C#, which is a part of the Microsoft Visual Studio. In this study the Microsoft .NET Framework in the version 1.1 is used.
C# was released in the year 2000 and integrates concepts of Java, C, C++ and Delphi. It is a procedural, object oriented language. It tries to give a simple, modern and robust environment for developers.[Wik08a]
C# is based on components. All objects are described as components. They are the core elements of this language. [Wik08a]

3.2 Program architecture

The following subchapter describes the class architecture and the design of the code used. Within the scope of this study, I shall refrain from discussing too specific programming details. The subsequent enumeration is intended to provide a concise outline of the most important functions.

3.2.1 mainForm

This class determines the graphical user interface (GUI). It is responsible for the adjustment of the single control elements. Furthermore, it is designed to react to activities, such as mouse clicks, drag-and-drop operations or data input. The list below gives a short overview of the most important functions:

```
public void initializeListView()
```
Initialises the ListView for the process order. It adds the title columns and sets the size of it.

```
public void initializeLoopListView()
```
Initialises the ListView for loop management.

```
    private void visualPanel_MouseDown(...)
    private void visualPanel_MouseUp(...)
    private void visualPanel_MouseMove(...)
    private void visualPanel_Paint(...)
```

Manages the drag-and-drop operations: If the current mode is the design mode, the `MouseDown` function saves the current start position of the mouse pointer and identifies the component clicked on via the `getHit` function in the `ListController`. The function `MouseMove` is called when the mouse is moved in the visual Panel. First of all, it is checked if the left mouse button is pressed and if a component was hit in the `MouseDown` area. If both conditions are fulfilled, the difference between the start position and the current position is calculated and the values are refreshed in the component moved. The new starting point is the current point. This procedure is repeated for every movement. The `MouseUp` - function deselects the currently selected component.
The `Paint` function calls the `public void paintAll(Graphics g)` in the `ListController` class, so that it refreshes the visual panel. (see subchapter ListController).

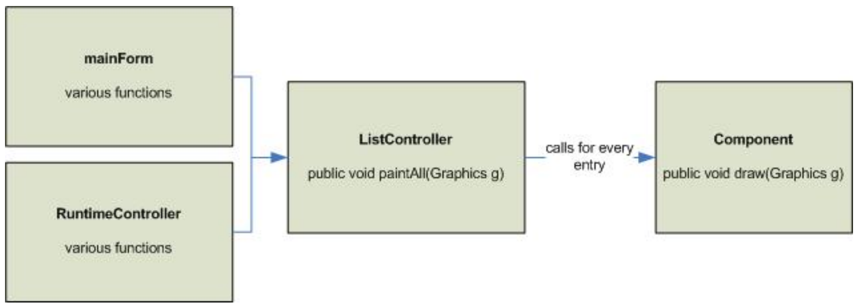


Figure 21: Usage of the draw - function

```
    private void Down_Click(...)
    private void Up_Click(...)
```

Specifies the behaviour of the re-ordering buttons at the process list. With these two buttons, the execution position of component can be easily changed. Furthermore, start-and-end positions of loops can be changed. All this can be done in the Component Property dialogue or in the Loop tab. In the process list, the user has one place for all rearrangement functions.

```
    private void Save_Click(...)
    private void Load_Click(...)
```

Manages the loading and saving of the created XML files.

```
private void loopListView_DoubleClick(...)
private void looptype1_CheckedChanged(...)
private void btnLoopSave_Click(...)
private void btnLoopNew_Click(...)
private void btnLoopDelete_Click(...)
private void loopListView_Click(...)
```

Manages the loop entries. The `new` - function creates a new loop, the `save` - function saves changes in an existing loop or saves a new loop while the `delete` - function removes a loop from the execution process.

```
private void btnNew_Click(...)
```

Creates a new component and shows it as a grey rectangle in the visual panel. The new item is listed in the process list at the end, with the start entry coming before the end entry. Now the user can change the properties of the new component by double clicking it.

```
private void btnRuntime_Click(...)
private void btnDesign_Click(...)
```

Switches between the two modes.

3.2.2 Component

This class manages the component properties and the arrangements of them in the visual panel.

```
public void draw(Graphics g)
```

Every `Component` "knows" how to draw itself. It loads the images for the activated and deactivated status. The activation status is checked by means of a member variable of the component class. This variable can be set by other classes.

```
public bool hit(Point p)
```

Returns true if the co-ordinates in the variable p hit the component in the visual panel. This function is needed to detect which component is dragged or double clicked in the visual panel in the design mode. If no component is found on the position `p`, the return value is false.

3.2.3 ComponentProperties

Manages the window `ComponentProperties`, which allows for the change of the states of a component. Most of the functions are self-explanatory and are used for the GUI management.
It has the following main functions:

- Filling all combo box lists.
- Read-out of the values of the fitting component out of the dataset.
- Filling all values into the text fields, combo boxes and radio buttons.
- Management of the Availability tab (delete and add functionality).
- Saving all changes if the OK button is clicked.
- Displacement of the current component if the Delete button is clicked.

3.2.4 IOControl

```
public IOControl(DataStructure ds, mainForm form)
```

The constructor initialises the IO card with the pre-cast functions of National Instruments. This is done by the following commands:

```
digitalWriteTask = new Task();
digitalWriteTask.DOChannels.CreateChannel(
       physicalChannelx,"",ChannelLineGrouping.OneChannelForAllLines);
```

This is done for every channel used.

```
public void SetCardO()
public void GetCardI()
```

The class holds a boolean array with the currant values for the input-and-output channels. These functions write these data from the array to the card outputs/inputs.
This is done by the following commands:

```
DigitalSingleChannelWriter writer = new
       DigitalSingleChannelWriter(digitalWriteTask.Stream);
writer.WriteSingleSampleMultiLine(true, output);
```

```
public int SetOutput(int no, int impulse)
```

Negotiates the value of the port under `no` specified output for the time indicated in `impulse`.

`public void ResetIO(DataStructure ds)`

Resets all output channels to the default values. It also provides the invert status of the specific port.

`public bool GetInput(int no)`

Returns the value of the in the variable `no` specified input port.

`public float checkingInput(int no, int msec)`

Watches an input port till it becomes a "true" value, which means 1 if the invert option is not checked or 0 if it is checked. Checking time is restricted to the time span msec. Thereafter, the function returns to a time-out error if the corresponding option is chosen in the Component Property dialogue.

`public void setPhysChan()`

This function reads out the addresses of the physical channels of the dataset and writes them into member variables of the `IOControl` class. These options can be set in the Port Property dialogue.

3.2.5 DataStructure

This class is responsible for the ERM management. It saves all data used in the software tool.

```
public void MakeComponentsTable(DataTable componentsTable)
public void MakeAvailibilityTable(DataTable availibilityTable)
public void MakeLoopTable(DataTable loopTable)
public void MakeParameterTable(DataTable parameterTable)
public void MakePhysChanTable(DataTable physChanTable)
public void MakePortTable(DataTable portTable)
```

Creates the data structure of the single tables, including the column types, as it is described in the chapter on ERM. Every column in every table is created by the `AddCol` function, which is described later, as follows:
eg.: `AddCol("StartPos", "System.Int32", loopTable);`

```
public void AddPhysChan(...)
public void AddAvailibility(...)
public void AddPort(...)
public void AddParameter(...)
public void AddLoop(...)
public void AddProcess(...)
```

Adds an entry to the specific table. The concrete values are given in the parameters.

```
public DataColumn AddCol(...)
```

Is a help - function for the Add functions listed above.

3.2.6 ListController

The main variable of this class is a component array. It holds all component entries used and manages them.

```
public void Add(Component c)
```

Adds a component to the array.

```
public void paintAll(Graphics g)
```

Calls the draw function of every component entry in the array.

```
public void showDialogComponent(...)
```

Opens the component property dialogue.

```
public Component getHit(Point p)
```

This function checks every component in the array if the position (coordinates in the visual panel) `p` hits it. This is done by calling the function `public bool hit(Point p)` in the `Component` class. If a fitting component is found, the component is returned. Otherwise `null` is returned.

3.2.7 Port

Holds the data for the port entries.

3.2.8 PortProperties

Manages the window `PortProperties`, which allows for changing the states of the ports. To be more precise:

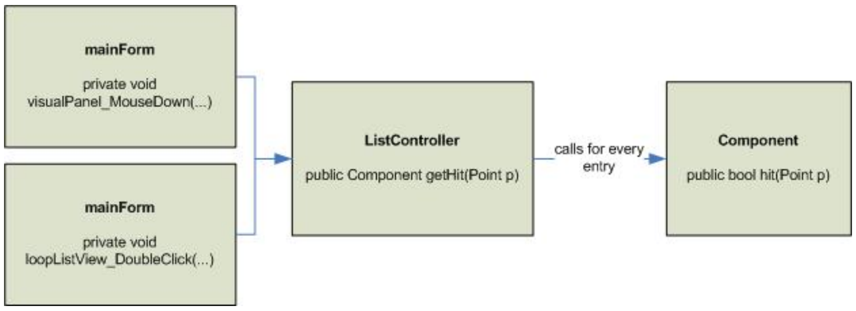

mainForm
private void visualPanel_MouseDown(...)
mainForm
private void loopListView_DoubleClick(...)
ListController
public Component getHit(Point p)
calls for every entry
Component
public bool hit(Point p)

Figure 22: Usage of the getHit - function

- the physical channels,
- the impulse time and
- the invert status.

Most of the functions are self-explanatory and are used for the GUI management.

3.2.9 ProcessListController

`public void refreshOrder(...)`

Refreshes the order of the process list when a component gets another process position. This is a very complex function. For the concrete realisation, consider the source code of ProCO.

`public void fillProcessList()`

Reads the data out of the data structure and fills the process order list, as shown in the chapter on *Design - mode*.

`public void fillLoopListView()`

Fills the loop list with the fitting values.

3.2.10 RuntimeController

`public Thread startThread(RuntimeController rtc)`

This function is executed when the start button is pressed in runtime mode. It is managed as a thread. Thereby, it is possible to start more than one process threads at the same time. (see *figure 23*)

`public void startProcessing()`

This function is indirectly called by the startThread function. It calls the executeSteps -

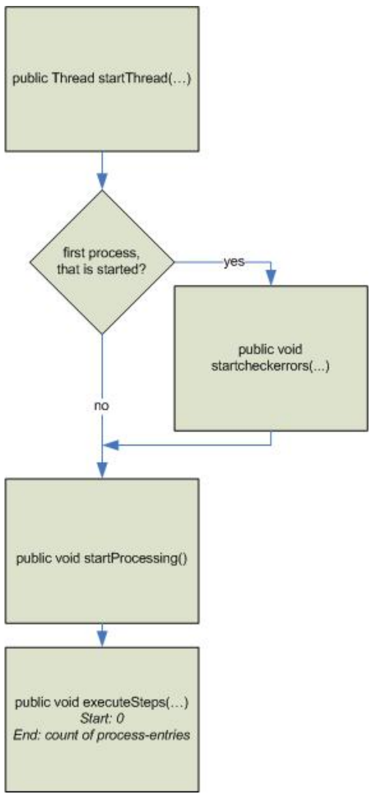

public Thread startThread(...)
first process, that is started?
yes
public void startcheckerrors(...)
no
public void startProcessing()
public void executeSteps(...)
Start: 0
End: count of process-entries

Figure 23: Starting of a new run

function with the start value 0 and the end value as the length of the process list. In this way, it executes the whole process list. (see *figure 23*)

`public void executeSteps(int start, int end, int count, int loopID)`
This function carries out the steps that lie between the start-and-end position for the times given in the variable `count` . This architecture is necessary to realise a recursive procedure which is used to execute any count of loops within loops that lie in other loops themselves and so on. (see *figure 24*)
The single process steps are performed. Before start entries are executed, it is checked if the component is “free” by means of the function `checkfree`. If it is free, it is allocated. If not, the procedure will wait in a loop until `checkfree` returns true and a member variable `waiting` is set with the component ID for that component for which one is waiting. Thereby, other processes are able to recognise if a process exists that has been started earlier and that is waiting for the same component. In this case, this process will have priority. The DLL invocation is triggered by the following code. First of all, an instance of the class which is identified by the string in the variable `FileClassName` is created; thereafter, the function (String `MethodName`) is called:

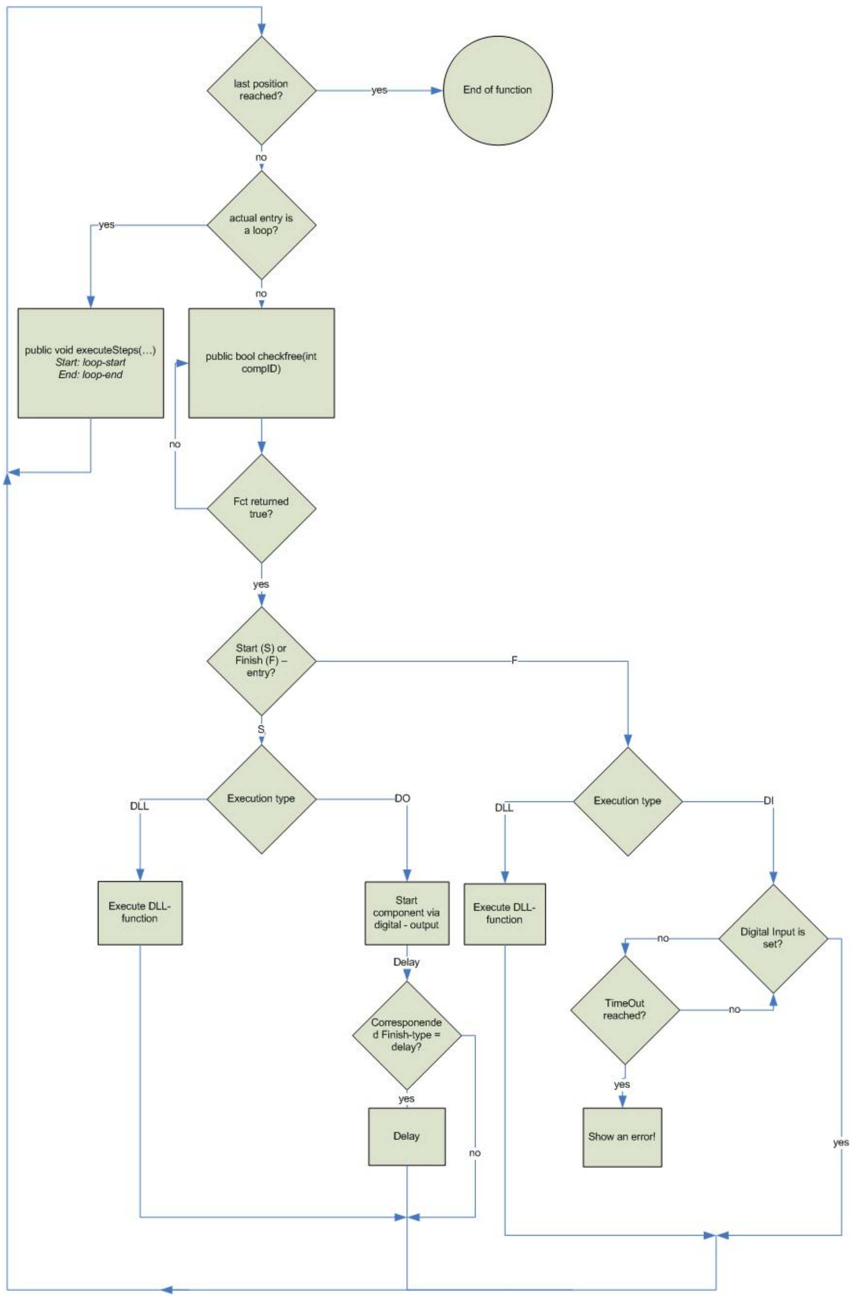


Figure 24: Usage of the executeSteps - function

```
object calcDisp = null;
Type calcObj=null;
calcObj = Type.GetTypeFromProgID(FileClassName);
calcDisp = Activator.CreateInstance(calcObj);
calcObj.InvokeMember(MethodName,
BindingFlags.InvokeMethod, null, calcDisp, null);
```

```
   public bool checkfree(int compID)
```

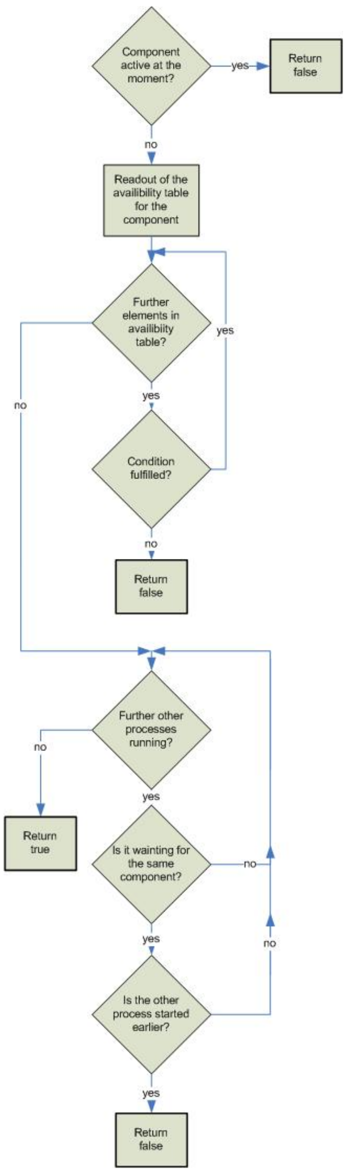

Component active at the moment?
yes
Return false
no
Readout of the availibility table for the component
Further elements in availibility table?
yes
no
yes
Condition fulfilled?
no
Return false
Further other processes running?
no
Return true
yes
Is it wainting for the same component?
no
yes
no
Is the other process started earlier?
yes
Return false

Figure 25: Checkfree - function

This function checks if a specific component is “free”. It checks if the component is used at the moment, as well as if the conditions which are given in the availability table are fulfilled.
Furthermore, this function checks if another process exists that was started before this process. In this case, the process, that was started first has priority. That is why every process has a member variable `runID`. This variable identifies in which order the processes were started. (see *figure 25*)

```
public void checkingErrorPorts()
```

It is called as a thread by the function `startcheckerrors` when the first process is executed. It observes the ports that are defined in the single component properties in the error handling tabs in the part *Recognition*. If an error occurs, the execution will be stopped and a message will be shown.

```
public void startcheckerrors(RuntimeController rtc)
```
See above.

3.3 ERM - Entity Relationship Model

In the *figure 26*, the Entity - Relationship - Model of ProCO is depicted. In this notation, the attributes are primary keys, which means that the value of these attributes is unique and the row can be identified by this value. They are marked with the tag *PK* and the parameter name is underlined. Foreign keys is taken to mean that the value refers to an entry in another table that is marked with the tag *FK*.

ParameterID	Contains the unique primary key for the parameter.
Name	Name of the parameter

Table 3: Attributes: Parameter

LoopID	Contains the unique primary key for the loop.
StartPos	Gives the start position of the loop in the process list.
EndPos	End position in the process list. The process steps between the start and the end position will be repeated for the specified number of times.
Type	Contains the type of the count for repetition. 0 means that the value given in the attribute *count* is used. 1 means that the parameter which is referred to in the attribute *ParameterID* is used for the number of repetition.
Count	see above.
ParameterID	see above.

Table 4: Attributes: Loop

The implementation is achieved with the `DataSet` type of the .NET framework. The hierarchical structure of this class is pictured in *figure 27*.

Each `DataSet` contains one or more `DataTables`. These `DataTables` contain `DataColumns` and `DataRows`. (see *figure 27*)
The data are not readout via conventional SQL statements. These are regularly used for

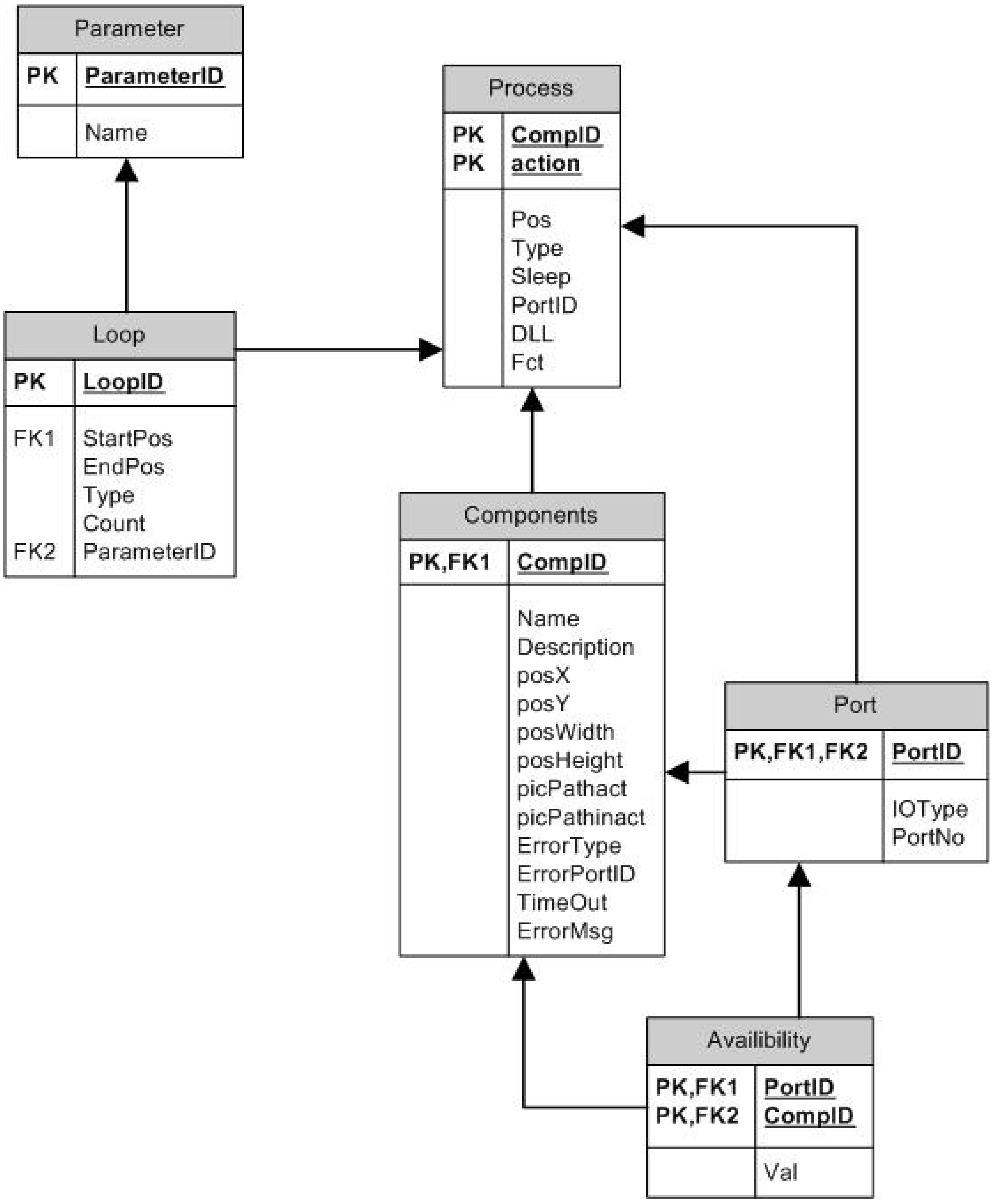


Figure 26: ERM

CompID	Contains the first part of the compound key for the process. It refers to the component which is involved in this process step.
action	Contains the second part of the compound key for the process. It contains one char: 'S' for start or 'F' for finish.
Pos	Position in the process list.
Type	This marker defines how the process step is started or finished. The possibilities are explained in the chapter *Adding a component*: digital IO, delay, DLL functions.
Sleep	see above.
PortID	see above.
DLL	Saves the information about the used DLL file. (see chapter *Adding a component*)
Fct	Saves the name of the function. (see chapter *Adding a component*)

Table 5: Attributes: Process

ParameterID	Contains the unique primary key for the parameter.
Name	Name of the parameter

Table 6: Attributes: Parameter

PortID	Contains the unique primary key for the port. In this table, all available ports are listed
IOType	0 for input port and 1 for output port.
PortNo	The number of the port.
Invert	In general, the activation of a component is started when the channel signal is switched from 0 to 1, but some machines give the signal the other way around. The components are kept inactive by assigning them the signal 1; they work by the decreasing of the signal level to 0.
Impulse	Gives the time, that the level (1 or 0; see above) is set.

Table 7: Attributes: Port

normal databases. The readout process is done by `select` commands which are a function of the `DataTable` class. The contents of the parameter for this function are similar to the `where` statement in a SQL statement. The result (return value) of the function is a `DataRow` Array (`DataRow[]`) with the entries fitting the select statement.
eg. `DataRow[] Iports = portTable.Select(''IOType = 1'');`

CompID	Contains the unique primary key for the component. A component in this context can be every element in the production process. Eg. robots, conveyor belts, turning machines ...
Name	This name is shown later in the process list in the main mask.
Description	This attribute contains further information about the component. eg. detailed model information ...
posX	Saves the information about the position in the visual panel in the main mask.
posY	see above.
posWidth	Saves the information about the dimensions of the corresponding picture.
posHeight	see above.
picPathact	Represents the path of the corresponding picture that is shown, when the component is active during the execution of the process in the runtime mode.
picPathinact	Path of the picture that is shown when the component is inactive and during the implementation in the design mode.
ErrorType	Gives the mode of error handling. 1 stands for error recognition via an input channel. When an error occurs, the component sends a signal to the host computer. When 1 is chosen, the software throws an exception with the text given under the attribute *ErrorMsg*. When 2 is chosen, the software recognises an error via a timeout, which means that an error is messaged if the component does not send a finish signal during the time specified under the attibute *TimeOut* (in ms). 3 means that no error handling shall be effected.
ErrorPortID	see above.
TimeOut	see above.
ErrorMsg	see above.

Table 8: Attributes: Components

The values of the selected items can be easily changed by editing the contents of the entries:
eg. `Iports[0][PortNo] = 3;`
The mapping of relations is pronouncedly different from SQL mapping. The `DataTable` class has a further sub class, viz. `DataRelation`. The creation of such a relation is done in the following way:

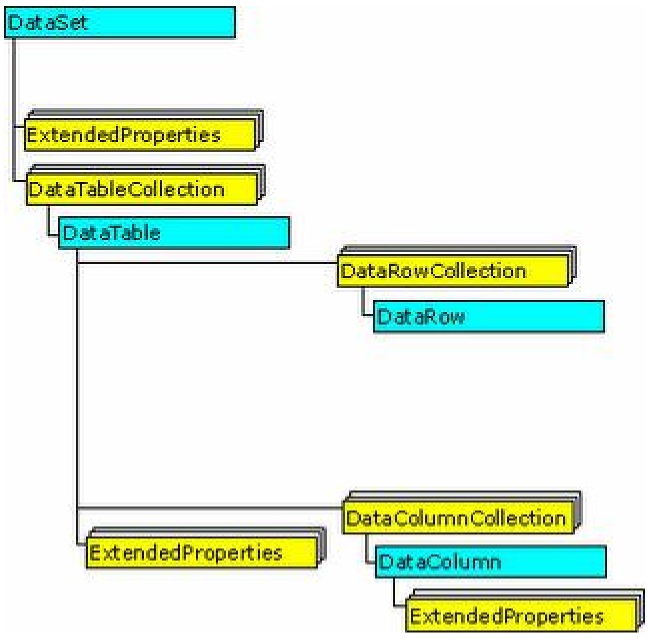


Figure 27: DataSet [ee08]

e.g. `DataRelation dr = new DataRelation(`

```
''CompPort'',   // user - defined name of the relation
ds.Tables[''Port''].Columns[''PortID''],  // higher - level table
ds.Tables[''Components''].Columns[''ErrorPortID''] // lower - level table
);
```

The child - or parent - rows can be selected like this:

`DataRow[] drPort = drComp.GetChildRows(ds.Relations[''CompPort'']);`

In this example, `ds` is the `DataSet` and `drComp` is an instance of the table `Components`.

3.4 XML

XML (eXtensible Markup Language) is a markup language - a language that describes how a text is structured or formatted. It is not dedicated to one special purpose. It can be used in various cases. Furthermore, it allows for extending it with personally defined elements. The main benefit is to create a framework for structured data. An XML document just contains characters and no binary data; thus, it is “humanly legible”. So hierarchical and structured data can be saved and easily exchanged between different systems. Moreover, it is a fee - free open standard.[Wik08b]

XML has two levels of correctness:

- Well-formed

  A document is well formed provided that it conforms to all XML syntax rules. An error in this context would be a start tag without an end tag.

- Valid
  A document is valid if it conforms to special semantic rules. These are mostly user defined. An error in this context would be an undefined element in the document contained. [Wik08b]

The structure of an XML document is quite simple. All data are enclosed by a start and an end tag.

eg. `<name>Frank Miller<\name>`

Every element can contain further sub-elements that have the same structure. These structures can be seen as a kind of tree. The most important thing in this context is that there is only one root element (also called document element).
A more complex example with a hierarchical structure is the following one. It is an example of a simple telephone directory:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
 <phonebook>
      <entry>
           <name>Frank Miller</name>
           <no>+41 1 456789012</no>
           <no>+41 1 345678901</no>
      </entry>
      <entry>
           <name>Pat Huber</name>
           <no>+41 34 45679012</no>
      </entry>
</phonebook>
```

C# supports the XML concept on a high level. When the data is saved by the means of the data type `DataSet`, which is a class of the .NET framework, the element can be easily saved by the command `DataSet.WriteXml(''filename.xml'')`. The equivalent is the command `DataSet.ReadXml(''filename.xml'')`.

The concrete realisation of the model described in the chapter *ERM* looks like the following listing:

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Components>
    <CompID>0</CompID>
    <Name>line 1</Name>
    <Description />
  <posX>215</posX>
  <posY>16</posY>
  <posWidth>142</posWidth>
  <posHeight>63</posHeight>
  <Pathact>C:\Diplomarbeit\Final\pics\con2_act.gif</Pathact>
  <Pathinact>C:\Diplomarbeit\Final\pics\con2_inact.gif</Pathinact>
  <ErrorType>2</ErrorType>
  <TimeOut>3000</TimeOut>
  <ErrorMsg />
</Components>
...
  <Process>
  <CompID>0</CompID>
  <Action>F</Action>
  <Pos>10</Pos>
  <Type>0</Type>
  <Sleep>0</Sleep>
  <PortID>0</PortID>
</Process>
<Process>
  <CompID>0</CompID>
  <Action>S</Action>
  <Pos>9</Pos>
  <Type>0</Type>
  <Sleep>0</Sleep>
  <PortID>24</PortID>
</Process>
...
<Loop>
  <LoopID>0</LoopID>
  <StartPos>5</StartPos>
  <EndPos>8</EndPos>
  <Type>0</Type>
  <Count>4</Count>
</Loop>
...
<Port>
  <PortID>1</PortID>
  <IOType>0</IOType>
  <PortNo>1</PortNo>
  <Invert>0</Invert>
  <Impulse>500</Impulse>
</Port>
...
<Parameter>
  <ParameterID>1</ParameterID>
  <Name>Throughput</Name>
```

```
</Parameter>
...
<PhysChan>
  <ChanID>1</ChanID>
  <Text>Dev1/port0/line0:7</Text>
</PhysChan>
...
```

In fact, the whole ERM is represented in the XML file. This listing is not the whole file. It is intended to give a first impression of the file.

4 Exemplary Realisation

4.1 Configuration

The system, described below consists of the following parts:

- Conveyor belts
- Robot (ABB IRB 2000)
- Turning machine (Gildemeister CTX 210)
- (Rod loader)

The system can be expanded in any order, e.g. by further turning machines or robots. The implementation - steps for new components remain the same. So, for this study it is enough to integrate just each one of these standard - components. A first impression can be gained from the *figures 28* and *29*.

4.2 Event Order

The process is very simple. An empty pallet is fed into the system at the position no. 1 (see *fig. 28*). The pallet is brought by the conveyor belts to position no. 2. Now, the turning machine begins to produce the turning work pieces. The raw - material comes from the rod loader. It is controlled by the turning - machine via M - commands. The finished pieces are taken out of the extraction place of the turning machine by the robot. The exact extraction - process will be described later in this sub - chapter.
Every pallet has four storage areas for finished work pieces. These areas can be changed easily, depending on the work piece dimensions and the accessibility for the robot. If the determined number of products is produced, the pallet will be brought via the conveyor belts to the start position no. 1 again.

A graphical outline of the Event - order is shown in *figure 31*.

4.3 DI/DO - Card

The physical control of the components is executed in this example via Digital Input / Digital Output Channels. On closer inspection, a PCI - card of the leading enterprise *National Instruments* is used.
The model NI 6528 provides 24 digital input und 24 digital output channels. The whole device is software configurable. It can be controlled by special software or by various

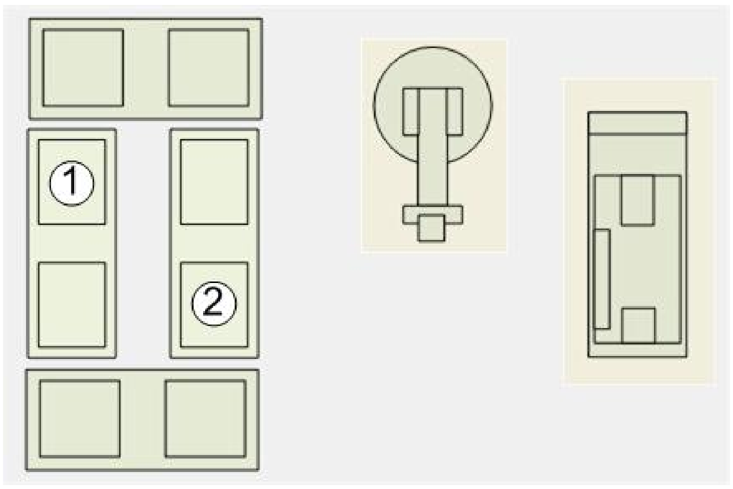


Figure 28: Schematic concept: FMC

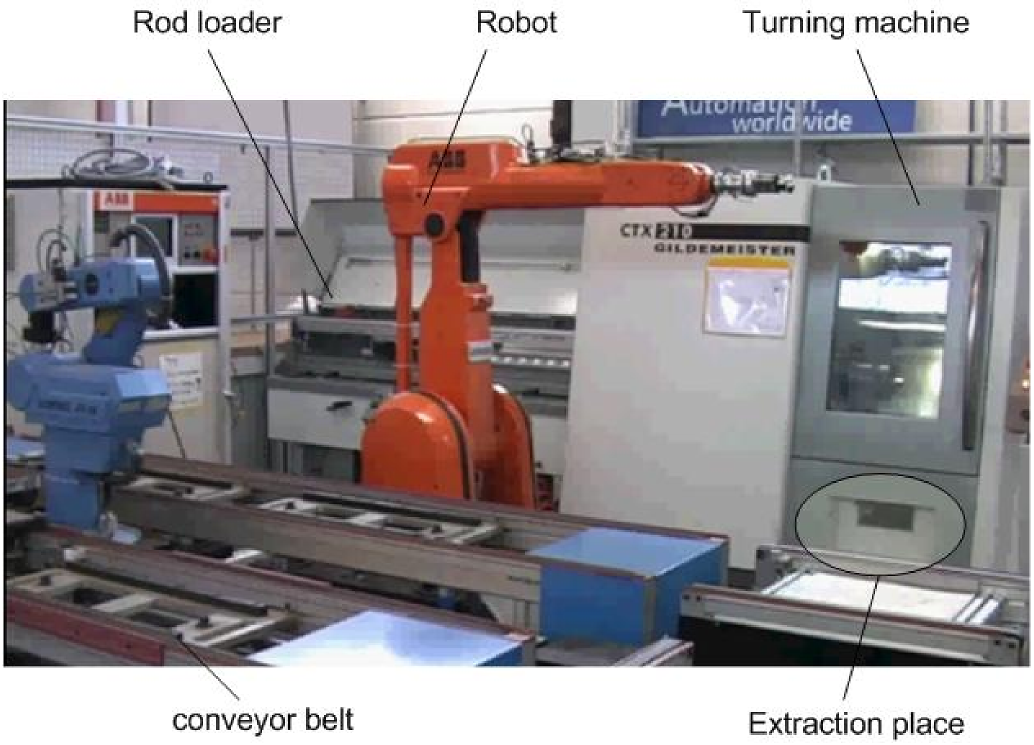


Figure 29: FMC

Figure 30: Palett

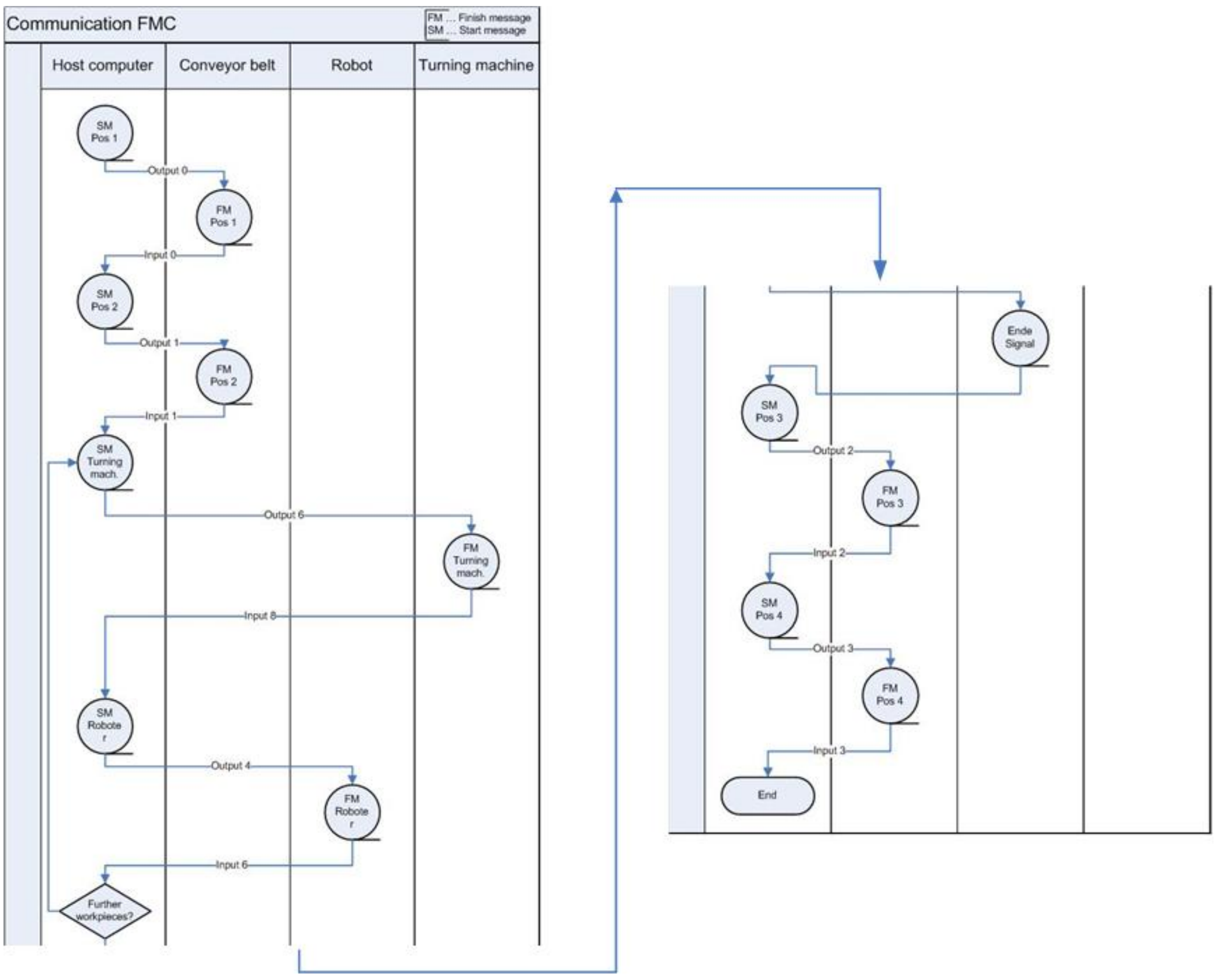


Figure 31: Event Order

progamming - languages like Visual Basic, Visual C++ or Visual C#.

In *figure 32*, the pinout is depicted. The input channels must be set in the following way: The pin with the higher voltage must be connected to the PX.Y+ pin and the one with the lower voltage to the PX.Y- pin. For the output channels, it is not important which one of the cables provides higher and lower voltage.[Ins07]
A cable leads from the PCI card to two panels, one with the pins 1 to 50 for input channels, the other one with the pins 51 to 100 for the output channels. Pins 49 and 99 are used for a 4.5 to 5.25V power supply from the computer. These pins are not relevant for this examination. All pins, except for 50, 100, 49 and 99 are isolated.[Ins07]
From -60 VDC till 1 VDC the incoming signal is stated as a low level voltage, from 3.2 VDC till 60 VDC it is stated as a high level voltage. In this context it is operated with 0 VDC and 20 VDC. Voltages over 60 VDC or under -60 VDC should not be exceeded. The result could be detrimental.[Ins07]

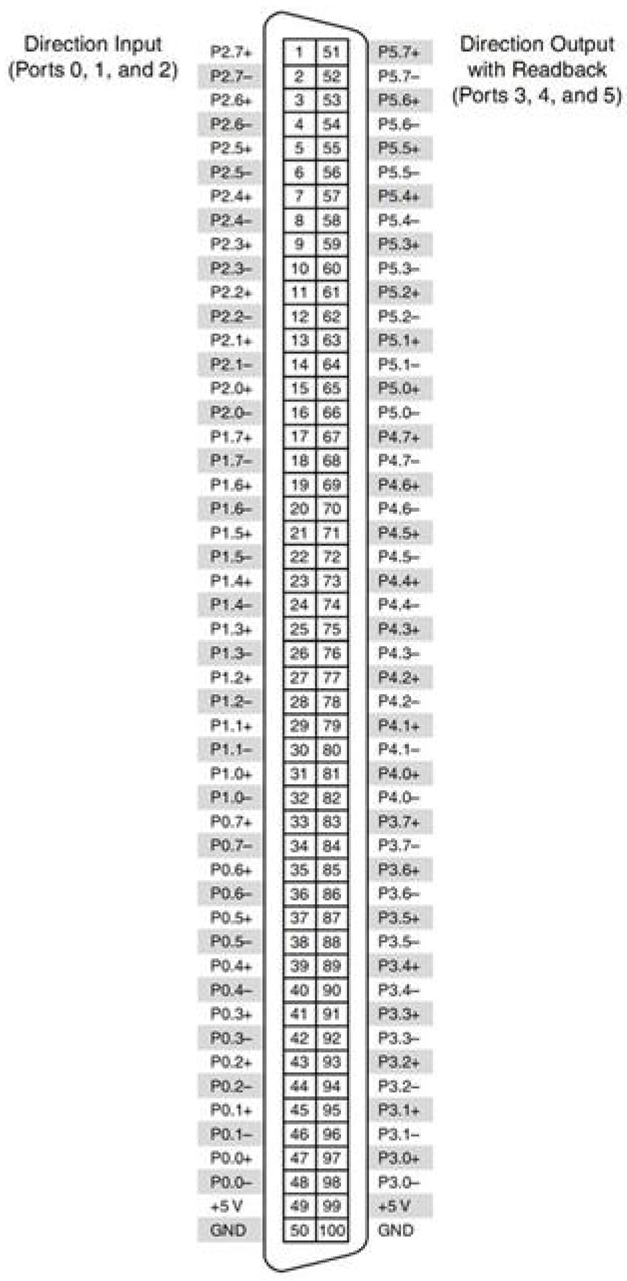


Figure 32: NI 6528 Pinout [Ins07]

The concrecte pinout for the components used on the panel is as follows (*figure 9* and *figure 10*):

P0.0- P0.0+	Conveyor belt: Line 1
P0.1- P0.1+	Conveyor belt: Line 2
P0.2- P0.2+	Conveyor belt: Line 3
P0.3- P0.3+	Conveyor belt: Line 4
P0.4- P0.4+	Conveyor belt: Error - State
P0.5- P0.5+	Conveyor belt: Emergency stop
P0.6- P0.6+	Robot
P1.0- P1.0+	Turning machine

Table 9: Pinout: Input

P3.0- P3.0+	Conveyor belt: Line 1
P3.1- P3.1+	Conveyor belt: Line 2
P3.2- P3.2+	Conveyor belt: Line 3
P3.3- P3.3+	Conveyor belt: Line 4
P3.4- P3.4+	Robot
P3.6- P3.6+	Turning machine

Table 10: Pinout: Output

4.4 Robot

4.4.1 Common aspects

As mentioned above, an ABB - robot is used in this manufacturing cell. It can be programmed via a hand-held unit. The common way to determine the position is via teach - in. That means that the robot arm is moved manually by the hand-held unit to the desired position and this position will be saved. Every `POS` command, which is described later, has six values saved internally: three for the position of the three axes and three for the orientation of the robot arm. This model of the robot has 16 in- and output ports for the communication with a host computer.

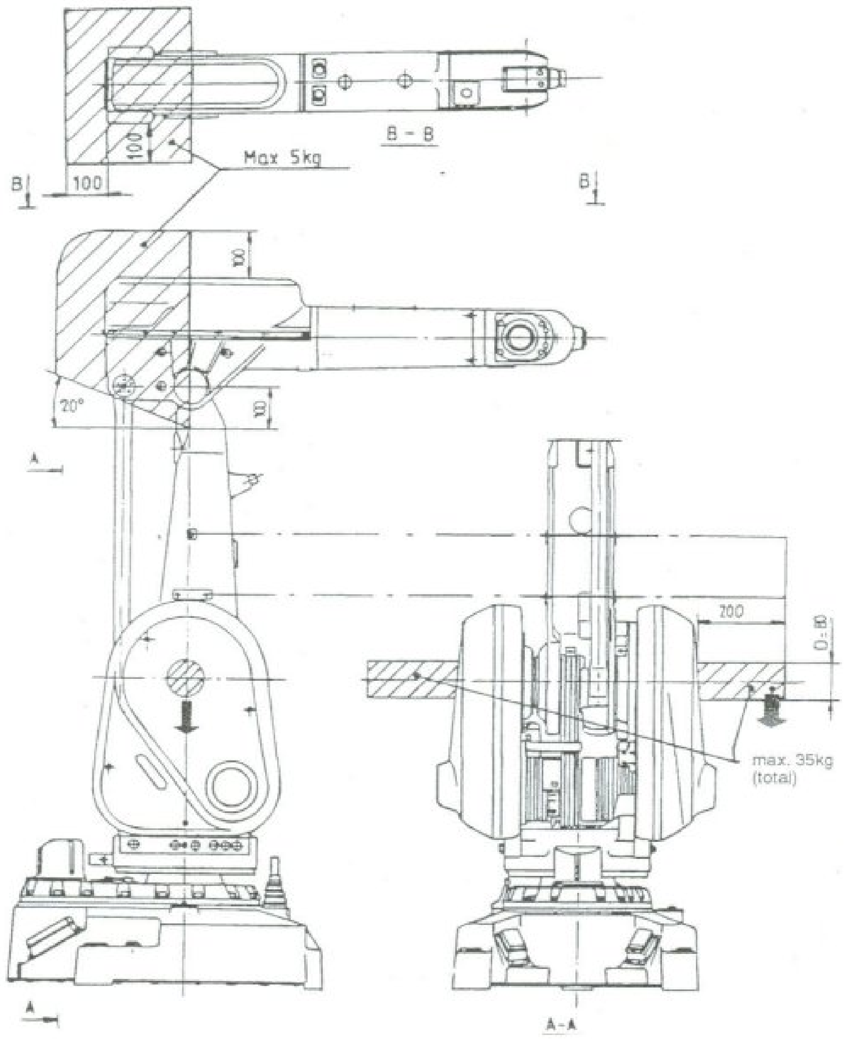


Figure 33: ABB IRB 2000[Geb08]

The exact specification of the robot as per the nameplate:

ABB IRB2000

L8941.5019-001
(7723220)
3x380V / 50Hz / 3,1 kW
IO Card: DSDXB001

In the rack on the right hand side card no. 1 is situated, on the left hand side card no. 2. Both cards are connected by ribbon cable to the terminal block of the control cabinet

4.4.2 Robot program

The listing below is the code used for this application.

```
10 V=2000 mm/s MAX=2000 mm/s
20 ROBOT KOORD
30 TCP 0
40 KOORDV 0
50 SETZE R1=0
70 POS V=50%
80 LÖSCHE AUSG6 VERZÖG 0.5 s
90 WARTE BIS EING9=1
100 LÖSCHE AUSG10 VERZÖG 0.5 s
120 SETZE R1 = R1 + 1
130 SPRUNG 150 WENN R1 <> 5
140 SETZE R1 = 1
150 POS V = 50 %
160 KARTES KOORD
170 POS V = 10 %
180 POS V = 10 %
190 POS V = 4 %
200 POS V = 2 %
202 WARTE 0.5 s
210 WARTE 0.5 s
220 POS V = 5 %
221 WARTE 0.5 s
230 SETZE AUSG6 VERZÖG 0.5 s
240 POS V = 10 %
250 POS V = 1 %
260 POS V = 1 %
270 POS V = 1 %
280 LÖSCHE AUSG6 VERZÖG 0.5 s
290 WARTE 1.5 s
300 POS V = 10 %
310 POS V = 20 %
320 POS V = 10 %
330 POS V = 50 %
340 SPRUNG 380 WENN R1 = 1
350 SPRUNG 460 WENN R1 = 2
360 SPRUNG 530 WENN R1 = 3
370 SPRUNG 620 WENN R1 = 4
380 POS V = 20 %
390 POS V = 10 %
400 POS V = 10 %
410 POS V = 10 %
420 SETZE AUSG6 VERZÖG 0.5 s
430 WARTE 1 s
440 POS V = 10 %
450 SPRUNG 690
460 POS V = 25 %
470 POS V = 5 %
480 POS V = 5 %
490 SETZE AUSG6 VERZÖG 0.5 s
500 WARTE 1 s
510 POS V = 5 %
520 SPRUNG 690
530 ROBOT KOORD
540 POS V = 30 %
550 POS V = 20 %
560 POS V = 5 %
580 SETZE AUSG6 VERZÖG 0.5 s
590 WARTE 1 s
```

```
600 POS V = 5 %
610 SPRUNG 690
620 ROBOT KOORD
630 POS V = 15 %
640 POS V = 5 %
650 SETZE AUSG6 VERZÖG 0.5 s
660 WARTE 1 s
670 POS V = 5 %
680 SPRUNG 690
690 SETZE AUSG10 VERZÖG 0.5 s
710 SPRUNG 70
PROGRAMM ENDE
```

10 Determines the standard and the maximal moving speed of the robot arm. The following statements, like `POS V = 5 %` refer to this standard speed.

20 In this programme two different co-ordinate systems are used: `ROBOT` and `KARTES`. The Cartesian system has the advantage that the moving between two points is done on a direct line. The robot co-ordinate system moves faster and more efficiently, but the exact moving activities are not exactly predictable. As a result, it can be dangerous to use this mode when obstacles are near to the robot arm.
`KOORDV` = Coordinate shift.

30, 40 TCP = Tool - Center - Point. The whole programming is done with teach - in. So the zero shift is not important in this context and is, therefore set to zero.

50 `SETZE` = set. This command is used to set a variable (register). In this case the register 1 is initialised to `0`. Register 1 is used to save the index of the next work - piece that will be placed on the pallet. All in all, the pallet assumes four positions in this case.

70 `POS` = move the arm to a specific position. The position saved is given via tech - in. The position at line 70 is the basic start position. It moves to the point with 50 percent of the standard - speed.

80 `LÖSCHE` = delete. It means to set the digital output (= `AUSG`) with the index, that follows this statement (6). `VERZÖG` (= delay) gives the waiting time after the output channel is set in seconds (`0.5 s`).
DO (= digital output) 6 in this context is used to open (= set, `SETZE`) or close (= delete, `LÖSCHE`) the gripper.

90 `WARTE` = wait. The robot waits until the digital input (= `EING`) reaches value 1. Then it continues with the execution of the batch. It is used to tell the robot to pick up the work - piece from the turning machine.

100 DO 10 is reset to 0. This channel is used to inform the host computer, that the work piece is put on the pallet, moving is completed and robot is ready to manage the next job.

120 Register 1 is increased. That means that the next index - position on the pallet is moved towards the following process.

130 When register 1 has the value 5, it is set back to 1. Otherwise, the next line (140) will be skipped.

150 The position directly in front of the turning machine is approached.

160 The following movements are done on a small area, viz. in the extraction place of the turning machine. So, for the reasons stated above the Cartesian system is chosen.

170 - 270 The robot gripper is positioned to the left of the work piece in the extraction place. The robot shifts the work piece slowly to the right, i.e. to a specific position. This is necessary to have a defined position of the product. The open gripper is moved to a position, so that the work piece is between the brackets of the gripper.
The delays are important to guarantee that the work piece is no longer in motion.

230 Now the gripper is opened.

280 Gripper is closed.

300 - 330 The robot arm is moved out of the extraction place. It is important to pay attention that there are no collisions. For this reason, more `POS` commands are necessary.

340 - 370 The position of the next place on the pallet is checked; it will be jumped to the specific position.

380 - 410, 460 - 480, 530 - 560, 620 - 640 The robot arm is moved above the depositing position on the pallet (four positions on the palett = four cases).

420f, 490f, 580f, 650f The grip is opened again. The robot waits for one seconds in order to make sure that the work piece has reached its final position.

440, 410, 600, 670 The arm is moved away from the pallet.

450, 520, 610, 680 The programme jumps to line 690.

690 DO 10 is set to 1. It tells the host computer that the moving process has finished.

710 The programme jumps back to line 60. Here it waits for the next start signal.

4.5 Turning machine

The turning machine used was produced by Gildemeister, type specification CTX 210. CTX is the indication for universal - turning machines.

Figure 34: Gildemeister CTX 210 [Gil08]

Technical specification:

- drive power: 7.5 kW
- range of speeds: 20 - 6.000 1/min
- Number of tools in turret: 12 pcs.
- Diameter of chuck: 165 mm

4.6 Work piece

Figure 35 illustrates the work piece that is produced in this exemplary realisation.

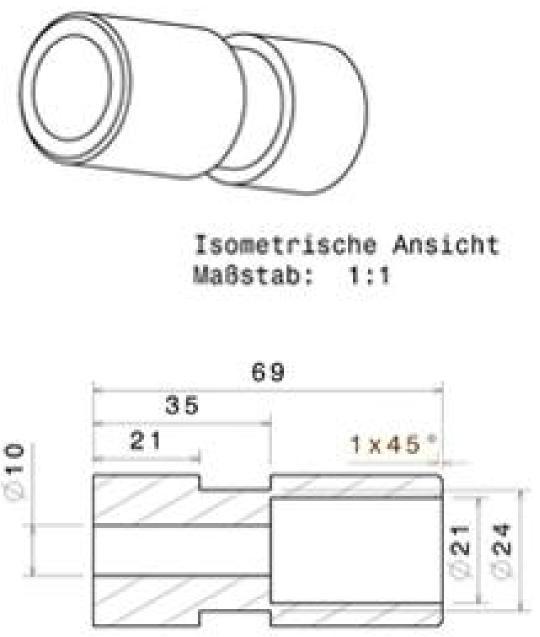


Figure 35: Produced workpiece

4.7 ProCO - configuration

This chapter describes the specific configuration that is necessary for the system to be realised:

Name	Activation	Deactivation	Error-Handling	Availibility	AP	DP
Line 1	DO 0	DI 0	DI 5	DI 0 = 0	1	2
Line 2	DO 1	DI 1	DI 5	DI 1 = 0	3	4
Turning machine	DO 6	DI 8	TimeOut: 240000	none	5	6
Robot	DO 4	DI 6	TimeOut: 60000	none	7	8
Line 3	DO 2	DI 2	DI 5	DI 2 = 0	9	10
Line 4	DO 3	DI 3	DI 5	DI 3 = 0	11	12

Table 11: Components

Components AP ... Position in the process - list for activation
DP ... Position in the process - list for deactivation

Loops The first loop with the index 0 repeats the starting of the turning machine and the starting of the robot four times. That is the count of the places on the pallet.

No	StartPos	EndPos	Count
0	5	8	4
1	1	12	Par1

Table 12: Loops

The second loop can be determined in the runtime - mode. It sets the count of repetition of the whole sequence.

No	Name	Action	Interface
LS	(1) LoopStart n=Par1		
1	Line 1	S	DO 0
2	Line 1	F	DI 0
3	Line 2	S	DO 1
4	Line 2	F	DI 1
LS	(0) LoopStart n=4		
5	Turning machine	S	DO 6
6	Turning machine	F	DI 8
7	Robot	S	DO 4
8	Robot	F	DI 6
LE	(0) LoopEnd		
9	Line 3	S	DO 2
10	Line 3	F	DI 2
11	Line 4	S	DO 3
12	Line 4	F	DI 3
LE	(1) LoopEnd		

Table 13: Process - list

Process - list This leads to the process list. The list is created automatically by ProCO, i.e. by the information given above.

ParameterID	Name
1	Throughput

Table 14: Parameters

Parameters

4.8 Start - up (Preparation)

It is important to start at first ProCO. Thereby, it is ensured that all output ports have a defined signal. Otherwise, it could happen that a machine starts immediately during the preparation process.
The next step is to arm the single components. Applied to the concrete example, this means to start the robot programme so that it is in the start position and waits for the start signal on the input channel defined. It is the same process as with the turning machine. Only if this step is made, the manufacturing should be started.
This must be done just once in a manufacturing period. More process cycles can be operated without restarting the system. Just when machines are switched off, e.g. at the end of the day, this procedure has to be started again before the next production - period begins.

5 Conclusion

The implementation of manufacturing cells is mostly a creative process. Especially, when existing components that have been used autonomously are connected to an automated system. If one did not paid attention to the fitness for use in manufacturing cells at the time of purchase, such problems arise. So, in this case individual solutions must be found or the components must be extended by appropriate modules. But the expenditures will be amortised within a fairly short period of time by shorter cycle times.

A further important issue is the fast practicability of such systems. Particularly for single orders with a high number of pieces, a temporary use of automated connections is an important factor. ProCO, the system developed, has the aim to provide a tool designed for such situations. It tries to help companies to raise their flexibility in the field of production.

References

[ee08] experts exchange.com. Datatable structure. *http://www.experts-exchange.com/Microsoft/Development/.NET/Q_23343849.html*, 20.3.2008.

[Geb08] Gebrauchtroboter.com. Daten abb irb 2000. *www.gebrauchtroboter.com*, 25.2.2008.

[Gil08] Gildemeister. Gildemeister ctx 210, produktspezifikation. *http://www.gildemeister.com/de,drehmaschinen,ctx210?opendocument*, 27.3.2008.

[Ins07] National Instrument. User guide, ni 6528. *http://www.ni.com*, 2007.

[Kie98] Hans B. Kief. Ffs-handbuch. *4. Auflage*, 1998.

[KR08] KUKA-Roboter. *www.kuka.com*, 15.2.2008.

[oA08] The University of Agder. Robot laboratory. *http://www.uia.no/en/portaler/om_universitetet/teknologi_og_realfag/ingenioervitenskap/mekatronikk/hovland*, 15.2.2008.

[Wik08a] Wikipedia. C sharp. *http://en.wikipedia.org/wiki/C_Sharp_20.3.2008.*

[Wik08b] Wikipedia. Xml. http://en.wikipedia.org/wiki/XML*, 25.2.2008.*